ARTIFICIAL LIFE, ADAPTIVE BEHAVIOR, AGENTS AND ANT COLONY OPTIMIZATION

Karthik Balakrishnan and
Vasant Honavar, chairs

Coverage and Generalization in an Artificial Immune System

Justin Balthrop
judd@cs.unm.edu
Computer Science Dept.
University of New Mexico
Albuquerque, NM 87131

Fernando Esponda
fesponda@cs.unm.edu
Computer Science Dept.
University of New Mexico
Albuquerque, NM 87131

Stephanie Forrest
forrest@cs.unm.edu
Computer Science Dept.
University of New Mexico
Albuquerque, NM 87131

Matthew Glickman
glickman@cs.unm.edu
Computer Science Dept.
University of New Mexico
Albuquerque, NM 87131

Abstract

LISYS is an artificial immune system framework which is specialized for the problem of network intrusion detection. LISYS learns to detect abnormal packets by observing normal network traffic. Because LISYS sees only a partial sample of normal traffic, it must generalize from its observations in order to characterize normal behavior correctly. A variation of the r-contiguous bits matching rule is introduced, and its effect on coverage and generalization is studied. The effect of representation diversity on coverage and generalization is also explored by studying permutations in the order of bits in the representation.

1 Introduction

The natural immune system uses a variety of evolutionary and adaptive mechanisms to protect organisms from foreign pathogens and misbehaving cells in the body. Artificial immune systems (AISs) seek to capture some aspects of the natural immune system in a computational framework, either for the purpose of modeling the natural immune system or for solving engineering problems. In either form, the fundamental problem solved by most AISs can be thought of as learning to discriminate between "self" (the normally occurring patterns in the system being protected, e.g., the body) and "non-self" (foreign pathogens, such as bacteria or viruses, or components of self that are no longer functioning normally). Almost any set of patterns that can be expressed as strings of symbols can be placed into this framework, for example, the set of normally occurring TCP connections in a local area network (LAN) and the set of TCP connections observed during a network attack [Hofmeyr, 1999, Kim and Bentley, 2001]. This is the example on which we will focus in this paper.

We are interested in the question of representation—how well a set of AIS detectors covers the set of normally occurring patterns (or conversely, how well it can detect the set of abnormal patterns). Because AIS detectors are typically generated on-line in a fluctuating environment, they are highly unlikely to be exposed to every possible normal pattern during training. Consequently, it is important for detectors to generalize from the set of observed normal patterns to the set of expected normal patterns. The generalization properties of the AIS affect both false positives (mistakenly identifying normal patterns as abnormal) and false negatives (mistakenly identifying abnormal patterns as legitimate). These are known as Type I and Type II errors respectively in the statistical decision theory literature.

There are several components of the AIS that affect how well it represents its environment and how well it generalizes. The first of these is the mapping from the domain to detectors, or what information is presented to the AIS. Here we will use the 49-bit compressed representation of TCP SYN packets, introduced by Hofmeyr [Hofmeyr, 1999, Hofmeyr and Forrest, 1999, Hofmeyr and Forrest, 2000]. In this representation each detector is a 49-bit string. Detectors are matched against the compressed 49-bit SYN packets (see Figure 1) using a partial matching rule which scores how closely they match. Choosing an appropriate mapping for a given problem in the AIS context has all the same complications as choosing a representation for a genetic algorithms problem. Some representations are clearly better than others, but it is difficult to formalize criteria by which one can choose a good one in a particular instance. The 49-bit representation chosen by Hofmeyr works surprisingly well, although it contains a minimal amount of information and the information is arranged in an arbitrary ordering.

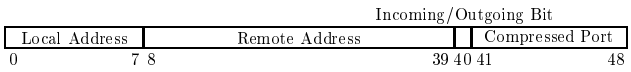


Figure 1: The 49-bit compression scheme used by LISYS to represent TCP SYN packets. Strings are compressed in two ways. First, it is assumed that one of the IP addresses is always internal, so only the final byte of this address needs to be stored. The port number is also compressed from 16 bits to 8 bits by re-mapping the ports into several different classes.

The second component is the match rule that is used to assess how well an AIS detector matches a particular pattern. A perfect *match* between a detector and a compressed SYN packet means that at each location in the 49-bit string, the symbols are identical. However, perfect matching (binding) is rare in the immune system and improbable between strings of any significant length. We use a matching rule known as r-contiguous bits [Percus *et al.*, 1993]. This rule looks for r contiguous matches between symbols in corresponding positions. Thus, for any two strings x and y, we say that $match(x, y)$ is true if x and y agree (match) in at least r contiguous locations. We also introduce a variant of this rule which we refer to as *r-contiguous templates*, or more simply, *r-chunks*. Both r-contiguous bits and r-chunks are related to genetic algorithms and classifier systems in interesting ways.

A third component is the permutation mask, also introduced by Hofmeyr. Permutation masks are a mechanism for introducing diversity of representation, crudely analogous to MHC diversity in the natural immune system. The idea behind this form of diversity is that different representations will match different patterns, and that the union of a set of different representations will have greater detection ability than any single representation. This insight is complicated by the form of our problem, in which detecting more patterns is not always better (because patterns detected in error lead to false positives). Permutation masks simply store a different permutation of the 49-bit mapping, one permutation for each detector set[1]. This, combined with r-contiguous bits matching, causes different permutations to discover different correlations among bits in the representation.

[1]Permutation masks are one possible means of generating *secondary representations*. A variety of alternative schemes are explored in [Hofmeyr, 1999].

2 LISYS

The following summary of LISYS is largely drawn from [Balthrop *et al.*, 2002]. LISYS is situated in a local-area broadcast network and used to protect the LAN from network-based attacks. In contrast with switched networks, broadcast LANs have the convenient property that every location (computer) sees every packet passing through the LAN. In this domain, *self* is defined to be the set of normal pairwise connections (at the TCP/IP level) between computers, and *non-self* is the set of connections, which are not normally observed on the LAN and are likely to be correlated with network intrusions. A connection is defined in terms of its "data-path triple"—the source IP address, the destination IP address, and the service (or port) by which the computers communicate [Mukherjee *et al.*, 1994, Heberlein *et al.*, 1990].

LISYS consists of sets of *detectors*, where each detector is a 49-bit string and a small amount of local state. The detectors can be distributed across multiple hosts, and they can perform their function with virtually no communication. The detectors assigned to a particular host are referred to as a *detector set.*

LISYS uses *negative detection* in the sense that valid detectors are those that fail to match the normally occurring behavior patterns in the network. LISYS generates random detectors, censors them against self, and eliminates those that match self (negative selection). The censoring process, known as the *tolerization period*, lasts for a few days during which time the detector is matched against every SYN packet occurring in the network. More efficient detector generation algorithms are described in [D'haeseleer *et al.*, 1996, Wierzchon, 2000, Wierzchon, 2001]. However, when generating detectors asynchronously for a dynamic self set, such as the network setting, these methods are not directly applicable and random generation seems to work well.

Detectors in LISYS have a finite lifetime. The expected lifetime of a mature detector is a parameter of the system. Detectors can die in several ways, through negative selection, old age, or lack of co-stimulation (see [Hofmeyr, 1999]). The finite lifetime of detectors, when combined with detector re-generation and tolerization, results in *rolling coverage* of the self set.

Each independent detector set has its own *permutation mask*, as described above. A permutation mask defines a permutation of the bits in the string representation of the network packets. Each detector set (network host) has a different, randomly-generated permutation mask. One feature of the negative-selection algorithm

as originally implemented is that it can result in undetectable patterns called *holes* [D'haeseleer *et al.*, 1996, D'haeseleer, 1996], or put more positively *generalizations* [Esponda and Forrest, 2002]. Holes can exist for any symmetric, fixed-probability matching rule, but permutation masks effectively change the match rule and thus the distribution of holes. Using a different permutation on each host allows us to control how much the system generalizes in the vicinity of self, and thus gives us more control over the undetectable holes [Esponda and Forrest, 2002].

The original LISYS system uses several other mechanisms, such as *activation thresholds*, *sensitivity levels*, and *co-stimulation* to reduce false positives, and *memory detectors* to increase true positives. For details on the full system, the reader is referred to [Hofmeyr, 1999, Hofmeyr and Forrest, 2000].[2]

3 Data Set

The experiments reported in this paper use the data set described in [Balthrop *et al.*, 2002]. Our data collection strategy was to control the data set as much as possible while still collecting data in a realistic context. The data set was collected from an internal restricted network of computers in a small university research group. The six internal computers in this network connected to the Internet through a single Linux machine that acted as a firewall, router and masquerading server for the internal machines. The internal network was set up as a broadcast network, so we were able to monitor the traffic of all the computers easily.

This scenario provided a data set that satisfied both objectives. The internal restricted network was much more controlled than the external university or departmental networks. In this environment, we can understand all of the connections that occur, and we can be relatively certain that there were no attacks during the normal training periods. Moreover, this environment is realistic. Many corporations have intranets in which activity is somewhat restricted and external connections must pass through a firewall. This environment could also model the increasingly common home network that connects to the Internet through a cable or DSL modem and has a single external IP address. Attacks are a reality in environments such as these, and the attack scenarios corresponded to plausible occurrences in this class of environment.

[2]The programs used to generate the results in this paper are available from http://www.cs.unm.edu/~immsec. The programs are part of LISYS and are found in the LisysSim directory of that package.

The normal network data in our data set consist of two weeks of data collected in November, 2001. In these data, there are a total of 22,329 TCP SYN packets, and roughly 55% of this is web traffic. Thus, there was an average of approximately 1600 packets per day during the normal period. Because the network data being produced is dependent on a small number of users, two weeks seemed to be the shortest period of time that could possibly give a reasonable characterization of self. Attack data were generated over the course of two days near the end of the collection period. The attacks took place about one week after the normal period ended, and consisted of 76,179 TCP SYN packets.

In [Hofmeyr, 1999], network connections to web servers are removed from the data by filtering out all connections to port 80. Instead of completely removing web connections, the data set simulates the behavior of a proxy server. All outgoing connections to port 80 (http) or port 443 (https) are re-mapped to port 3128 on the proxy machine. This is very close to what the traffic would have been like if we were using the web proxy cache SQUID.

All of the attacks, with the exception of the denial-of-service attack, were performed using a laptop connected to the internal network. The firewall machine was configured as a DHCP server, so the laptop was able to acquire a dynamic IP address because it had a physical connection to the internal network. We used the free security scanner Nessus to perform the attacks. A total of eight attacks were run, including denial of service (from an internal computer to an external computer), a firewall attack against the firewall/gateway machine, an ftp attack against an internal machine, an ssh probe against several internal machines, an attack probing for certain services, a TCP SYN scan, an nmap tcp connect() scan against several internal computers, and a full nmap port scan.

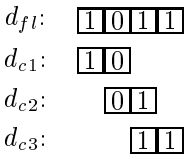
4 r-Chunks Matching

In this section we introduce a variant of the r-contiguous bits matching rule, which we refer to as "r-chunks." We will show in section 7 that r-chunks matching performs better than *full-length* r-contiguous bits matching for our data set. However, r-chunks matching also has the virtue of being more amenable to mathematical analysis than full-length matching [Esponda and Forrest, 2002]. r-Chunks matching is reminiscent of the $\{1, 0, \#\}$ matching rule for classifier systems [Holland *et al.*, 1986], with the additional restrictions that all detectors have a constant number of defined bits (the r parameter) and that all the de-

fined bits are located in contiguous positions. Matching with both r-chunks and full-length detectors is related to the crossover operator in genetic algorithms [Holland, 1975].

In r-chunks detectors, only r contiguous positions of the detector are specified (known as the *window* of the detector); the remaining bit positions can be thought of as "don't cares." Alternatively, an r-chunks detector can be thought of as a string of r bits together with a specification of the window to which it refers. An r-chunks detector d is said to match a string x if all the bits of d are equal to the r bits of x in the window specified by d.

The relation between full-length detectors and r-chunks is shown in the following figure for $l = 4$ and $r = 2$. A single full-length detector can be decomposed into $l - r + 1$ (the number of windows) r-chunks detectors. Let d_{fl} be the full-length detector and d_{c1}, d_{c2}, d_{c3} the r-chunks detectors into which it can be decomposed:

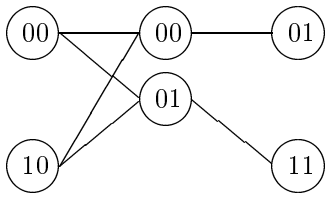


An important difference between the two match rules is in the number of undetectable strings they induce. We refer to these strings as "holes" and the set of holes for a given self set S as H. For full-length matching there are two sources of holes: crossover holes and length-limited holes.

Hence, a crossover hole is a string h not in S, for which all windows in h are crossovers of adjacent windows in S, according to the restricted crossover operation defined below. A crossover occurs in this context between two adjacent windows $W_i = v_i..v_{i+r-1}$ and $W_{i+1} = u_{i+1}..u_{i+r}$ whenever bits $v_j = u_j \; \forall_j : i + 1 \leq j \leq i + r - 1$. There is an example of this type of hole at the end of this section.

The second source of holes arises because in full-length detectors, all the bit positions are specified. This can induce holes h which are strings that have at least one window of r bits not present in S, but for which a detector still cannot be generated. For instance, let $S = \{110, 010\}, l = 3, r = 2$ and let $h = 011$ be a string that has r contiguous bits not exhibited in any string in S. A full-length detector for h must either start with the pattern 01 and/or end with the pattern 11 but any detector starting with 01 will match self and hence can not be generated. Similarly, if a potential detector for h ends with pattern 11 the two possible strings 011 and 111 match a string in S as well, therefore a detector for h cannot be generated.

r-Chunks detection does not induce length-limited holes, because a detector can always be generated for a pattern of length r which is not present in S. Thus, the only holes induced by r-chunks matching are crossover holes. This greatly simplifies the task of characterizing and managing holes. For example, the generalization of a set S, for r-chunks, can be depicted as a directed acyclic graph (DAG) with as many nodes as there are distinct bit patterns for each window (each node labelled as the bit pattern it represents) where two nodes are connected together if the windows they refer to crossover. Consider, for example, a self set S comprised of the following two strings $S = \{0001, 1011\}$ with $l = 4$, $r = 2$:



Following all the paths, starting from the leftmost nodes, yields the strings $\{0001, 0011, 1001, 1011\}$ which constitute the generalization of the r-chunks matching rule, out of which $\{0011, 1001\}$ are crossover holes. We refer to the holes plus the self strings that induced them as the *crossover-closure* [Helman, 2002].

5 The Experimental Setup

In LISYS, new detectors are generated when the system is initialized. Thereafter, new detectors are generated whenever another detector dies, usually through negative selection or old-age. Detectors are generated, trained, tested, and killed asynchronously throughout a LISYS run. Consequently, different detectors are tolerized at different times and are thus exposed to different samples of self.

Although this rolling coverage is desirable for dynamically changing self sets and to make evasion by an adversary more difficult, it also complicates analysis. Accordingly, for the experiments reported in this paper, we trained all detectors on the identical set of self strings (training set), and tested them subsequently against the identical set of test strings. We did not kill off detectors due to old age. In all of the experiments

the initial tolerization period was set to 15,000 packets, corresponding to approximately 8 days. Among these 15,000 initial packets, there were 131 unique strings to which the immature detectors were exposed.

6 The Effect of Permutation Masks

The goal of the first experiment was to assess how different permutations affect the performance of the system. Because performance is measured in terms of true and false positives, this experiment also tests the effect of permutations on the system's ability to generalize (because low false positive rates correspond to good generalization).

100 sets of detectors were tolerized using the 131 unique strings derived from the first 15,000 packets in the data-set (the *training set*), and each detector set was assigned a random permutation mask. Each detector set had exactly 5,000 mature detectors at the end of the tolerization period and an r-value of 10. These numbers were chosen on the basis of previous experiments [Balthrop *et al.*, 2002] which showed that 5,000 detectors provide maximal coverage (i.e. adding more detectors does not improve subsequent matching) for this data set and r threshold.[3] Each set of detectors was then run against the remaining 7,329 normal packets, as well as against the simulated attack data. In these data (the *test sets*), there are a total of 476 unique 49-bit strings. Of these 476, 50 also occur in the training set and are thus undetectable (because any detectors which would match them are eliminated during negative selection). This leaves 426 potentially detectable strings, of which 26 come from the normal test set and 400 are from the attack test set. The maximal possible coverage by a detector set is thus 426 unique matches.

An ideal detector set would achieve zero false positives on the normal test data and a high number of true positives on the attack data. Thus, a perfect detector set would match the 400 unique attack strings, and fail to match the 26 unique normal strings in the test set, thus generalizing from the self observed during training. Note that because network attacks rarely, if ever, produce only a single anomalous packet, we don't need to achieve perfect true-positive rates at the packet level in order to detect all attacks against the system.

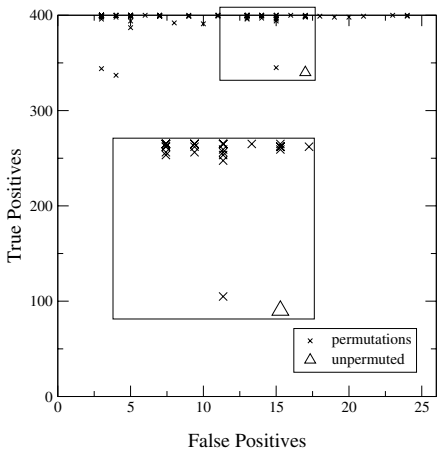


Figure 2: LISYS performance under different permutations. Each plotted point corresponds to a different permutation, showing false positives (x-axis) and true positives (y-axis). The inset shows a zoomed view of the same data.

Figure 2 shows the results of this experiment. The performance of each detector set is shown as a separate point on the graph. Each detector set has its own randomly generated permutation of the 49 bits, so each point shows the performance of a different permutation. The numbers on the x-axis correspond to the number of unique self-strings in the test set which are matched by the detector set, i.e. the number of false positives (up to a maximum of 26). The y-axis plots the corresponding value with respect to the attack data, i.e. the number of unique true positive matches (up to a maximum of 400). The graph shows that there is a large difference in the discrimination ability of different permutations. Points in the upper left of the graph are the most desirable, i.e. they correspond to permutations which minimize the number of false positives and maximize the number of true positives; points toward the lower right corner of the graph indicate higher false positives and/or lower true positives.

Surprisingly, the performance of the original (unpermuted) mapping is among the worst we found, suggesting that the results reported in [Balthrop *et al.*, 2002] are a worst case in terms of true vs. false positives. Almost any other random permutation we tried outperforms the original mapping. Although we don't yet have definitive proof, we believe this behavior arises in the following way.

[3] The use of 5,000 detectors to protect 131 unique strings is clearly a somewhat artificial situation. This arises from the small size of our data set and the decision to provide maximal coverage of non-self. In general, once the number of self strings increases above a certain threshold, the number of detectors needed to cover non-self through negative detection becomes less than that required for positive detection (see [Esponda and Forrest, 2002] for the exact tradeoff). And, for most applications, complete coverage of non-self is an overly strict requirement.

The LISYS design assumes that there are certain predictive bit-patterns that exhibit regularity in self, and that these can be the basis of distinguishing self from non-self. As it turns out, there are also deceptive bit-patterns which exhibit regularity in the training set (observed self), but the regularity does not generalize to the rest of self (the normal part of the test set). These patterns tend to cause false positives when self strings that do not fit the predicted regularity occur.

We believe that the identity permutation is bad because the predictive bits are at the ends of the string, while the deceptive region is in the middle. Under such an arrangement, it is difficult to find a window that covers many predictive bit positions without also including deceptive ones. It is highly likely that a random permutation will break up the deceptive region, and bring the predictive bits closer to the middle, where they will appear in more windows.

7 r-Chunks vs. Full-Length Detectors

In this section we compare the performance of r-chunks matching to that of r-contiguous bits matching with full-length detectors on our data set. The essential difference between full-length detectors and r-chunks lies in the holes which they induce, as discussed earlier. Holes are desirable to the extent that they prevent false positives (strings which are close to self and represent legitimate but novel behavior of the network)[4]; holes are undesirable to the extent which they lead to false negatives (a failure to match strings which correspond to attempted intrusions). Although both representations are subject to crossover holes, full-length detectors are additionally subject to length-limited holes. Therefore, we are interested in knowing if in practice length-limited holes generalize over true positives or false positives.

For this experiment, we generated one set of r-chunks detectors for each value of r, ranging from 1 to 12. Because there are only $2^r \times (l - r + 1)$ possible r-chunks detectors, we generated all of them, and then eliminated through negative selection any detector that matched a string in the training set. Full-length detectors were generated according the the procedure described in Section 5.

The results of this experiment are shown in Figure 3. As in the initial permutation-mask experiment, the number of false positives is plotted on the x-axis and the number of true positives on the y-axis. There are two sets of points, each connected by lines. One set indicates the results obtained with r-chunks for values of r ranging from 1 to 12. The second set, shows the results of using full-length detectors for $r = 8, 10,$ and 12.

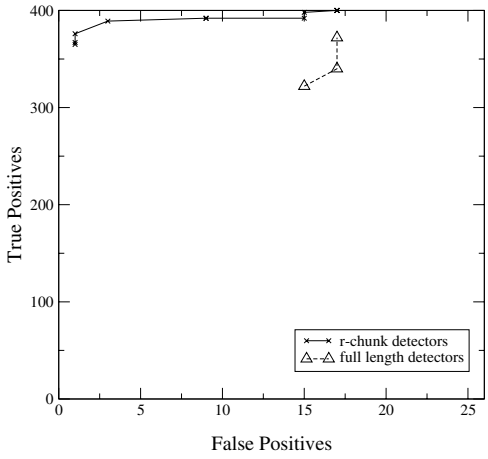


Figure 3: LISYS performance under different r-values. For r-chunks we plotted r =1..12 and for full length detectors we plotted r =8, 10 and 12 (the points for r-chunks and those for full-length detectors are each connected via a line to indicate the ordering in terms of r). Each point shows false positives (x-axis) and true positives (y-axis).

Section 4 tells us that for any self set, a given value of r will always achieve equivalent-or-greater overall coverage (i.e. a greater sum-total of true and false positives) when using r-chunks than with using full-length detectors. This follows from the fact that there are no holes induced by r-chunks which are not also induced using full-length detectors. The experiment shows whether or not this additional coverage is helpful. Figure 3 shows that for this data set r-chunks outperforms full-length detectors. The greater coverage achieved by r-chunks more often results in the detection of true positives than false positives. In fact, for any value of r shown using full-length detectors, there exists some value of r for which r-chunks achieve a higher rate of true positives while incurring an equal or lesser number of false positives.

Another property of r-chunks illustrated by the graph is that for a given value of r, equivalent-or-greater overall coverage will always be achieved using $r + 1$ rather than r. This is because any string detected using r can be detected using $r+1$. For this reason, as we increase r, while the number of true and/or false positives may increase or remain constant, neither value can decrease.

[4] This is the sense in which holes can be thought of as generalizations.

A surprising result is how well r-chunks performs as r becomes low (e.g. even for $r = 1$). An explanation for this phenomenon is discussed below. This is surprising in part because of the difficulty reported by Kim and Bentley [2001] in finding detectors using r-contiguous bits and negative selection, a result explained in part by their choice of a low value for r [Balthrop *et al.*, 2002].

7.1 r-Chunks and the Magic Bit

We were interested in how r-chunks could perform so well, especially for $r = 1$. A closer examination of the data revealed that the DHCP (Dynamic Host Configuration Protocol) configuration on the internal network was set up in such a way that dynamic IP addresses were always assigned with the final byte in the range 128-254, while static IP addresses were always in the range of 1-127 for the same byte. This is not an unusual DHCP configuration. As it happened, however, no hosts connected to the network using DHCP during the normal data collection period. When we ran the attacks, the attacking laptop did use DHCP to connect to the network, and the majority of the attacks were launched from this laptop (the Denial-of-Service attack is the only one that wasn't).

As a consequence, the majority of our attack data had the first bit of the 49-bit string (the internal IP is at the start of the string) set to one, while none of the normal data had this bit set. In other words, there was a single "magic bit" that identified approximately 84% of the attack SYN packets. r-chunks was able to detect this magic bit and take advantage of it. Thus, even the smallest possible window $r = 1$ could take advantage of the magic bit, and because $r + 1$ can detect everything that r can detect, all of the other r values can use the magic bit as well.

Although artifacts such as these are not unlikely occurrences in real data, we were curious to see what the results would be without the presence of a magic bit. Would r-chunks (and full-length detectors) still perform well? To answer this question we eliminated the magic bit from our data by systematically changing the internal address of the computer from which the attacks originated to look like the address of another internal computer. This scenario is also realistic, because the attacks could as easily have originated from an internal computer as from a malicious laptop, and such an internal attack might be more difficult to detect.

We repeated the r-chunks experiments with this modified data set. The results are shown in Figure 4. From this figure, we can see that r-chunks did not perform as well as before. In particular, the low r-values did not yield results as dramatically positive as before. Removing the magic bit also hurt the performance of r-contiguous bits for $r = 10$ and $r = 12$, although the effect was not as significant as for r-chunks. However, r-chunks without the "magic bit" still outperforms full-length detectors with the magic bit for all the r-values we tested ($r = 8, 10, 12$).

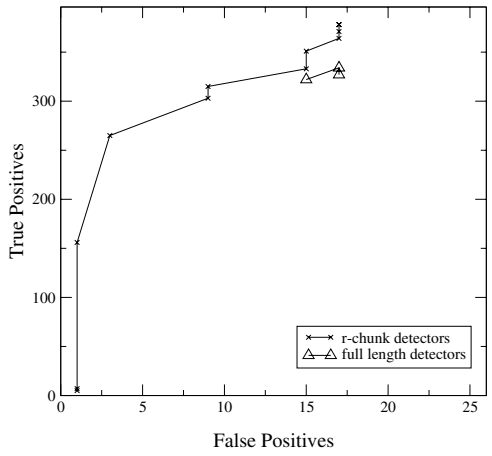


Figure 4: LISYS performance under different r-values after the magic bit has been removed. For r-chunks we plotted r =1..12 and for full length detectors we plotted r =8, 10 and 12. Each point shows false positives (x-axis) and true positives (y-axis).

8 Conclusions

In this paper we introduced a new matching rule, r-chunks, and showed that it performs better than full-length r-contiguous bits matching on one data set. r-Chunks is appealing because it is easier to analyze mathematically [Esponda and Forrest, 2002] and it scales well as the length of l increases (both in terms of efficiency of matching and in terms of number of detectors that are required for a given level of coverage). This second property is essential if AIS frameworks such as LISYS are to be used for real applications. We also studied the effect of different permutations on the ability of LISYS to generalize from an initial sample self. This form of generalization is important for controlling false positives. The results reported here show that some permutations perform much better than others, and we have given an informal explanation for why that is true.

The r-chunks detection scheme is intriguing because it solidifies the connection between r-contiguous bits matching and crossover. Although we have shown that

the crossover-closure is a good generalization for this data set, we still don't know whether it will carry over to related problems. However, the connection is tantalizing, and one that we plan to explore in future work.

It is important to emphasize that the results presented here are empirical and are based on one small data set. An important avenue for further work is to conduct experiments on other applications and to develop a mathematical understanding of the properties of this system. A second caveat concerns the simplified version of LISYS used to conduct these experiments. In the future, it will be important to confirm how well permutations and r-chunks perform in the context of the complete LISYS system.

Acknowledgments

The authors gratefully acknowledge the support of the National Science Foundation (grants IRI-9711199, CDA-9503064, and ANIR-9986555), the Office of Naval Research (grant N00014-99-1-0417), Defense Advanced Projects Agency (grant AGR F30602-00-2-0584), the Intel Corporation, and the Santa Fe Institute.

References

[Balthrop *et al.*, 2002] J. Balthrop, S. Forrest, and M. Glickman. Revisiting lisys: Parameters and normal behavior. In *CEC-2002: Proceedings of the Congress on Evolutionary Computing*, 2002.

[D'haeseleer *et al.*, 1996] P. D'haeseleer, S. Forrest, and P. Helman. An immunological approach to change detection: algorithms, analysis and implications. In *Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy*. IEEE Press, 1996.

[D'haeseleer, 1996] P. D'haeseleer. An immunological approach to change detection: theoretical results. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 1996.

[Esponda and Forrest, 2002] F. Esponda and S. Forrest. Defining self: Positive and negative detection. Technical Report TR-CS-2002-03, University of New Mexico, 2002.

[Heberlein *et al.*, 1990] L. T. Heberlein, G. V. Dias, K. N. Levitte, B. Mukherjee, J. Wood, and D. Wolber. A network security monitor. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEE Press, 1990.

[Helman, 2002] Paul Helman, 2002. Personal communication.

[Hofmeyr and Forrest, 1999] S. Hofmeyr and S. Forrest. Immunity by design: An artificial immune system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1289–1296, San Francisco, CA, 1999. Morgan-Kaufmann.

[Hofmeyr and Forrest, 2000] S. Hofmeyr and S. Forrest. Architecture for an artificial immune system. *Evolutionary Computation Journal*, 8(4):443–473, 2000.

[Hofmeyr, 1999] S. Hofmeyr. *An immunological model of distributed detection and its application to computer security*. PhD thesis, University of New Mexico, Albuquerque, NM, 1999.

[Holland *et al.*, 1986] J.H. Holland, K.J. Holyoak, R.E. Nisbett, and P. Thagard. *Induction: Processes of Inference, Learning, and Discovery*. MIT Press, 1986.

[Holland, 1975] John H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI, 1975.

[Kim and Bentley, 2001] J. Kim and P. J. Bentley. An evaluation of negative selection in an artificial immune system for network intrusion detection. In *GECCO-2001: Proceedings of the Genetic and Evolutionary Computation Conference*, 2001.

[Mukherjee *et al.*, 1994] B. Mukherjee, L. T. Heberlein, and K. N. Levitt. Network intrusion detection. *IEEE Network*, pages 26–41, 1994.

[Percus *et al.*, 1993] J. K. Percus, O. Percus, and A. S. Perelson. Predicting the size of the antibody combining region from consideration of efficient self/non-self discrimination. *Proceedings of the National Academy of Science*, 90:1691–1695, 1993.

[Wierzchon, 2000] S. T. Wierzchon. Discriminative power of the receptors activated by k-contiguous bits rule. *Journal of Computer Science and Technology*, 1(3):1–13, 2000.

[Wierzchon, 2001] S. T. Wierzchon. Deriving concise description of non-self patterns in an artificial immune system. In S. T. Wierzchon, L. C. Jain, and J. Kacprzyk, editors, *New Learning Paradigm in Soft Computing*, pages 438–458, Heidelberg New York, 2001. Physica-Verlag.

A Racing Algorithm for Configuring Metaheuristics

Mauro Birattari[†]
IRIDIA
Université Libre de Bruxelles
Brussels, Belgium

Thomas Stützle, Luis Paquete, and Klaus Varrentrapp
Intellektik/Informatik
Technische Universität Darmstadt
Darmstadt, Germany

Abstract

This paper describes a racing procedure for finding, in a limited amount of time, a configuration of a metaheuristic that performs as good as possible on a given instance class of a combinatorial optimization problem. Taking inspiration from methods proposed in the machine learning literature for model selection through cross-validation, we propose a procedure that empirically evaluates a set of candidate configurations by discarding bad ones as soon as statistically sufficient evidence is gathered against them. We empirically evaluate our procedure using as an example the configuration of an ant colony optimization algorithm applied to the traveling salesman problem. The experimental results show that our procedure is able to quickly reduce the number of candidates, and allows to focus on the most promising ones.

1 INTRODUCTION

A metaheuristic is a general algorithmic template whose components need to be instantiated and properly tuned in order to yield a fully functioning algorithm. The instantiation of such an algorithmic template requires to choose among a set of different possible components and to assign specific values to all free parameters. We will refer to such an instantiation as a *configuration*. Accordingly, we call *configuration problem* the problem of selecting the optimal configuration.

Practitioners typically configure their metaheuristics in an iterative process on the basis of some runs of different configurations that are felt as promising. Usually, such a process is heavily based on personal experience and is guided by a mixture of rules of thumb. Most often this leads to tedious and time consuming experiments. In addition, it is very rare that a configuration is selected on the basis of some well defined statistical procedure.

The aim of this work is to define an automatic hands-off procedure for finding a good configuration through statistically guided experimental evaluations, while minimizing the number of experiments. The solution we propose is inspired by a class of methods proposed for solving the model selection problem in memory-based supervised learning (Maron and Moore, 1994; Moore and Lee, 1994). Following the terminology introduced by Maron and Moore (1994), we call *racing method for selection* a method that finds a good configuration (model) from a given finite pool of alternatives through a sequence of steps.[1] As the computation proceeds, if sufficient evidence is gathered that some candidate is inferior to at least another one, such a candidate is dropped from the pool and the procedure is iterated over the remaining ones. The elimination of inferior candidates, speeds up the procedure and allows a more reliable evaluation of the promising ones.

Two are the main contributions of this paper. First, we give a formal definition of the metaheuristic configuration problem. Second, we show that a metaheuristic can be tuned efficiently and effectively by a racing procedure. Our results confirm the general validity of the racing algorithms and extend their area of applicability. On a more technical level, left aside the specific application to metaheuristics, we give some contribution to the general class of racing algorithms. In particular, our method adopts blocking design (Dean and Voss, 1999) in a nonparametric setting. In some sense, therefore, the method fills the gap between Hoeffding race (Maron and Moore, 1994) and BRACE (Moore and Lee, 1994): similarly to Hoeffding race it features a nonparametric test, and similarly to BRACE it considers a

[†]This research was carried out while MB was with Intellektik, Technische Universität Darmstadt.

[1]Several metaheuristics involve continuous parameters. This would actually lead to an infinite set of candidate configurations. In practice, typically only a finite set of possible parameter values are considered by discretizing the range of continuous parameters.

blocking design.

The rest of the paper is structured as follows. Section 2 gives a formal definition of the problem of configuring a metaheuristic. Section 3 describes the general ideas behind racing algorithms and introduces F-Race, a racing method specifically designed for matching the peculiar characteristics of the metaheuristic configuration problem. Section 4 proposes some background information on $\mathcal{MAX}$–$\mathcal{MIN}$-Ant-System and on the traveling salesman problem (TSP), which are respectively the metaheuristic and the problem considered in this paper. In particular, the section gives a description of the sub class of TSP instances, and of the candidate configurations of $\mathcal{MAX}$–$\mathcal{MIN}$-Ant-System that we consider in our experimental evaluation. Section 5 proposes some experimental results, and Section 6 concludes the paper.

2 CONFIGURING A METAHEURISTIC

This section introduces and defines the general problem of configuring a metaheuristic. Before proposing a formal definition, it is worth outlining briefly, with the help of an example, the type of problem setting to which our procedure applies. Namely, our methodology is meant to be applied to repetitive problems, that is, problems where many similar instances appear over time.

2.1 An Example: Delivering Pizza

The example we propose is admittedly simplistic and does not cover all possible aspects of the configuration problem; still it has the merit of highlighting those elements that are essential for the discussion that follows.

Let us consider the following **pizza delivery problem**. Orders are collected for a (fixed) time period of, say, 30 minutes. At the end of the time period, a pizza delivery boy has some limited amount of time for scheduling a reasonably short tour that visits all the customers that have called in the last 30 minutes. Then the boy leaves and delivers the pizzas following a chosen route. The time available for scheduling may be constant or may be expressed as a function of some characteristic of the instance itself, for example the size which in the pizza delivery problem might be measured by the number of customers to visit.

In such a setting, every 30 minutes a new instance of an optimization problem is given, and a solution as good as possible has to be found in a limited amount of time. It is very likely that every instance will be different from all previous ones in the location of the customers that need to be visited. Further, a certain variability in the instance size, that is the number of customers to be served, is to be expected, too.

The occurrence of different instances can be conveniently represented as the result of random experiments governed by some unknown probability measure, say P_I, defined on the class of the possible instances. In the example discussed here, it is reasonable to assume that different experiments are independent and all governed by the same probability measure. In Section 2.3, we will briefly discuss how to possibly tackle situations in which such assumptions appear unreasonable.

Now, our pizza delivery boy loves metaheuristics and uses one to find a shortest possible tour visiting all the customers. Being such a metaheuristic a general algorithmic template, different configurations are possible (see Section 4.2 for a more detailed example). In our setting, the problem that the delivery boy has to solve is to find the configuration that is expected to yield the best solution to the instances that he *typically* faces. The concept of *typical instance*, used here informally, has to be understood in relation to the probability measure P_I, and will receive a clear mathematical meaning presently.

Since P_I is unknown, the only information that can be used for finding the best configuration must be extracted from a sample of previously seen instances. By adopting the terminology used in machine learning, we will use the expression *training instances* to denote the available previous instances. On the basis of such training instances, we will look for the configuration that is expected to have the best performance over the *whole* class of possible instances.

The fact of extending results obtained on a usually small training set to a possibly infinite set of instances is a genuine *generalization*, as intended in supervised learning (Mitchell, 1997). In the context of metaheuristics configuration, generalization is fully justified by the assumption that the same probability measure P_I governs the selection of all the instances: both those used for training and those that will be solved afterwards. The training instances are in this sense representative of the whole set of instances.

2.2 The Formal Statement

In order to give a formal definition of the general problem of configuring a metaheuristic, we consider the following objects:

- Θ is the finite set of candidate configurations.
- I is the possibly infinite set of instances.
- P_I is a probability measure over the set I of instances: With some abuse of notation, we indicate with $P_I(i)$ the probability that the instance i is selected for being solved.[2]

[2]Since a probability measure is associated to (sub)sets and not

- $t : I \to \Re$ is a function associating to every instance the computation time that is allocated to it.

- $\boldsymbol{c}(\theta, i) = \boldsymbol{c}(\theta, i, t(i))$ is a random variable representing the cost of the best solution found by running configuration θ on instance i for $t(i)$ seconds.[3]

- $C \subset \Re$ is the range of $\boldsymbol{c}$, that is, the possible values for the cost of the best solution found in a run of a configuration $\theta \in Theta$ on an instance $i \in I$.

- P_C is a probability measure over the set C: With the notation[4] $P_C(c|\theta, i)$, we indicate the probability that c is the cost of the best solution found by running for $t(i)$ seconds configuration θ on instance i.

- $\mathcal{C}(\theta) = \mathcal{C}(\theta|\Theta, I, P_I, P_C, t)$ is the criterion that needs to be optimized with respect to θ. In the most general case it measures in some sense the desirability of θ.

On the basis of these concepts, the problem of configuring a metaheuristic can be formally described by the 6-tuple $\langle \Theta, I, P_I, P_C, t, \mathcal{C} \rangle$. The solution of this problem is the configuration θ^* such that:

$$\theta^* = \arg\min_{\theta} \mathcal{C}(\theta). \tag{1}$$

As far as the criterion $\mathcal{C}$ is concerned, different alternatives are possible. In this paper, we consider the optimization of the expected value of the cost $\boldsymbol{c}(\theta, i)$. Such a criterion is adopted in many different applications and, besides being quite natural, it is often very convenient from both the theoretical and the practical point of view. Formally:

$$\mathcal{C}(\theta) = E_{I,C}\Big[\boldsymbol{c}(\theta, \boldsymbol{i})\Big] = \int_I \int_C c(\theta, i)\, \mathrm{d}P_C(c|\theta, i)\, \mathrm{d}P_I(i), \tag{2}$$

where the expectation is considered with respect to both P_I and P_C, and the integration is taken in the Lebesgue sense (Billingsley, 1986).

The measures P_I and P_C are usually not explicitly available and the analytical solution of the integrals in Equation 2, one for each configuration θ, is not possible. In order to overcome such a limitation, the integrals defined in Equation 2 will be estimated in a Monte Carlo fashion on the basis of a training set of instances, as it will be explained in Section 3.

to single elements, the correct notation should be $P_I(\{i\})$. Our notational abuse consists therefore in using the same symbol i both for the element $i \in I$, and for the singleton $\{i\} \subset I$.

[3] In the following, for the sake of a lighter notation, the dependency of $\boldsymbol{c}$ on t will be often implicit.

[4] The same remark as in Note 2 applies here.

2.3 Further Considerations and Possible Extensions

The formal configuration problem, as described in Section 2.2, assumes that, as far as a given instance is concerned, no information on the performance of the various candidate configurations can be obtained prior to their actual execution on the instance itself. In this sense, the instances are *a priori* indistinguishable.

In many practical situations, it is known *a priori* that various types of instances with different characteristics may arise. In such a situation all possible prior knowledge should be used to cluster the instances into homogeneous classes and to find, for each class, the most suitable configuration.

The case mentioned in Section 2.1, in which it is not reasonable to accept that all instances are extracted independently and according to the same probability measure, can possibly be handled in a similar way. Often, some temporal correlation is observed among instances. In other words, temporal patterns can be observed on previous instances that bring *a priori* information on the characteristics of the current instance. This phenomenon can be handled by assuming that the instances are generated by a process akin to a time-series. Also in this case, different configuration problems should be formulated: Each class of instances to be treated separately would be composed by instances that follow in time a given pattern and that are therefore supposed to share similar characteristics. The aim is again to match the hypothesis of *a priori* indistinguishability of instances within each of the different configuration problems in which the original one is reformulated.

3 A RACING ALGORITHM

Before giving a definition of a racing algorithm for solving the problem given in Equation 1, it is convenient to describe a somewhat naive *brute-force* approach for highlighting some of the difficulties associated with the configuration problem.

A brute-force approach to the problem defined in Equation 1 consists in estimating the quantities defined in Equation 2 by means of a *sufficiently large* number of runs of each candidate on a *sufficiently large* set of training instances. The candidate configuration with the smallest estimated quantity is then selected.

However, such a *brute-force* approach presents some drawbacks: First, the size of the training set must be defined prior to any computation. A criterion is missing to avoid considering, on the one hand, too few instances, which could prevent from obtaining reliable estimates, and on the other hand, too many instances, which would then require a great deal of useless computation. Second, no criterion

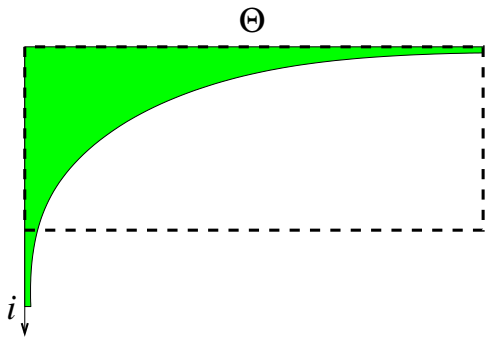


Figure 1: A visual representation of the amount of computation needed by the two methods. The surface of the dashed rectangle represents the amount of computation for brute-force, the shadowed area the one for racing.

is given for deciding how many runs of each configuration on each instance should be performed in order to cope with the stochastic nature of metaheuristics. Finally, the same computational resources are allocated to each configuration: manifestly poor configurations are thoroughly tested to the same extent as the best ones are.

3.1 Racing Algorithms: The Idea

Racing algorithms are designed to provide a better allocation of computational resources among candidate configurations and therefore to overcome the last of the three above described drawbacks of brute-force. At the same time, the racing framework indirectly allows for a clean solution to the first two problems of brute-force, that is the problems of fixing the number of instances and the number of runs to be considered.

To do so, racing algorithms sequentially evaluate candidate configurations and discard poor ones as soon as statistically sufficient evidence is gathered against them. The elimination of inferior candidates speeds up the procedure and allows to evaluate the promising configurations on more instances and to obtain more reliable estimates of their behavior. Figure 1 visualizes the two different ways of allocating computational resources to candidate configurations that are adopted by brute-force and by racing algorithms.

Let us suppose that a random sequence of training instances $\underline{\boldsymbol{i}}$ is available, where the generic k-th term $\underline{\boldsymbol{i}}_k$ is drawn from I according to P_I, independently for each k. We assume that $\underline{\boldsymbol{i}}$ can be extended at will and at a negligible cost, by sampling further from I.

With the notation $\underline{\boldsymbol{c}}^k(\theta, \underline{\boldsymbol{i}})$ we indicate an array of k terms whose generic l-th one is the cost $\boldsymbol{c}(\theta, \underline{\boldsymbol{i}}_l)$ of the best solution found by configuration θ on instance $\underline{\boldsymbol{i}}_l$ in a run of $t(\underline{\boldsymbol{i}}_l)$ seconds. It is clear therefore that, for a given θ, the array $\underline{\boldsymbol{c}}^k$ of length k can be obtained from $\underline{\boldsymbol{c}}^{k-1}$ by appending to the latter the cost concerning the k-th instance in $\underline{\boldsymbol{i}}$.

A racing algorithm tackles the optimization problem in Equation 1 by generating a sequence of nested sets of candidate configurations:

$$\Theta_0 \supseteq \Theta_1 \supseteq \Theta_2 \supseteq \ldots,$$

starting from $\Theta_0 = \Theta$. The step from a set Θ_{k-1} to Θ_k is obtained by possibly discarding some configurations that appear to be suboptimal on the basis of information available at step k.

At step k, when the set of candidates still in the race is Θ_{k-1}, a new instance $\underline{\boldsymbol{i}}_k$ is considered. Each candidate $\theta \in \Theta_{k-1}$ is executed on $\underline{\boldsymbol{i}}_k$ and each observed cost $\boldsymbol{c}(\theta, \underline{\boldsymbol{i}}_k)$ is appended to the respective $\underline{\boldsymbol{c}}^{k-1}$ to form the different arrays $\underline{\boldsymbol{c}}^k(\theta, \underline{\boldsymbol{i}})$, one for each θ. Step k terminates defining set Θ_k by dropping from Θ_{k-1} the configurations that appear to be suboptimal in the light of some statistical test that compares the arrays $\underline{\boldsymbol{c}}^k(\theta, \underline{\boldsymbol{i}})$ for all $\theta \in \Theta$. The description of the test considered in this paper is given in Section 3.2. It should be noticed here that, for any θ, each component of the array $\underline{\boldsymbol{c}}^k(\theta, \underline{\boldsymbol{i}})$, that is, any cost $\boldsymbol{c}(\theta, \boldsymbol{i})$ of the best solution found by a single run of θ over one generic $\boldsymbol{i}$ extracted according to P_I, is an estimate of $\mathcal{C}(\theta)$, as defined in Equation 2. The sampling average of $\underline{\boldsymbol{c}}^k(\theta, \underline{\boldsymbol{i}})$ is therefore itself an estimate of $\mathcal{C}(\theta)$ and can be used for comparing the performance yielded by different configurations.

The above described procedure is iterated and stops either when all configurations but one are discarded, or when some predefined total time T of computation is reached. That is, the procedure would stop before considering the $(k+1)$-th instance if $\sum_{l=1}^{k} t(\underline{\boldsymbol{i}}_{l+1}) \left|\Theta_l\right| > T$.

3.2 F-Race

The racing algorithm we propose, F-Race in the following, is based on the Friedman test, a statistical method for hypothesis testing also known as Friedman two-way analysis of variance by ranks (Conover, 1999).

For giving a description of the test, let us assume that F-Race has reached step k, and $n = |\Theta_{k-1}|$ configurations are still in the race. The Friedman test assumes that the observed costs are k mutually independent n-variate random variables $(\underline{\boldsymbol{c}}^k(\theta_1, \underline{\boldsymbol{i}}_l), \underline{\boldsymbol{c}}^k(\theta_2, \underline{\boldsymbol{i}}_l), \ldots, \underline{\boldsymbol{c}}^k(\theta_n, \underline{\boldsymbol{i}}_l))$ called *blocks* (Dean and Voss, 1999) where each block corresponds to the computational results on instance $\underline{\boldsymbol{i}}_l$ for each configuration in the race at step k. Within each block the quantities $\underline{\boldsymbol{c}}^k(\theta, \underline{\boldsymbol{i}}_l)$ are ranked from the smallest to the largest. Average ranks are used in case of ties. For each configuration $\theta_j \in \Theta_{k-1}$, let R_{lj} be the rank of θ_j within block l, and $R_j = \sum_{l=1}^{k} R_{lj}$ the sum of the ranks over all

instances $\underline{\boldsymbol{i}}_l$, with $1 \leq l \leq k$. The Friedman test considers the following statistic (Conover, 1999):

$$T = \frac{(n-1)\sum_{j=1}^{n}\left(R_j - \frac{k(n+1)}{2}\right)^2}{\sum_{l=1}^{k}\sum_{j=1}^{n} R_{lj}^2 - \frac{kn(n+1)^2}{4}}.$$

Under the null hypothesis that all possible rankings of the candidates within each block are equally likely, T is approximatively χ^2 distributed with $n-1$ degrees of freedom. If the observed T exceeds the $1-\alpha$ quantile of such a distribution, the null is rejected, at the approximate level α, in favor of the hypothesis that at least one candidate tends to yield a better performance than at least one other.

If the null is rejected, we are justified in performing pairwise comparisons between individual candidates. Candidates θ_j and θ_h are considered different if

$$\frac{|R_j - R_h|}{\sqrt{\frac{2k\left(1-\frac{T}{k(n-1)}\right)\left(\sum_{l=1}^{k}\sum_{j=1}^{n} R_{lj}^2 - \frac{kn(n+1)^2}{4}\right)}{(k-1)(n-1)}}} > t_{1-\alpha/2},$$

where $t_{1-\alpha/2}$ is the $1-\alpha/2$ quantile of the Student's t distribution (Conover, 1999).

In F-Race, if at step k the null of the aggregate comparison is not rejected, all candidates in Θ_{k-1} pass to Θ_k. On the other hand, if the null is rejected, pairwise comparisons are executed between the best candidate and each other one. All candidates that result significatively worse than the best are discarded and will not appear in Θ_k.

3.3 Discussion on the Role of Ranking in F-Race

In F-Race, ranking plays an important two-fold role. The first one is connected with the nonparametric nature of a test based on ranking. The main merit of nonparametric analysis is that it does not require to formulate hypotheses on the distribution of the observations. Discussions on the relative pros and cons of the parametric and nonparametric approaches can be found in most textbooks on statistics (Larson, 1982). For an organic presentation of the topic, we refer the reader, for example, to Conover (1999). Here we limit ourselves to mention some widely accepted facts about parametric and nonparametric hypothesis testing: When the hypotheses they formulate are met, parametric tests have a higher power than nonparametric ones and usually require much less computation. Further, when a large amount of data is available the hypotheses for the application of parametric tests tend to be met in virtue of the central limit theorem. Finally, it is well known that the t-test, the classical parametric test that is of interest here, is robust against departure from some of its hypotheses, namely the normality of data: When the hypothesis of normality is not strictly met t-test *gracefully* looses power.

For what concerns the metaheuristics configuration problem, we are in a situation in which these arguments look suspicious. First, since we wish to reduce as soon as possible the number of candidates, we deal with very small samples and it is exactly on these small samples, for which the central limit theorem cannot be advocated, that we wish to have the maximum power. Second, the computational costs are not really relevant since in any case they are negligible compared to the computational cost of executing configurations of the metaheuristic in order to enlarge the available samples. Section 5 shows that the doubts expressed here find some evidential support in our experiments.

A second role played by ranking in F-Race is to implement in a natural way a blocking design (Dean and Voss, 1999). The variation in the observed costs $\boldsymbol{c}$ is due to different sources: Metaheuristics are intrinsically stochastic algorithms, the instances might be very different one from the other, and finally some configurations perform better than others. This last source of variation is the one that is of interest in the configuration problem while the others might be considered as disturbing elements. Blocking is an effective way for normalizing the costs observed on different instances. By focusing only on the ranking of the different configurations within each instance, blocking eliminates the risks that the variation due to the difference among instances washes out the variation due to the difference among configurations.

The work proposed in this paper was openly and largely inspired by some algorithms proposed in the machine learning community (Maron and Moore, 1994; Moore and Lee, 1994) but it is precisely in the adoption of a statistical test based on ranking that it diverges from previously published works. Maron and Moore (1994) proposed Hoeffding Race that adopts a nonparametric approach but does not consider blocking. In a following paper, Moore and Lee (1994) describe BRACE that adopts blocking but discards the nonparametric setting in favor of a Bayesian approach. Other relevant work was proposed by Gratch et al. (1993) and by Chien et al. (1995) who consider blocking in a parametric setting.

This paper, to the best of our knowledge, is the first work in which blocking is considered in a nonparametric setting. Further, in all the above mentioned works blocking was always implemented through multiple pairwise paired comparisons (Hsu, 1996), and only in the more recent one (Chien et al., 1995) correction for multiple tests is considered. F-Race is the first racing algorithm to implement blocking through ranking and to adopt an aggregate test over all candidates, to be performed prior to any pairwise test.

4 $\mathcal{MAX}$–$\mathcal{MIN}$-ANT-SYSTEM FOR TSP

In this paper we illustrate F-Race by using as an example the configuration of $\mathcal{MAX}$–$\mathcal{MIN}$-Ant-System ($\mathcal{MM}$AS) (Stützle and Hoos, 1997; Stützle and Hoos, 2000), a particular Ant Colony Optimization algorithm (Dorigo and Di Caro, 1999; Dorigo and Stützle, 2002), over a class of instances of the Traveling Salesman Problem (TSP).

4.1 A Class of TSP Instances

Given a complete graph $G = (N, A, d)$ with N being the set of $n = |N|$ nodes, A being the set of arcs fully connecting the nodes, and d being the weight function that assigns each arc $(i, j) \in A$ a length d_{ij}, the Traveling Salesman Problem (TSP) is the problem of finding a shortest closed tour visiting each node of G once. We assume the TSP is symmetric, that is, we have $d_{ij} = d_{ji}$ for every pair of nodes i and j.

The TSP is extensively studied in literature and that serves as a standard benchmark problem (Johnson and McGeoch, 1997; Lawler et al., 1985; Reinelt, 1994). For our study we randomly generate Euclidean TSP instances with a random distribution of city coordinates and a random number of cities. Euclidean TSPs were chosen because such instances are used in a large number of experimental researches on the TSP (Johnson and McGeoch, 1997; Johnson et al., 2001). In our case, city locations are randomly chosen according to a uniform distribution in a square of dimension 10.000×10.000, and the resulting distances are rounded to the nearest integer. The number of cities in each instance is chosen as an integer randomly sampled according to a uniform distribution in the interval $[300, 500]$. We generated a total number of 400 such instances for our experiments reported in Section 5.

4.2 $\mathcal{MAX}$–$\mathcal{MIN}$-Ant-System

Ant Colony Optimization (ACO) (Dorigo et al., 1999; Dorigo and Di Caro, 1999; Dorigo and Stützle, 2002) is a population-based approach inspired by the foraging behavior of ants for the solution of hard combinatorial optimization problems. In ACO, artificial ants implement stochastic construction procedures that are biased by pheromone trails and heuristic information on the problem being solved. The solutions obtained by the ants may then be improved by applying some local search routine. ACO algorithms typically follow the high-level procedure given in Figure 2. $\mathcal{MM}$AS (Stützle and Hoos, 1996, 1997; Stützle and Hoos, 2000) is currently one of the best performing ACO algorithms for the TSP.

$\mathcal{MAX}$–$\mathcal{MIN}$-Ant-System constructs tours as follows: Initially, each of the m ants is put on some randomly chosen

procedure *Ant Colony Optimization*
  Init pheromones, calculate heuristic
  while(termination condition not met) **do**
    $p =$ ConstructSolutions(*pheromones, heuristic*)
    $p =$ LocalSearch(p) *% optional*
    GlobalUpdateTrails(p)
  end
end *Ant Colony Optimization*

Figure 2: Algorithmic skeleton of ACO for static combinatorial optimization problems.

city. At each construction step, ant k applies a probabilistic action choice rule. In particular, when being at city i, ant k chooses to go to a yet unvisited city j at the tth iteration with a probability of

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta}, \qquad \text{if } j \in \mathcal{N}_i^k; \quad (3)$$

where $\eta_{ij} = 1/d_{ij}$ is an *a priori* available heuristic value, α and β are two parameters which determine the relative influence of the pheromone trail and the heuristic information, and $\mathcal{N}_i^k$ is the feasible neighborhood of ant k, that is, the set of cities which ant k has not yet visited; if $j \notin \mathcal{N}_i^k$, we have $p_{ij}^k(t) = 0$.

After all ants have constructed a solution, the pheromone trails are updated according to

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij}^{best}, \qquad (4)$$

where $\Delta\tau_{ij}^{best} = 1/L^{best}$ if arc $(i, j) \in T^{best}$ and zero otherwise. Here T^{best} is either the *iteration-best* solution T^{ib}, or the *global-best* solution T^{gb} and L^{best} is the corresponding tour length. Experimental results showed that the best performance is obtained by gradually increasing the frequency of choosing T^{gb} for the pheromone trail update (Stützle and Hoos, 2000).

In $\mathcal{MM}$AS, lower and upper limits τ_{min} and τ_{max} on the possible pheromone strengths on any arc are imposed to avoid search stagnation. The pheromone trails in $\mathcal{MM}$AS are initialized to their upper pheromone trail limits τ_{max}, leading to an increased exploration of tours at the start of the algorithms.

In our experimental study, we have chosen a number of configurations that differ in particular parameter settings for $\mathcal{MM}$AS. We focused on alternative settings for the main algorithm parameters as they were identified in earlier studies, in particular we considered values of $\alpha \in \{1, 1.25, 1.5, 2\}$, $m \in \{1, 5, 10, 25\}$, $\beta \in \{0, 1, 3, 5\}$, $\rho \in \{0.6, 0.7, 0.8, 0.9\}$. Each possible combination of the parameter settings leads to one particular algorithm config-

uration, leading to a total number of $4 \times 4 \times 4 \times 4 = 256$ configurations. In our experiments each solution is improved by a 2.5-opt local search procedure (Bentley, 1992).

5 EXPERIMENTAL RESULTS

In this section we propose a Monte Carlo evaluation of F-Race based on a resampling technique (Good, 2001).

For comparison, we consider two other instances of racing algorithms both based on a paired t-test. They are therefore parametric, and they adopt a blocking design. We refer to them as *tn-Race* and *tb-Race*. The first does not adopt any correction for multiple-tests, while the second adopts the Bonferroni correction and is therefore *not unlike* the method described by Chien et al. (1995).

The goal is to select an *as good as possible* configuration out of the 256 configurations of the $\mathcal{MAX}$–$\mathcal{MIN}$-Ant-System described in Section 4.2.

Each configuration was executed once on each of the 400 instances for $10s$ on a CPU Athlon 1.4GHz with 512 MB of RAM, for a total time of about 12 days to allow in a following phase the application of the resampling analysis. The costs of the best solution found in each of these experiments were stored in a two-dimensional 400×256 array. In the following, when saying that we *run* configuration j over instance i, we will simply mean that we execute the *pseudo-experiment* that consists in reading the value in position (i, j) from the array of the results.

From the 400 instances, we extract 1000 pseudo-samples each of which is obtained by re-ordering randomly the original instances. Each pseudo-sample is used for a *pseudo-trial*, that is, for simulating a run of a racing algorithm: One after the other the instances are considered and, on the basis of the results of pseudo-experiments, configurations are progressively discarded. Each algorithm stops after executing 5×256 pseudo-experiments.[5] Upon time expiration, the best candidate in the pseudo-trial is selected and it is tested on 10 instances that were ***not*** used during the selection itself. The results obtained on these previously unseen instances are recorded and are used for comparing the three racing methods. To summarize, after 1000 pseudo-trials a vector of 10×1000 components is obtained for each of F-Race, tn-Race, and tb-Race. It is important to note that the three algorithms face the same pseudo-samples and that the candidates selected in each pseudo-trial by each algorithm are tested on the same unseen instances. The generic i-th components of the three 10×1000 vectors refers therefore to the results obtained by the champions of the three races respectively, where the three races were conducted on the basis of the same pseudo-sample: We are therefore justified in using paired statistical tests when comparing the three races among them.

On the basis of a paired Wilcoxon test we can state that F-Race is significatively better, at a significance level of 5%, than both tn-Race and tb-Race.[6]

Some insight on this result can be obtained from the following observation. By early dropping the less interesting candidates, F-Race is able to perform more experiments on the more promising candidates. On the 1000 pseudo-trials considered, at the moment in which the computation time was up and a decision among the surviving candidate had to be taken, the set of survivors was on average composed by 7.9 candidates and such survivors had been tested on average on 77.9 instances. In the case of tn-Race, the average size of the set of survivors upon expiration of computation time was 31.1, while the number of instances seen by such survivors was on average 18.2. For tb-Race the numbers are 253.8 and 5, respectively. In this sense, F-Race proved to be the bravest of the three, while tb-Race appeared to be extremely conservative and on average it dropped only slightly more than 2 candidates before the time limit.

On the basis of our Monte Carlo evaluation, some stronger statement can be pronounced on the quality of the results obtained by F-Race. We have shown above that the performance of F-Race was good in a *relative* sense: F-Race produced better results than its competitors. We state now that, in a precise sense to be defined presently, the performance of F-Race was *absolutely* good. We compare F-Race with *Cheat*, a brute-force method that, rather unfairly, uses in each pseudo-trial the same number of instances used by F-Race and on these instances runs all the candidate configurations. In doing so, Cheat allows itself an enormously large amount of computation time. In our experiments, Cheat has performed on average about 19950 experiments per trial which is equivalent to about 55 hours of computation against the 3.5 hour available to F-Race. The selection operated by Cheat is the *optimum* that can be obtained from the fixed set of training instances, and considering only one run of each configuration on each instance. F-Race can be seen as an approximation of Cheat: The set of experiment performed by F-Race is a *proper subset* of the experiments performed by Cheat.

Now, in the statistical analysis of the results obtained by our Monte Carlo experiments, we were not able to reject the null that F-Race and Cheat produce equivalent results. Also in this case, we have worked at the significance level of 5%: neither Wilcoxon test nor t-test were able to show significance.

[5] In such a time, by definition, brute-force would be able to test the 256 candidates on only 5 instances. The 5×256 pseudo-experiments simulate 3.5 hours of actual computation on the computer used for producing the results proposed here.

[6] The same conclusion can be drawn on the basis of a paired t-test.

6 CONCLUSIONS

The paper has given a formal definition of the problem of configuring a metaheuristic and has presented F-Race, an algorithm belonging to the class of racing algorithms proposed in the machine learning community for solving the model selection problem (Maron and Moore, 1994).

In giving a formal definition of the configuration problem, we have stressed the important role played by the probability measure defined on the class of the instances. Without such a concept, it is impossible to give a meaning to the generalization process that is implicit when a configuration is selected on the basis of its performance on a limited set of instances.

F-Race, the algorithm we propose in this paper, is the specialization of the generic class of racing algorithms to the configuration of metaheuristics. The adoption of the Friedman test, which is nonparametric and two-way, matches indeed the specificities of the configuration problem. As shown by the experimental results proposed in Section 5, F-Race obtains better results than its competitors that adopt a parametric approach. This better performance can be indeed explained by the ability of discarding inferior candidates earlier and faster than the competitors. Still, we do not feel like using these results for claiming a general *presumed* superiority of F-Race against its fellow racing algorithms. Rather, we wish to stress the appeal of the racing idea in itself, and we want to interpret our results as an evidence that this idea is extremely promising for configuring metaheuristics and should be further investigated.

Acknowledgments

This work was supported by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential programme of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

Bentley, J. L. (1992). Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, 4(4):387–411.

Billingsley, P. (1986). *Probability and Measure*. John Wiley & Sons, New York, NY, USA, second edition.

Chien, S., Gratch, J., and Burl, M. (1995). On the efficient allocation of resources for hypothesis evaluation: A statistical approach. *Pattern Analysis and Machine Intelligence*, 17(7):652–665.

Conover, W. J. (1999). *Practical Nonparametric Statistics*. John Wiley & Sons, New York, NY, USA, third edition.

Dean, A. and Voss, D. (1999). *Design and Analysis of Experiments*. Springer Verlag, New York, NY, USA.

Dorigo, M. and Di Caro, G. (1999). The Ant Colony Optimization meta-heuristic. In Corne, D., Dorigo, M., and Glover, F., editors, *New Ideas in Optimization*, pages 11–32. McGraw Hill, London, UK.

Dorigo, M., Di Caro, G., and Gambardella, L. M. (1999). Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172.

Dorigo, M. and Stützle, T. (2002). The ant colony optimization metaheuristic: Algorithms, applications and advances. In *Metaheuristics Handbook*. Kluwer Academic Publishers. In press.

Good, P. I. (2001). *Resampling Methods*. Birkhauser, Boston, MA, USA, second edition.

Gratch, J., Chien, S., and DeJong, G. (1993). Learning search control knowledge for deep space network scheduling. In *International Conference on Machine Learning*, pages 135–142.

Hsu, J. (1996). *Multiple Comparisons*. Chapman & Hall/CRC, Boca Raton, Fl, USA.

Johnson, D. S. and McGeoch, L. A. (1997). The travelling salesman problem: A case study in local optimization. In Aarts, E. H. L. and Lenstra, J. K., editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, Chichester, UK.

Johnson, D. S., McGeoch, L. A., Rego, C., and Glover, F. (2001). 8th DIMACS implementation challenge. `http://www.research.att.com/~dsj/chtsp/`.

Larson, H. (1982). *Introduction to Probability Theory and Statistical Inference*. John Wiley & Sons, New York, NY, USA.

Lawler, E. L., Lenstra, J. K., Kan, A. H. G. R., and Shmoys, D. B. (1985). *The Travelling Salesman Problem*. John Wiley & Sons, Chichester, UK.

Maron, O. and Moore, A. W. (1994). Hoeffding races: Accelerating model selection search for classification and function approximation. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems*, volume 6, pages 59–66. Morgan Kaufmann Publishers, Inc.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, New York, NY, USA.

Moore, A. W. and Lee, M. S. (1994). Efficient algorithms for minimizing cross validation error. In *International Conference on Machine Learning*, pages 190–198. Morgan Kaufmann Publishers, Inc.

Reinelt, G. (1994). *The Traveling Salesman: Computational Solutions for TSP Applications*, volume 840 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Germany.

Stützle, T. and Hoos, H. (1996). Improving the Ant-System: A detailed report on the $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. Technical Report AIDA–96–12, FG Intellektik, TU Darmstadt, Germany.

Stützle, T. and Hoos, H. H. (1997). The $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System and local search for the traveling salesman problem. In Bäck, T., Michalewicz, Z., and Yao, X., editors, *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, pages 309–314. IEEE Press, Piscataway, NJ, USA.

Stützle, T. and Hoos, H. H. (2000). $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. *Future Generation Computer Systems*, 16(8):889–914.

Dynamic Search with Charged Swarms

T.M.Blackwell

Department of Computer Science
University College London
Gower Street,
London, UK
tim@theHotandtheCool.co.uk

P. J. Bentley

Department of Computer Science
University College London
Gower Street,
London, UK
P.Bentley@cs.ucl.ac.uk

Abstract

Two novel particle swarm optimization (PSO) algorithms are used to track and optimize a 3-dimensional parabolic benchmark function where the optimum location changes randomly and with high severity. The new algorithms are based on an analogy of electrostatic energy with charged particles. For comparison, the same experiment is performed with a conventional PSO algorithm. It is found that the best strategy for this particular problem involves a combination of neutral and charged particles.

1 INTRODUCTION

Particle Swarm Optimization (PSO) is a population based evolutionary technique applied to optimization problems. It differs from other population approaches such as genetic algorithms, by the inclusion of a solution (or particle) velocity, which moves the position of the solution in the space of all possible solutions, rather than relying on recombination of existing solutions. Linear spring forces govern the dynamics of the population (or swarm); each particle is attracted to its previous best position, and to the global best position attained by the swarm, where fitness is quantified by the value of a function at that position. These swarms have proven to be very successful in finding global optima in various static contexts such as the optimization of certain benchmark functions (Eberhart and Shi 2001a).

The real world is rarely static, however, and many systems will require frequent re-optimization due to system and/or environmental change. The problem of re-scheduling system resources is an important example of this. One important and implicit constraint is the requirement to balance the desired error of the solution with the need to be prepared to respond rapidly to change. For example, to achieve a low error will require a large number of iterations/generations, and will leave the evolutionary population well adapted to that situation. But system and environment change may occur on short time-scales and may be large enough to leave the population ill-adapted to the new problem, so that a solution considered good enough may be hard to find within this time-scale.

This work addresses these issues with the use of two novel swarm algorithms. These algorithms are tested and compared with the conventional PSO algorithm for an extreme search problem wherein the optimum location (solution) is randomized within a box representing the entire dynamic range.

2 BACKGROUND

Eberhart and Shi (2001b) have applied the conventional PSO algorithm to some dynamic search and optimization problems. In their experiments, they use a time-scale of 100 iterations, and choose as a benchmark the (3-dimensional) parabolic function and the sphere function in 10-dimensions. The optimum location of these functions was moved along a line by increments of 0.2% and 1% of the dynamic range, with each change occurring at 100 iterations. It was found that, under these conditions, the PSO algorithm performed at least as well as other evolutionary techniques (Angeline 1997, Bäck 1998).

One drawback noted by Eberhart and Shi is the lack of a strategy for dealing with a wide variety of change. One possibility is to randomize the swarm when a change is detected. In their work (2001b), the particle positions are retained, but the personal and global best positions are calculated with respect to the new optimum location. Another possibility would be to randomize the swarm when a change is detected. In general, a good strategy is needed that can account for chaotic rather than linear change, and for change that is commensurate with the entire range of the dynamic variables, and not just limited to one per cent of this range.

This work investigates the capabilities of two novel swarm algorithms to overcome an extreme problem of this type. The two new algorithms were originated by the authors in quite a different context: the problem of artificial improvised music (Blackwell and Bentley

2002a). It was demonstrated that particle swarms can, if suitably interpreted as music, generate interesting melodies. Moreover, they can also interact with an external musician. External audio events are interpreted and placed in the search space, and become targets or attractors for the swarm. These targets may change on very small time-scales, and by large amounts. It was found in this work that inter-particle repulsion or "collision-avoidance" balances the target attractions and leads to an extended swarm that follows this change well. Various features of the algorithm have been reported in a subsequent paper and the suggestion made that they may have relevance to optimization problems (Blackwell and Bentley, 2002b).

The particular form of the repulsive force we have introduced is identical to the familiar electrostatic inverse square law between identically charged particles. In this paper we consider two different swarms: the first is composed entirely of identically charged particles, and the second has an equal number of charged and 'neutral' particles. Neutral particles do not experience the repulsive force. (Within this electrostatic analogy, it could be said that conventional PSO algorithms concern only neutral particles.) The idea is that the neutral particles will gather around the global best position (as if in a nucleus) whilst the charged particles will continue to explore the solution space as they orbit the nucleus. Hence there will be a balance between exploration and exploitation. This type of swarm could be termed 'atomic' since it has much in common with models of the atom. As such, it moves away from the original idea of an insect swarm or avian flock which inspired much of the early work on particle swarms.

3 THE PROBLEM

The dynamic problem investigated in this work is to find the global minimum f(**0**) of some function f($\mathbf{x}$-$\mathbf{x}_{opt}$) where $\mathbf{x}_{opt}$ is the optimum location. For dynamic search, $\mathbf{x}_{opt}$ = $\mathbf{x}_{opt}$(t), where t is an iteration counter although it could be a time variable as determined by the actual dynamic environment. Eberhart and Shi (2001b) hold $\mathbf{x}_{opt}$ fixed for 100 iterations at a time, and $\mathbf{x}_{opt}$ varies in increments of s**1,** where **1** is the unit vector in n-dimensions (linear change) for s = 0.1 and s = 0.5. The dynamic range of the variables is [-50, 50] in each dimension.

4 PARTICLE DYNAMICS

Within the PSO methodology, the particle dynamics are determined by an update rule which modifies particle velocities. New positions are then found by adding the updated velocity to the current position. The particle update algorithm used in this work is given by the application of three simple steps:

$$\mathbf{v}_i \leftarrow w\mathbf{v}_i + c_1r_1(\mathbf{x}_{pb,i} - \mathbf{x}_i) + c_2r_2(\mathbf{x}_{gb}-\mathbf{x}_i) \quad (1)$$

$$\textbf{if } (\,|\mathbf{v}_i| > v_{max}) \;\; \mathbf{v}_i \leftarrow (v_{max} / |\mathbf{v}_i|\,)\,\mathbf{v}_i \quad (2)$$

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i \quad (3)$$

In these rules, i is a particle label and each particle has a position $\mathbf{x}$ and a velocity $\mathbf{v}$ (n-dimensional vectors). The inertia weight w, and spring constants c_1 and c_2 are the adjustable parameters of the algorithm. r_1 and r_2 are random numbers drawn from the unit interval, $r_1, r_2 \in [0,1]$. $\mathbf{x}_{pb,i}$ is the best position attained by particle i and the global best location $\mathbf{x}_{gb}$ is the best position attained by any particle.

Rule (1) adds the particle accelerations from the spring forces to a damped velocity $w\mathbf{v}_i$. Rule (2) clamps the velocity to the dynamic range $[-v_{max}, v_{max}]$, which serves to limit the position increment applied in Rule (3). Notice that our rule (2) implements spherically symmetric velocity clamping, whereas other PSO algorithms clamp the velocity to a box. Since the following experiments involve qualitative observations on the spatial distribution of particles at any iteration, it is necessary to preserve spherical symmetry in the update rules.

Table 1: Search algorithm

Initialize a swarm $\{\mathbf{x}_i, \mathbf{v}_i\}$, i =1,…M, with $\mathbf{x}_i \in [0, x_{max}]^n$ and $\mathbf{v}_i \in [-v_{max}, v_{max}]^n$

Set all personal best positions to $\mathbf{x}_{pb,i}$ to $\mathbf{x}_i$

t←0

do:

  for i = 1 to M

    if f($\mathbf{x}_i$ -$\mathbf{x}_{opt}$) < f($\mathbf{x}_{pb,i}$ -$\mathbf{x}_{opt}$)

      then $\mathbf{x}_{pb,i}$← $\mathbf{x}_i$

    if f($\mathbf{x}_{pb,i}$ -$\mathbf{x}_{opt}$) < f($\mathbf{x}_{gb}$-$\mathbf{x}_{opt}$)

      then $\mathbf{x}_{gb}$← $\mathbf{x}_{pb,i}$

  endfor

  if (t%100 = 0)

    then $\mathbf{x}_{opt} \in [(x_{max}/2)\text{-}L/2, (x_{max}/2)+L/2]^3$

  for i = 1 to M

    Apply particle update algorithm (1) – (3)

  endfor

  t←t+1

until stopping criterion is met

This clamping is a constraint on global exploration. The balance between this and local exploitation of good solutions is given by the inertial weight. In the non-dynamic case it is advantageous to reduce w from 1 down to near zero during the course of training run, since this allows full exploitation of possible good solutions (Eberhart and Shi 2001a). However, in the dynamic case it cannot be predicted whether exploration or exploitation is needed at any given time. With these factors in mind, Eberhart and Shi (2001b) used a value of 1.494 for the spring constants and a random inertia weight w ∈ [0.5, 1]. These values were chosen to agree, on the average,

with Clerc's analysis for convergence (Clerc 1999), and to provide a balance between exploration and exploitation.

The PSO algorithm for the dynamic problem investigated here is given in table 1.

In order to introduce the notion of charge – and hence collision avoidance – all that needs to be done is to modify the rule for particle accelerations, rule (1). The grounds for this extension, and a full description of the effects of the various parameters on particle motion, are given in (Blackwell, 2001) and (Blackwell and Bentley, 2002b). In those studies, an additional acceleration towards the swarm centre was also implemented, but this is not used here.

The necessary amendment to the particle update algorithm is an extra particle acceleration $\mathbf{a}_i$ given by

$$\mathbf{a}_i = \sum_{j \neq i} \frac{Q_i Q_j}{r_{ij}^3} \mathbf{r}_{ij}, \qquad p_{core} < r_{ij} < p \qquad (4)$$

where $\mathbf{r}_{ij} = \mathbf{x}_i - \mathbf{x}_j$, $r_{ij} = |\mathbf{x}_i - \mathbf{x}_j|$ and each particle has a charge of magnitude Q_i. Neutral particles are assigned a charge $Q_i = 0$ and so will not contribute to the sum in (4). A charged particle i will have $Q_i > 0$ and will experience the repulsive effects from all other charged particles $j \neq i$. Repulsion is only experienced for separations within the shell $p_{core} < r_{ij} < p$. The lower cut-off p_{core} is a safeguard against the singularity of the inverse square law. The upper cut off p is a tunable parameter allowing the domain of influence of the repulsion to be controlled.

The charged particle update algorithm, which replaces the particle update algorithm within the search algorithm (Table 1), is given by replacing rule (1) with:

$$\mathbf{v}_i \leftarrow w\mathbf{v}_i + c_1 r_1(\mathbf{x}_{pb,i} - \mathbf{x}_i) + c_2 r_2(\mathbf{x}_{gb} - \mathbf{x}_i) + \mathbf{a}_i \quad (1')$$

It is worth noting that since the particles are updated in turn (i.e. from i = 1 to i = M), contributions to $\mathbf{a}_i$ can involve non-updated particle positions (j > i) as well as updated positions (j < i). This was found to give better avoidance in earlier experiments.

5 EXPERIMENTS

The experiments were conducted on the parabolic function in n = 3 dimensions, $f(\mathbf{x}-\mathbf{x}_{opt}) = (\mathbf{x}-\mathbf{x}_{opt}) \bullet (\mathbf{x}-\mathbf{x}_{opt})$.

The task is made dynamic by placing $\mathbf{x}_{opt}$ in a cube of side L and then randomly re-positioning it to another point in this cube every 100 iterations. The severity *s* therefore varies randomly from zero up to $(\sqrt{3})L$. With L set to $2v_{max}$, this gives a severity of up to $2\sqrt{3}$ times the dynamic range. The parameter x_{max} solely determines the initial distribution of the particle velocities and positions and plays no part in subsequent updates after the first jump in $\mathbf{x}_{opt}$. The values of the spatial parameters set out in Table 2.

Table 2: Spatial parameters

L	x_{max}	v_{max}	M
64	128	32	20

The values of the electrostatic parameters p_{core}, p and Q are set out in Table 3.

Table 3: Electrostatic parameters

p_{core}	p	Q
1	$\sqrt{3}x_{max}$	16

The values of the PSO parameters w, c_1 and c_2 are those used by Eberhart and Shi (2001b). The inertia weight w varies randomly between 0.5 and 1.0, so that the mean 0.75 is close to the Clerc constriction factor 0.729 (Clerc 1999). The spring constants c_1 and c_2 are set to 1.494, also in accordance with Clerc's analysis.

In all these experiments, 50 optimum jumps are made, (or 5000 iterations of the system). In addition to the numerical data produced by these experiments, a three dimensional animation was set up which enabled a qualitative assessment of the three swarms.

For comparison, four experiments were performed:

I Neutral swarm.

The first experiment uses the conventional PSO algorithm, which is implemented by setting the charge on all 20 particles to zero (i.e. each particle is neutral).

II Charged swarm.

In the second experiment, all 20 particles carry the same charge, Q. In other words, all particles experience repulsive forces from the other particles.

III Atomic swarm.

The third experiment evaluates the *atomic swarm*, where 10 particles have charge Q and the remaining half are neutral.

IV Neutral swarm, one optimum jump.

The fourth experiment is identical to Experiment 1 except that just 200 iterations were allowed. The positions and velocities of the particles were saved to file for analysis of individual particle motion.

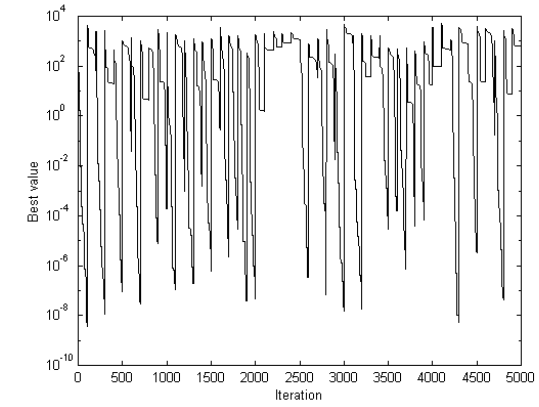


Figure 1: Neutral swarm

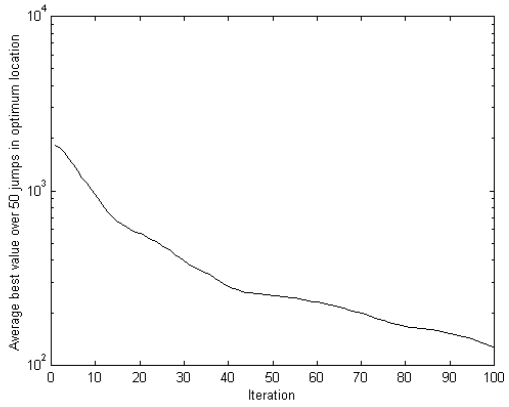


Figure 2: Neutral swarm, average best values per 100 iterations.

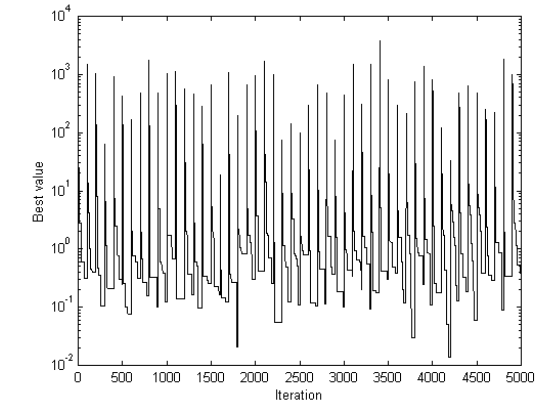


Figure 3: Charged swarm

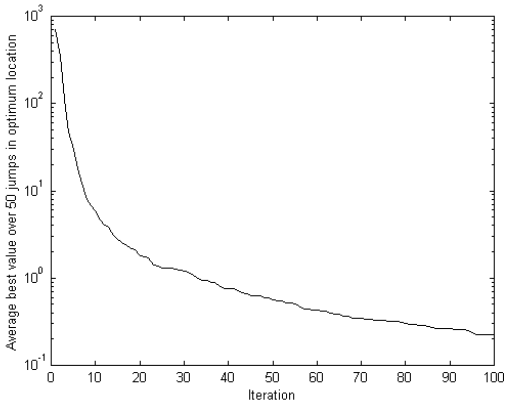


Figure 4: Charged swarm, average best values per 100 iterations.

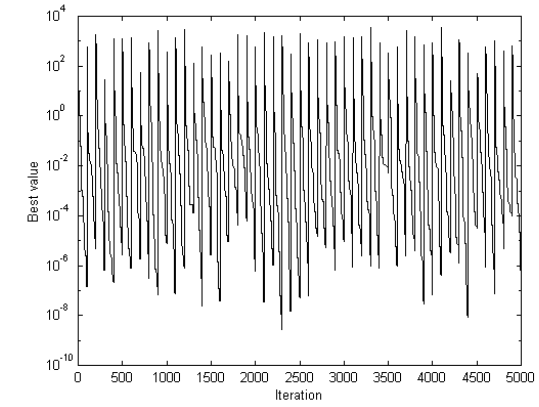


Figure 5: Atomic swarm

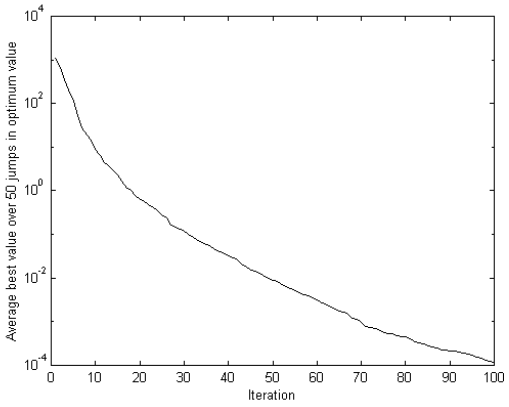


Figure 6:Atomic swarm, average best values per 100 iterations.

6 RESULTS

The results for the experiments on the neutral, charged and atomic swarms are shown in Figures 1 to 6. In each case, two graphs have been prepared, plotted as a function of iteration number t: the best value found by the swarm, $f(\mathbf{x}_{gb} - \mathbf{x}_{opt})$, and the average best value over the 50 problem optimum jumps (i.e., average best per 100 iterations, from optimum jump to the iteration before the next optimum jump).

6.1 EXPERIMENT I: NEUTRAL SWARM

The best value attained after 100 iterations, i.e. just before the first optimum jump, is of the order of 10^{-8}. This is at least two orders of magnitude lower than the best value obtained in the first 100 iterations of an equivalent experiment (Eberhart and Shi 2001b figure 1, p98), although comparable to the best values obtained in subsequent optimum positions. The discrepancy is presumably due to initial conditions, although the spherically symmetric clamping rule may play a part.

However, the results depart significantly at the first optimum jump, due to the increased severity. The remarkable feature of Figure 1 is the small spikes at the optimum jump followed by a leveling of the graph for some tens of iterations, after a short fall. This plateau in best values sometimes then drops before the next optimum jump, but often does not. For example, there is a run from t = 2100 to 2500 where the initial short fall is not improved upon. The plot of the averages over the 50 optimum jumps shows an average best of 125 at 100 iterations. The slope at this point is -3.2, indicating only a slow improvement (3% per iteration) with increasing iteration.

The 3D animations showed a very unusual feature. For the first 100 iterations, the particles were clumped very closely around the optimum. At the optimum jump, however, the particles moved along a line in the general direction of the new optimum, and then began to oscillate along this line about a point close, but not adjacent to the new optimum. After some tens of iterations the oscillations would cease and the particles would begin to swarm towards the new optimum, although they might not reach it before the next jump. This behavior was repeated invariably at each optimum jump, and in repeats of this experiment.

6.2 EXPERIMENT II: CHARGED SWARM

Figure 3 shows the best values over the 5000 iterations. By comparison with Figure 1, the spikes are now long, showing an improved best value by a factor of 10^3 after just a few iterations at each jump. The leveling out now occurs at a much smaller best value, a feature illustrated in Figure 4 which shows the average best values. In Figure 4, the lowest average best value obtained is 0.226, and the slope at this point is -2.10×10^{-3} showing an improvement of 1% per iteration at this point.

The animations revealed typical swarming behavior: at each optimum jump the swarm moved towards the new optimum, with irregular motion about the swarm centre. After a few iterations the swarm centre was coincident with the optimum and the particle motion continued to be chaotic and spherically symmetric about this point, with particles amplitudes of some tens of units. These pictures agreed with previous swarm experiments (Blackwell and Bentley, 2001b).

6.3 EXPERIMENT III: ATOMIC SWARM.

Once more, the plot of best values, Figure 5, shows spikes at each optimum jump, but the spikes drop to a much lower best value, in the range 10^{-4} to 10^{-8} in 49 of the 50 jumps. The figure does not show the plateaus that are a feature of Figures 1 and 3. The plot of average best values, Figure 6, shows a much improved average best at 100 iterations of 1.12×10^{-4}, with a slope of -1.21×10^{-5} or 11% of the best value per iteration at this point. The average global best just before the next optimum jump is at least 6 orders of magnitude better than the neutral swarm and about 2000 times better than the charged swarm.

In order to distinguish charged from neutral particles for the purposes of the animation, the particles were colored red (charged) and blue (neutral). At each optimum jump, the animations displayed the particles moving in an irregular swarming motion towards the optimum, followed by a long period where the blue neutral particles clumped around the optimum, moving ever slower with very small amplitude, surrounded with a 'cloud' of charged red particles, moving much like the charged swarm described in II. The picture was very reminiscent of representations of an atomic nucleus surrounded by an electron cloud.

6.4 EXPERIMENT IV: NEUTRAL SWARM, ONE OPTIMUM JUMP.

A further experiment was conducted on the neutral swarm to give greater insight on the linear non-swarming behavior of the particles just after the optimum jump, as observed in the animation. Just two hundred iterations were completed, allowing a single optimum jump to occur.

At t =100, the 20 particles were clumped very tightly around x_o, and moving slowly. Between t = 100 and t = 120, all the particles followed a very similar trajectory. For this reason, some results for just a single particle, particle 0, will be presented. Table 4 shows x, y and z components of the optimum location, global best location and position and velocity of particle 0 at iteration 99, just before the effects of the optimum jump have influenced the particle dynamics.

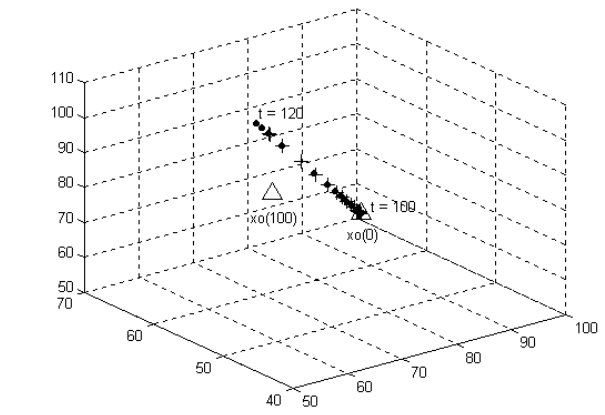


Figure 7: Position of particle 0 for t = 100 to t = 120

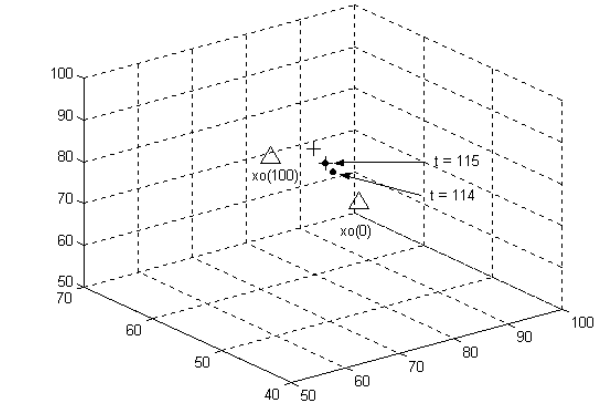


Figure 8: Snapshots at t = 114 and t = 115

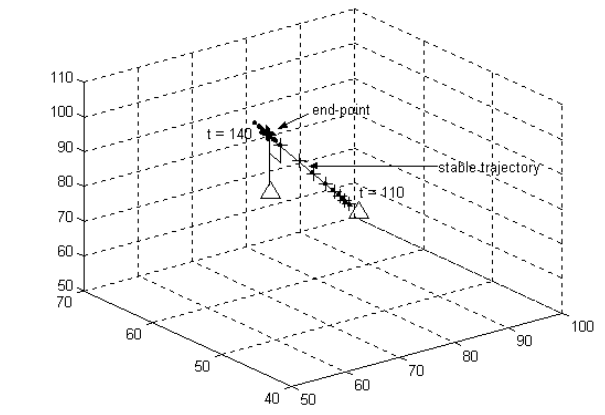


Figure 9: Position of particle 0 for t = 110 to t = 140

Table 4: Opt., best & particle 0 components at t = 99.

	x_{opt}(100)	x_{gb}(100)	x_0(100)	v_0(100)
x	95.71089	95.710884	95.5921	0.61443764
y	65.98201	65.98204	65.87358	0.52882737
z	56.992935	56.992916	57.107887	-0.58677804

Figure 7 shows the positions of particle 0 (circles) and the global best (+'s) for iterations t = 100 to t = 120. The optimum location is depicted with a triangle. Figure 8 shows just two snapshots at t = 114 and t = 115. Finally, Figure 9 shows the position of particle 0 and global best between iterations 110 and 140. For the purpose of the subsequent analysis, a line showing the stable trajectory and its end point has been marked on the figure.

7 ANALYSIS

The strange behavior of the neutral swarm just after the optimum jump is the crucial difference between Experiments I and II. This analysis section will start with a possible explanation for this phenomenon.

The situation for the neutral swarm just at and after the optimum jump must be studied, and the optimum jump of Experiment IV is a good place to start. At t = 100, the closest particle, k, to the new optimum location will now be at the new global best. Particle k will at this stage experience no acceleration, and therefore its next location is $\mathbf{x}_k(100) = \mathbf{x}_k(99) + w\mathbf{v}_k(99)$. Notice that the velocity $\mathbf{v}_k(99)$ is unlikely to be pointing towards $\mathbf{x}_{opt}(100)$. However, if $\mathbf{v}_k(99)$ has some component that lies along $\mathbf{x}_{opt}(100) - \mathbf{x}_k(99)$ then the new position $\mathbf{x}_k(100)$ will improve upon the previous position and may even be the new global best when the updates at this iteration are completed. Suppose this is so. Then, by a similar argument, $\mathbf{x}_k(101)$ will lie along the same trajectory $\mathbf{x}_k(100) - \mathbf{x}_k(99)$. Meanwhile the other particles, which were at their personal bests at t = 99, will experience accelerations towards $\mathbf{x}_k(99)$ of magnitude $c_1 r_1 |\mathbf{x}_k(99) - \mathbf{x}_i(99)|$. This will not be a large acceleration, but it will give the new velocity vector $\mathbf{v}_i(100)$, an additional component along the trajectory defined by $\mathbf{x}_k(100) - \mathbf{x}_k(99)$. At the next iteration, if the above scenario is played out, velocity components along the trajectory of particle k will again be reinforced.

Of course there may well be some jostling for leadership, but occasionally a leader will be found whose trajectory defines a line of global bests. The remaining particle velocities are pulled ever more in the direction of this trajectory and the accelerations place them ever closer to positions along this trajectory. The result is collinear motion along a line that is closing on $\mathbf{x}_{opt}$ but is not necessarily coincident with $\mathbf{x}_{opt}$. The leadership may then be exchanged, but motion along this line will always be reinforced until a final global best position is found which

is near to the point of closest approach between the trajectory – the 'end-point' - and $\mathbf{x}_{opt}$. Animations of many runs of the swarm provide empirical evidence that this scenario invariably occurs.

The stable trajectory is clearly seen in Figure 7. Figure 8 shows two snapshots that illustrate the attraction of the particles to the trajectory. At t = 114, particle 0 is displaced from the global best position. The acceleration is sufficient to place it very close to $\mathbf{x}_{gb}(114)$ at t = 115, but the global best has now moved along the stable trajectory to $\mathbf{x}_{gb}(115)$.

Consider now what happens when the global best is near to the end-point. Velocities perpendicular to the stable trajectory will be very small so that accelerations towards $\mathbf{x}_{opt}(100)$ will also be correspondently small. Moreover, the dissipative effects of the inertia weight will also be progressively slowing particle motion down. The result is that it may take a long time for the swarm to move away from the end of the stable trajectory, and the global best will hardly improve in the remaining iterations before the next optimum jump. The particle motion is now a spring-like oscillation along the stable trajectory, centered on the end-point. The stable trajectory and end-point are depicted in Figure 9 which shows the position of particle 0 and global best between iterations 110 and 140.

This analysis can be applied to the results of Experiment 1 (Figure 1). The plateau show that the global best scarcely improves over some tens of iterations. This is due to the collinear motion followed by oscillation about the end-point of the stable trajectory. In fact, there is a particularly bad run between iterations 2100 and 2500 when the particles never improve on their global best, which corresponds to an optimum value of 1000. This is 11 orders of magnitude from the best that the swarm is capable of finding (10^{-8}). It is these high values that push up the average best value found (Figure 2).

The charged swarm is less affected by this pathology since the collision avoiding acceleration will push particles away from the stable trajectory. In fact animations never show linear collapse; instead, the swarm maintains a near spherical shape, much more reminiscent of an insect swarm. However, this also has its drawbacks. Figure 3 does show some horizontal portions, for global bests in the range 10^{-1} to 1. The repulsions now work against exploitation so that better solutions than this are found in only 7 of the 50 optimum jumps.

The atomic swarm also does not suffer from the pathology of the neutral swarm. The charged particles allow for fast targeting, after which the neutral particles can continue searching the solution space in the near vicinity of the global best. Indeed, at the 100th iteration, the rate of improvement of best value is 11%, which shows that significant improvement can still occur. The corresponding rates for the neutral swarm and the charged swarm are 3% and 1%, indicating only slow progress is possible.

8 CONCLUSIONS

This work presents a new particle swarm algorithm based on an analogy of electrostatic energy. In addition, a dynamic search problem has been formulated that is more representative of real-world problems. The experiments considered here suggest that atomic particle swarms may offer a good strategy for dealing with such severe dynamic optimization over short time scales. Certainly this has been the case for the dynamic three dimensional parabolic function considered here, where the average best value obtained over 50 optimum jumps was, by the 100th iteration, 1.12×10^{-4}. This compares very well with the equivalent figure of 125 for the conventional PSO algorithm.

The poor behavior of the conventional (i.e., neutral) particle swarm seems to be due to a curious pathology of 'linear collapse', just after problem optimum jump. This was observed in animations and analysis suggests that the cause is the establishment of a linear trajectory that links global best positions and serves as an attractor for the swarm. At the end of this stable trajectory is a stable global best position, which can be some way from the optimum location, and from which the swarm has difficulty improving upon.

The charged particle swarm has the advantage that the particle trajectories are always around an extended swarm shape, allowing good global search. The maintenance of an extended swarm was the reason for the use of collision avoidance in the earlier work on improvised music (Blackwell and Bentley 2002a,b). In this context, it is desirable to have a very fast swarm response to a changing audio input, yet undesirable for the swarm to cluster too closely around a target – this would lead to dull melodies and parody. The downside is that the particle repulsions prevent detailed exploration of the search space.

However, experiments suggest that a swarm of neutral and charged particles (reminiscent to representations of the atom) does not suffer from linear collapse, and always allows for detailed exploitation. The advantage of an atomic swarm over randomizing strategies (e.g., where the particle positions are randomized when an problem optimum shift is noticed) is one of simplicity. No further analysis is needed to tell just when a change has occurred, and how the swarm should respond to this change.

References

P.J.Angeline (1997). Tracking extrema in dynamic environments. *Proc. Evolutionary Programming VI*, Indianapolis, IN. Berlin: Springer-Verlag (Berlin), pp 335-345.

T. Bäck (1998). On the behavior of evolutionary algorithms in dynamic environments. *Proc. Int. Conf. on Evolutionary Computation*, Anchorage, AK. Piscataway, NJ: IEEE Press pp 446-451

Blackwell, T. (2001) "Making Music with Swarms," *MSc thesis, University College London.*

Blackwell, T. and Bentley, P. J. (2002a) Improvised Music with Swarms, *Proc CEC 2002* (to be published).

Blackwell, T. and Bentley, P. J. (2002b) Don't push me! Collision-Avoiding Swarms, *Proc CEC 2002* (to be published).

M.Clerc (1999). The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. *Proc CEC 1999*. Washington, DC, pp 1951 – 1957.

R.C.Eberhart and Y.Shi (2001a). Particle swarm optimization: Developments, applications and resources. *Proc CEC 2001*. Piscataway, NJ, pp 81-86.

R.C.Eberhart and Y.Shi (2001b). Tracking and optimizing dynamic systems with particle swarms. *Proc CEC 2001*. Piscataway, NJ, pp 94-100.

Ant Colony Optimization for the Edge-Weighted k-Cardinality Tree Problem

Christian Blum
IRIDIA
Université Libre de Bruxelles
Brussels, Belgium
e-mail: cblum@ulb.ac.be
tel: (32) 2 650 3168, fax: (32) 2 650 2715

Abstract

In this paper we deal with an *NP*-hard combinatorial optimization problem, the k-cardinality tree problem in edge-weighted graphs. This problem has several applications in practice, which justify the need for efficient methods to obtain good solutions. Metaheuristic applications have already been shown to be successful in tackling the k-cardinality tree problem in the past. In this paper we propose an ACO algorithm for the edge-weighted k-cardinality tree problem based on the Hyper-Cube Framework for Ant Colony Optimization. We investigate the usefulness of a higher order pheromone representation in contrast to the standard first order pheromone representation and compare our algorithms to a multi–start local search and a heuristic developed to tackle the problem.

1 Introduction

The k-cardinality tree problem is a combinatorial optimization problem which generalizes the well known minimum weight spanning tree problem. It consists in finding in a node- or edge-weighted graph $G = (V, E)$ a subtree with exactly k edges, such that the sum of the weights is minimal. Due to various applications, e.g. in oil-field leasing [23], facility layout [18], quorum-cast routing [9] and telecommunications [21] it has gained considerable interest in recent years. In this paper we will deal with the k-cardinality tree problem in edge-weighted graphs. The problem can be formally defined as follows. Let $G = (V, E)$ be a graph with a weight function $w : E \rightarrow I\!N$ on the edges. We denote by $\mathcal{T}_k$ the set of all k-cardinality trees in G. Then the edge-weighted problem (G, w, k) is to find a k-cardinality tree $T_k \in \mathcal{T}_k$ which minimizes

$$w(T_k) = \sum_{e \in E(T_k)} w(e). \quad (1)$$

Several authors have proved independently that the edge-weighted k-cardinality tree problem (1) is *NP*-hard, see [17, 26]. In [26] it has been shown that it is still *NP*-hard if $w(e) \in \{1, 2, 3\}$ for all edges e and $G = K_n$, but polynomially solvable if there are only two distinct weights. Several authors have considered special types of graphs. One of the results is that the problem is polynomially solvable if G is a tree (see [25]). The edge-weighted problem is *NP*-complete for planar graphs and for points in the plane, when edge weights correspond to distances between the points (see [26]). In the same paper polynomial algorithms for decomposable graphs and graphs with bounded tree-width have been given. There is also a polynomial algorithm for the case when all points lie on the boundary of a convex region. In [14], the authors have focused on properties of the distance matrix. They have assumed that $G = K_n$ and have proved several results (both *NP*-completeness and polynomial time solvability) on the complexity of the problem with graded distance matrices.

Concerning methodology, both exact and heuristic algorithms have been developed, with a general focus on approximation algorithms. We first note that integer programming formulations have been presented in [17] and later in [20]. Based on detailed studies of the associated polyhedron in the former paper a Branch and Cut algorithm has been developed and implemented in [19]. The code and also implementations of most of the heuristics in [16] are documented in [15]. A Branch and Bound method is described in [9]. The heuristics mentioned are based on greedy and dual greedy strategies and also make use of dynamic programming approaches. Other constructive heuristics have been presented in [9].

More recently, authors successfully applied metaheuristic methods to the k-cardinality tree problem (see Tab. 1 for an overview). Metaheuristics[1] include but are not restricted to Simulated Annealing (SA), Evolutionary Computation (EC) with its most famous representative the Genetic Algorithm (GA),

[1] See [5] for an overview on metaheuristics.

Table 1: An overview of metaheuristic approaches to tackle the k-cardinality tree problem.

Publication	Problem-type	Metaheuristic
M.J. Blesa and F. Xhafa [2], 2000	edge-weighted	TS
M.J. Blesa, P. Moscato and F. Xhafa [1], 2001	edge-weighted	Memetic Algorithm
C. Blum [3], 1998 C. Blum and M. Ehrgott [4], 2001	node-weighted	TS and EC
F. Catanas [8], 1997	node-weighted + edge-weighted	TS and EC
K. Jornsten and A. Lokketangen [24], 1997	edge-weighted	TS
N. Mladenovic [27], 2001	edge-weighted	VNS

Tabu Search (TS), explorative search methods such as Iterated Local Search (ILS), and Ant Colony Optimization (ACO). Among the metaheuristics applied to the k-cardinality tree problem are Evolutionary Computation, Tabu Search, and Variable Neighborhood Search (VNS) (see Tab. 1). The aim of this paper is to show how Ant Colony Optimization can be successfully applied to the edge-weighted k-cardinality tree problem. We investigate the usefulness of a higher order pheromone representation in contrast to the standard first order representation and compare the results obtained by our algorithms to a multi–start local search and a heuristic developed to tackle the problem.
The remainder of the paper is organized as follows. In Sec. 2 we briefly outline the concepts of Ant Colony Optimization and a particular way of implementing ACO algorithms, called the Hyper-Cube Framework. In Sec. 3 we present the framework and the components of the ACO algorithm to tackle the edge-weighted k-cardinality tree problem. In Sec. 4 we present results and finally in Sec. 5 we draw some conclusions and give an outlook to future work.

2 Ant Colony Optimization

Ant Colony Optimization (ACO) [10, 13, 11] is a recently proposed metaheuristic approach for solving hard combinatorial optimization problems. The inspiring source of ACO is the foraging behavior of real ants. This behavior enables them to find shortest paths between food sources and their nest. While walking from food sources to the nest and vice versa, ants deposit a substance called *pheromone* on the ground. When they decide about a direction to go they choose, in probability, paths marked by strong pheromone concentrations. This basic behavior is the basis for a cooperative interaction which leads to the emergence of shortest paths.
In ACO algorithms, an artificial ant incrementally constructs a solution by adding opportunely defined solution components to a partial solution under consideration[2]. For doing that, artificial ants perform randomized walks on a completely connected graph $\mathcal{G}_c = (\mathcal{C}, \mathcal{L})$ whose vertices are the solution components $\mathcal{C}$ and the set $\mathcal{L}$ are the connections. This graph is commonly called *construction graph*. The problem constraints Ω are built into the ants' constructive procedure in a way such that in every step of the construction process only feasible solution components are permitted to be added to the current partial solution. In ACO algorithms we work with a set of *pheromone values* τ and also with a set of *heuristic values* η. These values are used by the ants' heuristic rule to make probabilistic decisions on how to move on the construction graph. The probabilities involved in moving on the construction graph are commonly called *transition probabilities*.
The first ACO algorithm proposed was Ant System (AS) [13]. Although AS is important, because it was the first ACO algorithm proposed, in the last few years some changes and extensions of AS have been proposed, e.g. Ant Colony System (ACS) [12] and $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [29]. In general, ACO algorithms have been proven to be a very effective – for some problems like the QAP even the state-of-the-art – metaheuristic method for combinatorial optimization problem solving.

2.1 The Hyper-Cube Framework

The Hyper-Cube Framework – recently proposed by Blum et al. [6] – is a certain way of implementing ACO algorithms. This way of implementing ACO algorithms comes with several benefits. Maybe the most important one is the property of scaling objective function values.
To a set of pheromone values $\tau = \{\tau_1, ..., \tau_n\}$ in ACO algorithms usually a pheromone updating rule of the following kind is applied.

$$\tau_i \leftarrow (1-\rho) \cdot \tau_i + \sum_{j=1}^{n_s} \Delta^j \tau_i \qquad (2)$$

where

$$\Delta^j \tau_i = \begin{cases} f(s^j) & \text{if } s^j \text{contributes to } \tau_i \\ 0 & \text{otherwise .} \end{cases} \qquad (3)$$

[2]Therefore, the ACO metaheuristic can be applied to any combinatorial optimization problem for which a constructive heuristic can be defined.

$\Delta^j\tau_i$ is the contribution of a solution s^j to the update for pheromone value τ_i (n_s is the number of solutions used for updating the pheromone values), ρ is the evaporation rate (a small positive constant), and f is a function which is monotone in the quality of the solution (for minimization problems it usually maps the quality of a solution to its inverse). In the Hyper-Cube Framework a normalization of the contribution of every solution used for updating the pheromone values is done in the following way.

$$\tau_i \leftarrow (1-\rho)\cdot\tau_i + \rho\cdot\textstyle\sum_{j=1}^{n_s}\Delta'^j\tau_i \quad (4)$$

where

$$\Delta'^j\tau_i \;=\; \begin{cases} \frac{f(s^j)}{\sum_{l=1}^{n_s} f(s^l)} & \text{if } s^j \text{ contributes to } \tau_i \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

where we multiply the sum of normalized contributions with the evaporation rate ρ. This formula can be reformulated as:

$$\tau_i \leftarrow \tau_i + \frac{\rho}{\sum_{l=1}^{n_s} f(s^l)}\left(\sum_{j=1}^{k} f(s^j)\cdot\delta(s^j,\tau_i) - \tau_i\right) \quad (6)$$

where

$$\delta(s^j,\tau_i) \;=\; \begin{cases} 1.0 & \text{if } s^j \text{ contributes to } \tau_i \\ 0.0 & \text{otherwise} \end{cases} \quad (7)$$

This leads to a scaling of the objective function values and the pheromone values are implicitly limited to the interval $[0,1]$ (see [6] for a more detailed description).

3 ACO for the k-cardinality tree problem

In this section we outline the framework of our ACO algorithm for the edge-weighted k-cardinality tree problem. The basic framework of our algorithm is shown in Alg. 1. In Alg. 1, $\tau = \{\tau_1, ..., \tau_n\}$ is a set of pheromone values, n_a is the number of ants used in every iteration, ${T_k}^j$ are solutions to the problem, cf is a numerical value which we called the convergence factor, ${T_k}^{ib}$ is the iteration best solution, ${T_k}^{rb}$ is the restart best solution and ${T_k}^{gb}$ is the best solution found from the start of the algorithm.

InitializePheromoneValues(τ): In every version of our algorithm we initialize all the pheromone values to 0.5.

ConstructSolution(τ): To tackle the k-cardinality tree problem with an ACO algorithm we have to define the constructive heuristic to be used in a probabilistic manner to construct solutions to the problem. In ACO algorithms artificial ants construct a solution by building a path on a *construction graph* $\mathcal{G} = (\mathcal{C}, \mathcal{L})$ where the elements of the set $\mathcal{C}$ (called *components*) and the elements of the set $\mathcal{L}$ (called *links*) are given for the k-cardinality tree problem as follows:

$$\begin{aligned} \mathcal{C} &= E(G) \cup \{c_{source}, c_{sink}\} \\ \mathcal{L} &= \{(e_i,e_j) \mid e_i, e_j \in E(G), e_i \neq e_j\} \\ &\cup \{(c_{source}, e) \mid e \in E(G)\} \\ &\cup \{(e, c_{sink}) \mid e \in E(G)\} \end{aligned}$$

Note that all links in $\mathcal{L}$ are directed. This graph $\mathcal{G}$ is fully connecting the edges of G (which are the components of $\mathcal{G}$) plus a source component c_{source} (and arcs from the source component to every component of $\mathcal{G}$) and a sink component c_{sink} (and arcs from every component in $\mathcal{G}$ to the sink component).
To build a solution an ant starts from the source component c_{source} of the construction graph and does k construction steps as shown in Alg. 2. In every step of the ant construction phase we can only add an

Algorithm 1 ACO for the k-cardinality tree problem

```
input: a problem instance (G, w, k)
T_k^gb ← NULL
T_k^rb ← NULL
cf ← 0
InitializePheromoneValues(τ)
while termination conditions not met do
  for j = 1 to n_a do
    T_k^j ← ConstructSolution(τ)
    LocalSearch(T_k^j)
  end for
  T_k^ib ← argmin(w(T_k^1), ..., w(T_k^{n_a}))
  ApplyPheromoneUpdate(cf,τ,T_k^ib,T_k^rb,T_k^gb)
  Update(T_k^ib,T_k^gb,T_k^rb)
  cf ← ComputeConvergenceFactor(τ, T_k^ib)
  if algorithm converged then
    ResetPheromoneValues(τ)
    T_k^rb = NULL
  end if
end while
output: T_k^gb
```

Algorithm 2 Ant construction phase

```
Ant is placed on c_source
J_1 = {e = [v_r, v_s] | e ∈ C}
for t = 1 to k do
  Choose e* = [v_i, v_j] ∈ J_t to probability p(e*|T_t)
  Ant moves to the component associated with e*
  E(T_t) = E(T_t) ∪ e*
  V(T_t) = V(T_t) ∪ {v_i, v_j}
  J_{t+1} = {e = [v_r, v_s] | e ∉ E(T_t), either v_r ∈
  V(T_t) or v_s ∈ V(T_t)}
end for
Ant moves to c_sink
```

edge $e = [v_i, v_j]$ to the partial k-cardinality tree T_t ($t \in \{1, k-1\}$) if exactly one of the two nodes incident with this edge (v_i, or v_j) is already in the node set $V(T_t)$ of T_t. The generation of the transition probabilities $p(e|T_t)$ for all $e \in J_t$ is dependent on the pheromone representation to be explained in the following.

There are a number of design decisions to be made when developing an ACO algorithm to tackle a combinatorial optimization problem. One of the most crucial decisions is the choice of a pheromone model. For the TSP for example it is a fairly obvious choice to put a pheromone value on every link between a pair of cities. For other combinatorial optimization problems the choice is not as obvious as for the TSP (see [28] for MAX-SAT, or [7] for FOP Shop scheduling). Often this problem can be stated as the problem of assigning pheromone values to the decision variables themselves (first order pheromone values) or to subsets of decision variables (higher order pheromone values). In the following we present two different pheromone representations (models) for the edge-weighted k-cardinality tree problem.

1) Pheromone values on decision variables: This first pheromone model (called $\mathsf{PH}_{\mathsf{1storder}}$ further on) is the most simple choice of a pheromone representation for the edge-weighted k-cardinality tree problem. To every edge $e_i \in E(G)$[3] we have associated a pheromone value τ_{e_i}. Therefore, if $|E(G)| = m$ we have m pheromone values. The probabilities $p(e_i|T_t)$ for the edges in set J_t of the ant construction phase to be chosen by the ant (called transition probabilities) are determined as follows. If $t = 1$ the transition probabilities are

$$p(e_i|T_1) = \begin{cases} \frac{\tau_{e_i}}{\sum_{e_l \in J_1} \tau_{e_l}} & : \text{ if } e_i \in J_1 \\ 0 & : \text{ otherwise} \end{cases} \quad (8)$$

where J_1 is the set of operations allowed to be scheduled next (see Alg. 2). As a good edge (an edge with a low weight) is not necessarily a good starting point for building a low weight k-cardinality tree, we decided not to use any heuristic information in this formula. This is different for the next $k-1$ construction steps. For $t > 1$ the transition probabilities are

$$p(e_i|T_t) = \begin{cases} \frac{\tau_{e_i} \cdot \frac{1}{w(e_i)}}{\sum_{e_l \in J_t} \tau_{e_l} \cdot \frac{1}{w(e_l)}} & : \text{ if } e_i \in J_t \\ 0 & : \text{ otherwise} \end{cases} \quad (9)$$

This means that for the second and consecutive steps the distribution given by the pheromone values is influenced by the weights of the edges. Low edge weights result in a higher probability to be chosen by the ants and the other way around. With this pheromone representation the algorithm tries to learn for every edge the desirability of having it in a solution. This pheromone model doesn't take into account any dependencies between decision variables.

2) Pheromone values on pairs of decision variables: This pheromone model (called $\mathsf{PH}_{\mathsf{2ndorder}}$ further on) takes into account dependencies between decision variables. To every pair $< e_i, e_j >$ (where $e_i \neq e_j$) of edges in $E(G)$ we have associated a pheromone value $\tau_{<e_i,e_j>}$ (where $\tau_{<e_i,e_j>}$ and $\tau_{<e_j,e_i>}$ are the same). We also use the pheromone values of the pheromone model $\mathsf{PH}_{\mathsf{1storder}}$ for the first construction step (when $t = 1$). Therefore, in this model we have $m + (m^2 - m)$ pheromone values. If $t = 1$ the transition probabilities $p(e_i|T_1)$ are generated as shown in equation (8). If $t > 1$ the transition probabilities $p(e_i|T_t, t)$ are generated as follows:

$$\begin{cases} \frac{\left(\sum_{e_j \in E(T_t)} \tau_{<e_j,e_i>}\right) \cdot \frac{1}{w(e_i)}}{\sum_{e_l \in J_t} \left(\sum_{e_j \in E(T_t)} \tau_{<e_j,e_l>}\right) \cdot \frac{1}{w(e_l)}} & : \text{ if } e_i \in J_t \\ 0 & : \text{ otherwise} \end{cases} \quad (10)$$

where J_t is as described above. With this pheromone representation the algorithm tries to learn for every pair of edges the desirability of having them together in a solution. As we have pheromones on pairs of edges, this pheromone model takes into account all first order dependencies between decision variables.

$\mathsf{LocalSearch}(T_k{}^j)$: The most important ingredient of a local search method is the neighborhood function. Let T_k be a k-cardinality tree. The neighborhood $\mathcal{N}_{Swap}(T_k)$ of a k-cardinality tree T_k consists of all k-cardinality trees which can be generated from T_k by cutting off one of the leaf edges e from T_k and adding one edge from the neighborhood of $T_k \setminus e$. This neighborhood function has the advantage to be easy to compute, but it is probably coming with the disadvantage of quite a few low quality local minima. However we decided to use this simple neighborhood function in a steepest descent local search (best improvement) in order not to spend a too high percentage of the computation time on the local search.

$\mathsf{ApplyPheromoneUpdate}(cf, \tau, T_k{}^{ib}, T_k{}^{rb}, T_k{}^{gb})$: For updating the pheromone values we are using a so-called *online delayed pheromone update rule*. We always use 3 different solutions for updating the pheromone values[4], the best solution found in the current iteration $T_k{}^{ib}$, the restart best solution $T_k{}^{rb}$ and the best solution found since the start of the algorithm $T_k{}^{gb}$. In contrast to the usual updating rule of the Hyper-Cube Framework as shown in equation (6), in our updating rule the influence of each on of these 3 solutions is dependent one the state of convergence of the algorithm

[3] We consider the edges of graph G to be the decision variables of the problem.

[4] Note that a similar scheme was used in [29].

(given by the convergence factor cf) rather than by the quality of the solutions themselves. First we compute an update value ξ_e for every edge $e \in E(G)$ in the following way.

$$\xi_e \leftarrow \kappa_{ib}\delta({T_k}^{ib}, e) + \kappa_{rb}\delta({T_k}^{rb}, e) + \kappa_{gb}\delta({T_k}^{gb}, e) \quad (11)$$

where κ_{ib} is the influence weight of ${T_k}^{ib}$, κ_{rb} the influence weight of ${T_k}^{rb}$, κ_{gb} the influence weight of ${T_k}^{gb}$ and $\kappa_{ib} + \kappa_{rb} + \kappa_{gb} = 1.0$. The δ-function is defined as follows.

$$\delta(T_k, e_i) \quad = \quad \begin{cases} 1.0 & \text{if } e_i \in E(T_k) \\ 0.0 & \text{otherwise} \end{cases} \quad (12)$$

To the pheromone values τ_{e_i} of pheromone model $\mathsf{PH}_{\mathsf{1storder}}$ we then apply the following update rule.

$$\tau_{e_i} \quad \leftarrow \quad \tau_{e_i} + \rho \cdot (\xi_{e_i} - \tau_{e_i}) \quad (13)$$

To the pheromone values $\tau_{<e_i,e_j>}$ of the pheromone model $\mathsf{PH}_{\mathsf{2ndorder}}$ we apply basically the same pheromone update rule. We compute for every ordered pair of edges $< e_i, e_j >$ the value $\xi_{<e_i,e_j>}$ by using in equation (11) the following δ-function.

$$\delta(T_k, \tau_{<e_i,e_j>}) \quad = \quad \begin{cases} 1.0 & \text{if } e_i, e_j \in E(T_k) \\ 0.0 & \text{otherwise} \end{cases} \quad (14)$$

Then for updating the pheromone values we use the following rule.

$$\tau_{<e_i,e_j>} \leftarrow \tau_{<e_i,e_j>} + \rho \cdot \left(\xi_{<e_i,e_j>} - \tau_{<e_i,e_j>}\right) \quad (15)$$

Depending on the convergence factor cf the influence of every one of these 3 solutions on the pheromone update is determined. The convergence factor cf is a value providing an estimate about the state of convergence of the system. The convergence factor is computed in the following way.

$$cf \quad = \quad \frac{\sum_{e \in E(G)} \delta({T_k}^{ib}, e) \cdot (1.0 - \tau_e)}{k} \quad (16)$$

where we use the δ-function defined in equation (12). As by using the Hyper-Cube Framework for updating the pheromone values, the pheromone values can only assume values between 0.0 and 1.0 (see [6]) it obviously holds that cf also only can assume values between 0.0 and 1.0. It is also clear that if cf is close to 0.0 the system is in a state where the probability to produce solution ${T_k}^{ib}$ is close to 1 and therefore the probability to produce a solution different to ${T_k}^{ib}$ is close to 0. This is what we informally call the state of convergence for our system.

From experience gathered with the algorithm we chose the schedule of settings for values ρ,κ_{ib}, κ_{rb} and κ_{gb} as shown in Tab. 2. In the following we give an interpretation of the choice of parameters shown in Tab. 2. At the beginning of the search process the evaporation rate (which is in the Hyper-Cube Framework more appropriately called learning rate) is set to the value 0.15, because at the beginning of the search there is no need to be very careful. The algorithm should rather drift around in the search space to get a kind of global perspective. Also the influence of the best solution found in an iteration is quite high, which also supports the algorithm drifting through the search space. Once the algorithm starts converging (cf falls below 0.3) we decrease the learning rate (in order to perform a more careful search) and increase the influence of the best solution found since the restart of the algorithm. Once the algorithm is near to the state of convergence only the restart best solution is used to update the pheromone values and we decrease the learning rate even more in the hope to find a better solution near the restart best solution. Before the algorithm is completely converged we use the best solution found since the start of the algorithm to update the pheromone values. This action basically results in a shift of the probability distribution given by the pheromone values toward the best solution found. The reason behind that is the hope to find a better solution in-between two good solutions which are the restart best and the overall best solution in this case. This idea is very similar to ideas we can find in Path Relinking [22] for example.

Update($T_k{}^{ib}$,$T_k{}^{rb}$,$T_k{}^{gb}$): In this procedure we replace the old solution ${T_k}^{rb}$ with ${T_k}^{ib}$ if $w({T_k}^{ib}) < w({T_k}^{rb})$. We do the same for ${T_k}^{gb}$.

ComputeConvergenceFactor(τ, ${T_k}^{ib}$): The convergence factor cf is re-computed in every iteration according to equation (16).

ResetPheromoneValues(τ): In this procedure we reset all pheromone values τ_e to the start value 0.5.

4 Test results

We chose three different problem instances for a preliminary testing of our algorithms. Two of them are complete grid graphs[5] with 10 rows and 10 columns which sums up to 100 nodes and 180 edges. These graphs are called 10x10_1.gg and 10x10_2.gg in the following. The weights of the nodes were randomly generated using a uniform distribution on the integers between 1 and 100. There are two motivations for choosing grid graphs for testing our algorithms. Problems in practice are often modeled as grid graphs (e.g., the oil-field leasing problem in [23]). Also, it was observed in earlier publications (see [3]) that the problem is considerably harder to solve in grid graphs compared to unstructured graphs. Additionally we chose one of the

[5] No edges or nodes are missing in the grid.

Table 2: The schedule used for values ρ,κ_{ib}, κ_{rb} and κ_{gb} depending on the value of the convergence factor cf.

	$cf > 0.3$	$cf \leq 0.3$, $cf > 0.05$	$cf \leq 0.05$, $cf > 0.025$	$cf \leq 0.025$
ρ	0.15	0.1	0.05	0.1
κ_{ib}	2/3	1/3	0	0
κ_{rb}	1/3	2/3	1	0
κ_{gb}	0	0	0	1

graphs with 400 nodes and 800 edges used by Jornsten and Lokketangen in [24] to test their algorithm. This graph is called g400-4-01.dat. We applied the following four different algorithms to these three graphs:

- Alg. 1 using pheromone model $\mathsf{PH}_{\mathsf{1storder}}$, further on called ACO1.
- Alg. 1 using pheromone model $\mathsf{PH}_{\mathsf{2ndorder}}$, further on called ACO2.
- Alg. 1 using pheromone model $\mathsf{PH}_{\mathsf{1storder}}$, without updating the pheromone values. This is resulting basically in a multi–start local search, further on called MSLS.
- A heuristic based on greedy strategies developed in [16], which is called KCP.

The results are shown in Tables 3–5 (best results in bold). We started the algorithms for a range of cardinalities between 2 and $|V(G)|$. On each graph, each algorithm was applied 20 times for each cardinality. The results are reported for every algorithm (except for KCP) on every problem instance in four columns. The first column (titled "Obj.") contains the average of the best found solutions out of 20 runs. The second column (titled $\sqrt{\sigma}$) contains the standard deviation of these best found solutions. The third column (titled "time") contains the average of the times (in seconds) when the best solution of a run was found. Finally the fourth column (titled $\sqrt{\sigma}$) contains the standard deviation of these times. The stopping criterion for all the algorithms (except KCP) was a maximum amount of running time. We allowed the same amount of running time to all the algorithms. This amount of running time is dependent on the cardinality, and given in seconds by $1 + \frac{k \cdot |V(G)|}{100}$.
From the results in Tables 3–5 we can draw several conclusions. ACO1 is among the tested algorithms clearly the best one. Except for very high cardinalities – where the heuristic KCP is likely to produce the optimal solution – it clearly beats the other algorithms in average quality, in standard deviation of the quality, and in average time the best solution was found. This superiority is especially obvious for cardinalities in the middle of the cardinality range where the problem is harder to solve than at the beginning or the end of the cardinality range. The difference between ACO1 and MSLS points out, that the usage of pheromone values for the k-cardinality tree problem seems very fruitful. At first sight it seems surprising that ACO2 doesn't reach the quality of ACO1, because ACO2 is taking into account first order dependencies between decision variables compared to no dependencies in ACO1. However, if we consider the quadratic increase in complexity of the algorithm[6], the outcome of the experimental results become understandable. Due to the considerably increased complexity, ACO2 needs much more time to find good solutions. This is getting more obvious with growing graph size. Therefore the "use of more information" seems not to be very promising in ACO algorithms for the k-cardinality tree problem.

5 Conclusions and outlook to the future

In this work we presented an ACO algorithm for the edge-weighted k-cardinality tree problem. We presented two different pheromone models, the first of them not taking into account any dependencies between decision variables, the second one taking into account first order dependencies between decision variables. It turned out, that for the k-cardinality tree problem it doesn't seem beneficial to take into account dependencies between decision variables, because the increased complexity slows the algorithm considerably down. In the future we plan to improve the efficiency of our algorithm in order to compare it to state-of-the-art metaheuristics for the k-cardinality tree problem. We also plan to investigate the usefulness of diversification schemes for our algorithm.

Acknowledgments: This work was supported by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential program of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

[1] M.J. Blesa, P. Moscato, and F. Xhafa. A Memetic Algorithm for the Minimum Weighted k-Cardinality

[6] The number of pheromone values is quadratic in the number of edges of the graph.

Table 3: Results for grid graph 10x10_1.gg (100 nodes, 180 edges)

	ACO1				ACO2				MSLS				KCP
k	Obj.	$\sqrt{\sigma}$	time	$\sqrt{\sigma}$	Obj.	$\sqrt{\sigma}$	time	$\sqrt{\sigma}$	Obj.	$\sqrt{\sigma}$	time	$\sqrt{\sigma}$	Obj.
2	**5**	0	0.008	0.007	**5**	0	0.065	0.106	**5**	0	0.002	0.005	**5**
5	**52**	0	0.020	0.009	**52**	0	0.186	0.251	**52**	0	0.011	0.017	**52**
10	**161**	0	0.042	0.038	**161**	0	0.175	0.163	**161**	0	0.017	0.022	**161**
15	**254**	0	0.23	0.140	**254**	0	0.998	0.579	**254**	0	0.414	0.454	258
20	**368**	0	0.525	0.363	**368**	0	3.101	3.342	**368**	0	2.999	2.171	370
25	**465**	0	4.662	3.896	469.05	6.378	8.457	6.607	477	6.950	12.79	7.294	505
30	**579**	0	7.296	5.378	582.75	3.809	15.023	8.814	597.95	6.747	15.725	8.521	627
35	**689.45**	0.944	15.094	10.585	691.7	2.002	24.568	8.196	721.65	9.3148	13.049	9.478	752
40	**804.55**	1.394	17.435	12.332	806.1	1.618	19.529	11.490	853.75	13.396	17.758	9.352	878
45	**912**	3.641	24.067	14.795	918.55	3.605	24.704	10.933	973.3	14.179	24.387	14.124	988
50	**1023.45**	1.145	16.132	13.863	1027.25	4.865	29.549	12.344	1103.75	22.318	25.173	14.431	1106
55	**1139.3**	3.812	22.04	15.270	1144.75	6.248	34.9	8.576	1253	17.474	29.934	15.324	1280
60	**1257.15**	0.366	35.317	16.970	1263.7	4.932	37.319	9.840	1396.55	22.497	32.940	17.351	1404
65	**1400.1**	0.307	32.226	15.456	1406.2	4.741	39.4555	9.147	1556.6	24.489	25.657	17.176	1565
70	**1538.25**	0.550	29.426	13.584	1542.8	4.396	52.42	8.685	1709.85	37.307	29.738	17.042	1696
75	**1689.5**	1.192	42.561	21.223	1694.75	8.801	59.657	9.304	1872	31.522	35.631	25.631	1840
80	**1867.65**	1.755	44.51	17.818	1877.35	7.922	68.369	8.701	2065.1	21.875	33.666	25.342	2005
85	**2066.9**	2.381	48.310	20.819	2081.1	7.362	71.026	9.454	2263.4	27.933	42.992	24.616	2128
90	**2299.05**	1.431	51.622	23.100	2311.05	6.082	80.548	10.948	2530.2	29.881	48.499	20.828	2301
95	**2594.5**	1.877	64.463	26.011	2610.25	8.058	81.702	8.712	2872.9	22.083	49.320	27.162	2599
98	2795.15	2.906	71.549	23.218	2812.6	11.408	90.541	6.419	3124.6	35.858	68.147	24.817	**2791**

Table 4: Results for grid graph 10x10_2.gg (100 nodes, 180 edges)

	ACO1				ACO2				MSLS				KCP
k	Obj.	$\sqrt{\sigma}$	time	$\sqrt{\sigma}$	Obj.	$\sqrt{\sigma}$	time	$\sqrt{\sigma}$	Obj.	$\sqrt{\sigma}$	time	$\sqrt{\sigma}$	Obj.
2	**4**	0	0.012	0.010	**4**	0	0.065	0.068	**4**	0	0.003	0.005	**4**
5	**24**	0	0.013	0.006	**24**	0	0.054	0.045	**24**	0	0.004	0.007	**24**
10	**65**	0	0.027	0.015	**65**	0	0.138	0.147	**65**	0	0.018	0.017	**65**
15	**162**	0	0.116	0.051	**162**	0	0.561	0.229	**162**	0	0.230	0.139	**162**
20	**255**	0	0.202	0.163	**255**	0	1.025	1.558	**255**	0	0.197	0.256	257
25	**309**	0	0.453	0.180	**309**	0	1.675	0.831	**309**	0	2.505	1.835	329
30	**422**	0	1.034	0.348	**422**	0	3.447	1.201	425.3	3.357	14.779	9.166	425
35	**520**	0	3.331	3.425	520.1	0.307	7.463	6.599	535.1	9.425	17.029	10.449	570
40	**602**	0	5.14	5.285	602.75	1.831	10.09	2.613	663.9	19.185	19.774	10.753	658
45	**693.25**	1.118	15.246	10.150	695.3	2.617	25.856	12.139	759.4	19.146	23.635	14.133	780
50	**792**	0	3.916	0.693	**792**	0	13.740	2.510	873.35	30.475	24.791	13.466	857
55	**905**	0	11.800	6.686	906.6	1.391	21.907	6.549	1012.3	23.299	26.930	18.319	973
60	**1019**	0	6.544	1.136	**1019**	0	24.003	5.312	1152.2	16.577	29.297	19.926	1080
65	**1148.8**	0.410	18.610	17.345	1153.45	6.581	47.362	10.014	1279.55	20.309	31.098	19.773	1189
70	**1277**	0	16.936	10.631	1285.8	8.983	55.328	9.765	1439.35	24.381	38.059	16.969	1344
75	**1432.6**	1.142	28.197	16.183	1435.05	4.370	62.882	7.456	1599.4	29.077	34.194	23.764	1465
80	**1607.7**	3.130	29.585	20.031	1615.2	8.489	60.664	12.852	1794.1	25.708	39.664	20.211	1634
85	**1853**	0	30.097	18.776	1860.8	10.211	69.03	14.123	2065.9	30.371	45.102	26.022	1863
90	**2118**	2.051	35.263	21.807	2123.4	6.029	73.458	11.994	2375.05	31.091	36.224	22.214	2132
95	2429.7	1.592	49.771	21.225	2439.55	7.279	75.830	10.991	2701.5	31.103	50.298	33.037	**2429**
98	2631.35	0.988	58.222	30.827	2640.65	9.027	81.049	14.101	2934.05	30.5829	50.318	27.746	**2631**

Table 5: Results for graph g400-4-01.dat from Jornsten and Lokketangen (400 nodes, 800 edges)

	ACO1				ACO2				MSLS				KCP
k	Obj.	$\sqrt{\sigma}$	time	$\sqrt{\sigma}$	Obj.	$\sqrt{\sigma}$	time	$\sqrt{\sigma}$	Obj.	$\sqrt{\sigma}$	time	$\sqrt{\sigma}$	Obj.
2	**8**	0	0.132	0.116	8.85	2.621	2.354	1.959	**8**	0	0.088	0.091	**8**
20	**253**	0	5.388	5.916	254.1	2.291	36.806	17.368	**253**	0	16.270	11.070	272
40	**563**	0	35.785	20.175	566.95	3.691	100.601	30.748	608.45	16.519	94.069	45.563	604
60	**919.25**	0.716	102.26	59.518	923.1	3.291	207.44	20.860	1013.2	27.532	96.098	74.733	1022
80	**1306.75**	2.048	192.245	73.594	1334.15	10.095	295.5	32.245	1483.35	24.615	135.729	98.816	1408
100	**1731.1**	5.280	271.189	86.696	1817.75	12.961	367.046	36.947	1971.15	33.750	186.865	130.132	1926
120	**2180**	10.031	338.813	71.064	2328.95	26.727	452.901	41.416	2512.3	35.797	231.954	159.193	2350
140	**2641.05**	17.425	408.078	149.159	2875.35	34.392	523.885	55.076	3048.25	43.770	326.553	138.83	2822
160	**3129.8**	14.028	461.798	132.681	3477.55	62.543	617.647	80.852	3653.8	56.113	313.43	195.834	3220
180	**3641.15**	16.480	445.803	50.486	4139.45	50.625	687.47	104.207	4254.95	43.192	381.348	205.387	3758
200	**4180.6**	21.685	573.88	65.199	4756	76.595	824.476	110.62	4874.1	45.750	416.265	212.639	4262
220	**4753.95**	16.452	702.897	69.385	5465.75	62.905	837.577	171.348	5546	57.945	493.855	255.638	4806
240	**5342.85**	18.765	861.051	84.423	6176.7	83.805	1009.41	105.39	6227	78.016	673.384	233.757	5429
260	**5960.35**	19.505	1005.04	38.325	6938.05	62.001	945.168	216.461	6943.55	70.770	529.437	275.012	6040
280	**6618.45**	28.010	1083.75	29.585	7669.05	74.339	1139.98	199.618	7683.1	59.158	547.638	309.312	6715
300	**7312.75**	16.476	1161.57	38.844	8493.9	102.496	1044.78	252.145	8418.2	92.126	659.805	333.841	7386
320	**8029.15**	20.625	1232.46	48.928	9298.25	56.904	1283.54	257.799	9275.2	56.083	768.435	361.237	8063
340	8808.1	19.325	1318.16	41.868	10168	90.851	1359.32	271.292	10132.4	64.570	678.346	405.055	**8805**
360	9641.65	23.351	1403.69	33.868	11126.6	114.205	1497.57	293.392	11125.3	75.764	652.205	373.956	**9554**
380	10585.5	20.914	1466.83	36.829	12336.1	95.999	1555.28	427.12	12256.3	74.524	746.958	405.787	**10451**
398	11702.6	28.593	1441.75	82.004	14327	113.37	1461.42	385.02	14216.5	74.663	729.97	448.678	**11433**

Tree Subgraph Problem. In *Proceedings of the Metaheuristics International Conference MIC'2001*, volume 1, pages 85–90, Porto, Portugal, July 2001.

[2] M.J. Blesa and F. Xhafa. A C++ Implementation of Tabu Search for k-Cardinality Tree Problem based on Generic Programming and Component Reuse. In c/o tranSIT GmbH, editor, *Net.ObjectDsays 2000 Tagungsband*, pages 648–652, Erfurt, Germany, 2000. Net.ObjectDays-Forum.

[3] C. Blum. Optimality criteria and local search methods for node-weighted k-cardinality tree and subgraph problems. Master's thesis, Department of Mathematics, University of Kaiserslautern, Germany, 1998. In German.

[4] C. Blum and M. Ehrgott. Local search algorithms for the k-cardinality tree problem. Technical Report TR/IRIDIA/2001-12, IRIDIA, Université Libre de Bruxelles, Belgium, 2001. submitted to Discrete Applied Mathematics.

[5] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. Technical Report TR/IRIDIA/2001-13, IRIDIA, Université Libre de Bruxelles, Belgium, 2001.

[6] C. Blum, A. Roli, and M. Dorigo. HC-ACO: The hyper-cube framework for ant colony optimization. In *Proceedings of Metaheuristics International Conference MIC'2001*, pages 399–403, Porto, Portugal, July 2001.

[7] C. Blum and M. Sampels. Ant Colony Optimization for FOP Shop scheduling: a case study on different pheromone representations. Accepted for presentation at the Congress on Evolutionary Computation, CEC 2002, May 2002.

[8] F. Catanas. Heuristics for the p-minimal spanning tree problem. Technical report, Centre for Quantitative Finance, Imperial College, London, UK, 1997.

[9] S.Y. Cheung and A. Kumar. Efficient quorumcast routing algorithms. In *Proceedings of INFOCOM'94*, Los Alamitos, USA, 1994. IEEE Society Press.

[10] M. Dorigo. *Optimization, Learning and Natural Algorithms* (in Italian). PhD thesis, DEI, Politecnico di Milano,Italy, 1992. pp. 140.

[11] M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.

[12] M. Dorigo and L. M. Gambardella. Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[13] M. Dorigo, V. Maniezzo, and A. Colorni. Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics - Part B*, 26(1):29–41, 1996.

[14] T. Dudas, B. Klinz, and G.J. Woeginger. The computational complexity of the k-minimum spanning tree problem in graded matrices. *Computers and Mathematics with Applications*, 36:61–67, 1998.

[15] M. Ehrgott and J. Freitag. K_TREE / K_SUBGRAPH: A program package for minimal weighted k-cardinality-trees and -subgraphs. *European Journal of Operational Research*, 1(93):214–225, 1996. code available at http://www.mathematik.uni-kl.de/~wwwwi/WWWWI/ORSEP/contents.html.

[16] M. Ehrgott, J. Freitag, H.W. Hamacher, and F. Maffioli. Heuristics for the k-cardinality tree and subgraph problem. *Asia Pacific Journal of Operational Research*, 14(1):87–114, 1997.

[17] M. Fischetti, H.W. Hamacher, K. Joernsten, and F. Maffioli. Weighted k-cardinality trees: Complexity and polyhedral structure. *Networks*, 24:11–21, 1994.

[18] L.R. Foulds and H.W. Hamacher. A new integer programming approach to (restricted) facilities layout problems allowing flexible facility shapes. Technical Report 1992-3, University of Waikato, Department of Management Science, 1992.

[19] J. Freitag. Minimal k-cardinality trees. Master's thesis, Department of Mathematics, University of Kaiserslautern, Germany, 1993. in german.

[20] N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, pages 302–309, Los Alamitos, USA, 1996. IEEE Society Press.

[21] N. Garg and D. Hochbaum. An $O(\log k)$ approximation algorithm for the k minimum spanning tree problem in the plane. *Algorithmica*, 18:111–121, 1997.

[22] F. Glover, M. Laguna, and R. Marti. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 29(3):653–684, 2000.

[23] H.W. Hamacher and K. Joernsten. Optimal relinquishment according to the Norwegian petrol law: A combinatorial optimization approach. Technical Report No. 7/93, Norwegian School of Economics and Business Administration, Bergen, Norway, 1993.

[24] K. Joernsten and A. Løkketangen. Tabu search for weighted k-cardinality trees. *Asia Pacific Journal of Operational Research*, 14(2):9–26, 1997.

[25] F. Maffioli. Finding a best subtree of a tree. Technical Report 91.041, Politecnico di Milano, Dipartimento di Elettronica, Italy, 1991.

[26] M.V. Marathe, R. Ravi, S.S. Ravi, D.J. Rosenkrantz, and R. Sundaram. Spanning trees – short or small. *SIAM Journal on Discrete Mathematics*, 9(2):178–200, 1996.

[27] N. Mladenovic. Variable neighborhood search for the k-cardinality tree problem. In *Proceedings of the Metaheuristics International Conference MIC'2001*, 2001.

[28] A. Roli, C. Blum, and M. Dorigo. ACO for maximal constraint satisfaction problems. In *Proceedings of Metaheuristics International Conference MIC'2001*, pages 187–191, Porto, Portugal, July 2001.

[29] T. Stützle and H. H. Hoos. $\mathcal{MAX}$-$\mathcal{MIN}$ ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000.

On a Particularity in Model-Based Search

Christian Blum[*], **Michael Sampels**
IRIDIA
Université Libre de Bruxelles
Brussels, Belgium
e-mail: {cblum,msampels}@ulb.ac.be
tel: (32) 2 650 3168, fax: (32) 2 650 2715

Mark Zlochin
Department of Computer Science
Technion – Israel Institute of Technology,
Haifa, Israel
e-mail: zmark@cs.technion.ac.il
tel: (972) 4 829 4948

Abstract

Giving positive feedback to good solutions is a common base technique in model-based search algorithms, such as Ant Colony Optimization, Estimation of Distribution Algorithms, or Neural Networks. In particular, the reinforcement of components of good solutions by positive feedback is known as a successful technique in tackling hard combinatorial optimization problems. We show by a simple model-based search algorithm for the node-weighted k-cardinality tree problem that this strategy doesn't guarantee steadily increasing performance of the algorithm in general. It is rather possible that for some "problem"-"probabilistic model" combinations the average performance of the system is decreasing and even the average probability of sampling good solutions is decreasing over time. The result is proven analytically and the consequences are studied in some empirical case studies.

1 Introduction

Model-based search (MBS) [10] algorithms are increasingly popular methods for solving combinatorial optimization problems. In MBS algorithms, such as Ant Colony Optimization (ACO) [2] or Estimation of Distribution Algorithms (EDAs)[1] [7, 8], candidate solutions are generated using a parametrized probabilistic model that is updated using the previously seen solutions in such a way that the search will concentrate in the regions of the search space containing high quality solutions. In particular, reinforcment of solution components depending on the solution quality is an important factor in the development of heuristics to tackle hard combinatorial optimization problems. It is assumed that good solutions don't occur sporadically, but consist of good solution components. To learn which components contribute to good solutions can help to assemble them to better solutions. In general, a model-based search approach attempts to solve an optimization problem by repeating the following two steps:

- Candidate solutions are constructed using some parametrized probabilistic model, that is, a parametrized probability distribution over the solution space.
- The candidate solutions are used to modify the model in a way that is deemed to bias future sampling toward low cost solutions.

Often it is implicitly assumed that the average performace of model-based algorithms is increasing over time. However, during empirical investigations of an Ant Colony Optimization Algorithm for the Group Shop scheduling problem[2] we observed that for some of the probabiltistic models chosen the performance of the system was decreasing over time. This triggered us to explore "problem"-"probabilistic model" combinations where the expected performance of a model-based search algorithm decreases over time. As a test case we chose the node-weighted k-cardinality tree problem, an NP-hard combinatorial optimization problem.

The paper is organized as follows. In Sec. 2 we briefly present the node-weighted k-cardinality tree problem. In Sec. 3 we outline a simple model-based search algorithm for the k-cardinality tree problem. Section 4 contains the analytical analysis of the system for a

[*] Corresponding author

[1] EDAs are covering several algorithms emerging from the field of Evolutionary Computation.

[2] Group Shop scheduling is a general formulation of scheduling problems covering Job Shop scheduling and Open Shop scheduling

small problem instance. Section 5 deals with empirical results and Sec. 6 finally offers conclusions and an outlook to the future.

2 The node-weighted k-cardinality tree problem

The k-cardinality tree problem is a combinatorial optimization problem that generalizes the well–known minimum weight spanning tree problem. It consists in finding in a node- or edge-weighted graph a subtree with exactly k edges, such that the sum of the weights is minimal. Due to various applications, such as oil-field leasing [6], facility layout [4], quorum-cast routing [1] and telecommunications [5], it has gained considerable interest in recent years. In this paper we deal with the k-cardinality tree problem in node-weighted graphs. The problem can be formally defined as follows. Let $G = (V, E)$ be a graph (where $|V| = n$ and $|E| = m$) with a weight function $w : V \to I\!N$ on the nodes. We denote the set of all k-cardinality trees in G by $\mathcal{T}_k$. Then the node-weighted problem (G, w, k) is to find a k-cardinality tree $T_k \in \mathcal{T}_k$ that minimizes

$$w(T_k) = \sum_{v \in V(T_k)} w(v). \qquad (1)$$

The general problem is NP-hard, and in [9] NP-completeness results have been obtained for grid and split graphs.

3 A simple MBS algorithm for the k-cardinality tree problem

In this section we briefly outline a simple model-based search algorithm based on positive feedback for the k-cardinality tree problem. It is constructed in a straight-forward manner and it is representative for the class of model-based algorithms. The pseudo-code for this algorithm is shown in Alg. 1. In Alg. 1, T_k^i denotes the ith solution constructed in the current iteration, $n_s \geq 1$ is the total number of solutions constructed in every iteration, and $\tau = \{\tau_{v_1}, ..., \tau_{v_n}\}$ is a set of model parameters. After initialization of the model parameters, in every step of the algorithm n_s solutions are constructed using the current values of the model parameters. These solutions are then used to update the model parameters which are defined as follows: To every node $v \in V(G)$ we have associated a model parameter τ_v. The components of Alg. 1 are to be explained in more detail in the following.

Algorithm 1 A model-based algorithm for the node-weighted k-cardinality tree problem

input: A problem instance (G, w, k)
InitializeModelParameters(τ)
while termination conditions not met **do**
  for $i = 1, ..., n_s$ **do**
    $T_k^i \leftarrow$ ConstructSolution(τ)
  end for
  ApplyModelParameterUpdate(τ,T_k^1,...,$T_k^{n_s}$)
end while
output: A k-cardinality tree T_k^{best}

InitializeModelParameters(τ): At the beginning of the algorithm, the model parameters τ_v are all initialized to the same small numerical value $c > 0$.

ConstructSolution(τ): In order to construct solutions to the problem we have to formalize how to use the model parameter values to construct solutions[3]. For constructing a solution, $k + 1$ construction steps are done as shown in Alg. 2. In every step of the construction phase, we can only add a node v to the partial k-cardinality tree T_t ($t \in \{1, ..., k-1\}$) if there is an edge $e \in E(G)$ that connects one of the nodes in T_t with v. The probabilities $p(v_i \mid T_t)$ for all $v_i \in J_t$ are then defined in the following way.

Algorithm 2 Solution construction

$V(T_0) \leftarrow \emptyset$ and $E(T_0) \leftarrow \emptyset$
$J_0 \leftarrow V(G)$
Choose $v^\star \in J_0$ with probability $p(v^\star \mid T_0)$ {See eqn. (2)}
$V(T_0) \leftarrow V(T_0) \cup v^\star$
for $t = 1$ to k **do**
  $J_t \leftarrow \{v \in V(G) \setminus V(T_{t-1}) \mid \exists\, e[v_r, v] \in E(G)$ with $v_r \in V(T_{t-1})\}$
  Choose $v^\star \in J_t$ with probability $p(v^\star \mid T_t)$
  Find an edge e connecting $v^\star$ with T_t
  $E(T_t) \leftarrow E(T_{t-1}) \cup \{e\}$
  $V(T_t) \leftarrow V(T_{t-1}) \cup \{v^\star\}$
end for

$$p(v_i \mid T_t) = \begin{cases} \frac{\tau_{v_i}}{\sum_{v \in J_t} \tau_v} & \text{if} \quad v_i \in J_t \\ 0 & \text{otherwise} \end{cases} \qquad (2)$$

where J_t is the set of nodes allowed to be added next to the partial k-cardinality tree T_t (see Alg. 2).

ApplyModelParameterUpdate(τ,T_k^1,...,$T_k^{n_s}$): Once all solutions of an iteration are constructed, a rule updating the model parameters is applied. For the set of model parameters $\{\tau_{v_1}, ..., \tau_{v_n}\}$ this update rule is defined as:

$$\tau_{v_i} \leftarrow (1 - \rho) \cdot \tau_{v_i} + \tfrac{1}{n_s} \cdot \textstyle\sum_{j=1}^{n_s} \Delta\tau_{v_i}^j \qquad (3)$$

[3] We have to define a method of using the model to sample the search space.

where

$$\Delta\tau_{v_i}^{j} = \begin{cases} \frac{1}{w(T_k^j)} & \text{if } v_i \in T^j \\ 0 & \text{otherwise .} \end{cases} \quad (4)$$

In (4), T^j is the jth solution produced in the current iteration, $w(T^j)$ is the quality of solution T^j and $0 < \rho < 1$ is a parameter supporting the intensification of the search (a similar parameter in ACO algorithms is called evaporation rate). This update rule is similar to update rules used in Ant Colony Optimization and some other population-based methods from the field of Evolutionary Computation. It should lead to an increase of model parameter values associated with solution components which have been found in better quality solutions compared with other solution components, hence increasing the expected performanc of the algorithm over time. In the following section we will prove that this expectation is not always met in practice.

4 A counterexample: Decreasing average performance of a MBS algorithm over time

In this section we choose a small problem instance of the k-cardinality tree problem and we show that the expected performance of Alg. 1 is decreasing. The problem instance under consideration is shown in Fig. 1. It shows an undirected graph $G = (V, E)$ formally defined as follows.

$$V(G) = \{v_1, v_2, v_3, v_4\} \quad (5)$$
$$E(G) = \{e_{v_1,v_2}, e_{v_2,v_3}, e_{v_3,v_4}\} \quad (6)$$
$$w(v_i) = w_i > 0, \text{ for } i = 1, ..., 4 \quad (7)$$

For the weights we choose

$$w_2, w_3 > w_1, w_4, \quad (8)$$

and for the sake of simplicity we choose

$$w_1 = w_4 \text{and } w_2 = w_3 \ . \quad (9)$$

In graph G we want to solve the 1-cardinality tree problem with the model-based search algorithm outlined in the last section. In G we can find 3 different 1-cardinality trees:

$$\begin{aligned} T^a \text{ with } & V(T^a) = \{v_1, v_2\}, E(T^a) = \{e_{v_1,v_2}\} \\ \Rightarrow\ & w(T^a) = w_1 + w_2 \\ T^b \text{ with } & V(T^b) = \{v_2, v_3\}, E(T^b) = \{e_{v_2,v_3}\} \\ \Rightarrow\ & w(T^b) = w_2 + w_3 \\ T^c \text{ with } & V(T^c) = \{v_3, v_4\}, E(T^c) = \{e_{v_3,v_4}\} \\ \Rightarrow\ & w(T^c) = w_3 + w_4 \end{aligned}$$

Because of (8) and (9) it holds that

$$w(T^a) = w(T^c) < w(T^b). \quad (10)$$

Therefore T^a and T^c are both optimal solutions of this problem instance and T^b is the worst solution. With model parameter values $\tau_{v_1}(t), ..., \tau_{v_4}(t)$ at discrete time steps $t = 0, 1, ...$, the probabilities $p_a(t)$, $p_b(t)$ and $p_c(t)$ to construct solutions T^a, T^b and T^c are the following:

$$\begin{aligned} p_a(t) &= \frac{\tau_{v_1}(t)}{\sum_{i=1}^{4}\tau_{v_i}(t)} \\ &+ \frac{\tau_{v_2}(t)}{\sum_{i=1}^{4}\tau_{v_i}(t)} \cdot \frac{\tau_{v_1}(t)}{\tau_{v_1}(t)+\tau_{v_3}(t)} \quad (11) \end{aligned}$$

$$\begin{aligned} p_b(t) &= \frac{\tau_{v_2}(t)}{\sum_{i=1}^{4}\tau_{v_i}(t)} \cdot \frac{\tau_{v_3}(t)}{\tau_{v_1}(t)+\tau_{v_3}(t)} \\ &+ \frac{\tau_{v_3}(t)}{\sum_{i=1}^{4}\tau_{v_i}(t)} \cdot \frac{\tau_{v_2}(t)}{\tau_{v_2}(t)+\tau_{v_4}(t)} \quad (12) \end{aligned}$$

$$\begin{aligned} p_c(t) &= \frac{\tau_{v_4}(t)}{\sum_{i=1}^{4}\tau_{v_i}(t)} \\ &+ \frac{\tau_{v_3}(t)}{\sum_{i=1}^{4}\tau_{v_i}(t)} \cdot \frac{\tau_{v_4}(t)}{\tau_{v_4}(t)+\tau_{v_2}(t)} \quad (13) \end{aligned}$$

We use the notation p_{v_i} for the probability of node v_i to be found in a solution constructed. These probabilities are obviously the following ones.

$$p_{v_1}(t) = p_a(t) \quad (14)$$
$$p_{v_2}(t) = p_a(t) + p_b(t) \quad (15)$$
$$p_{v_3}(t) = p_b(t) + p_c(t) \quad (16)$$
$$p_{v_4}(t) = p_c(t) \quad (17)$$

Let us now examine the evolution of the model parameter values over time.

Evolution of τ_{v_1} over time: After every construction step there are two possibilities. Either v_1 is a part of the constructed solution, or it is not. Then the expected value of τ_{v_1} at time $t+1$ is the following.

$$E(\tau_{v_1}(t+1)) = \left((1-\rho)\cdot\tau_{v_1}(t) + \tfrac{1}{w_1+w_2}\right)\cdot p_{v_1}(t)$$

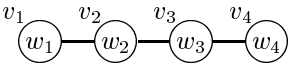


Figure 1: The problem instance consists of four nodes v_1, v_2, v_3 and v_4, connected by three edges. The node weights of the nodes v_i are indicated by w_i.

$$
\begin{aligned}
&+ \quad (1-\rho)\cdot\tau_{v_1}(t)\cdot(1-p_{v_1}(t)) \\
&= \quad \tau_{v_1}(t)p_{v_1}(t) - \rho\tau_{v_1}(t)p_{v_1}(t) \\
&+ \quad \tfrac{1}{w_1+w_2}p_{v_1}(t) + \tau_{v_1}(t) - \tau_{v_1}(t)p_{v_1}(t) \\
&- \quad \rho\tau_{v_1}(t) + \rho\tau_{v_1}(t)p_{v_1}(t) \\
&= \quad \tau_{v_1}(t) + \left(\tfrac{1}{w_1+w_2}\cdot p_{v_1}(t) - \rho\tau_{v_1}(t)\right)
\end{aligned}
$$

As $p_{v_1}(t) = p_a(t)$ we get

$$
E(\tau_{v_1}(t+1)) \;=\; \tau_{v_1}(t) + \left(\tfrac{1}{w_1+w_2}\cdot p_a(t) - \rho\tau_{v_1}(t)\right) \tag{18}
$$

Evolution of τ_{v_2} over time: After every construction step there is – as above for τ_{v_1} – the possibility that v_2 is a part of the solution constructed. But there are also two different solutions v_2 can be part of. In the following p_{a+b} will stand for $p_a + p_b$, w_{1+2} will stand for $w_1 + w_2$, and w_{2+3} will stand for $w_2 + w_3$.

$$
\begin{aligned}
&E(\tau_{v_2}(t+1)) \;= \\
&\left((1-\rho)\cdot\tau_{v_2}(t) + \tfrac{p_a(t)}{p_{a+b}(t)}\tfrac{1}{w_{1+2}} + \tfrac{p_b(t)}{p_{a+b}(t)}\tfrac{1}{w_{2+3}}\right)\cdot p_{v_2}(t) \\
&\qquad + (1-\rho)\cdot\tau_{v_2}(t)\cdot(1-p_{v_2}(t))
\end{aligned}
$$

As $p_{v_2} = p_{a+b}$ we get

$$
\begin{aligned}
E(\tau_{v_2}(t+1)) \quad &= \quad (1-\rho)\cdot\tau_{v_2}(t)\cdot p_{a+b}(t) + \tfrac{p_a(t)}{w_{1+2}} \\
&+ \quad \tfrac{p_b(t)}{w_{2+3}} + (1-\rho)\,\tau_{v_2}(t)\,(1-p_{a+b}(t)) \\
&= \quad \tau_{v_2}(t)p_{a+b}(t) - \rho\tau_{v_2}(t)p_{a+b}(t) \\
&+ \quad \tfrac{p_a(t)}{w_{1+2}} + \tfrac{p_b(t)}{w_{2+3}} + \tau_{v_2}(t) \\
&- \quad \tau_{v_2}(t)p_{a+b}(t) - \rho\tau_{v_2}(t) \\
&+ \quad \rho\tau_{v_2}(t)p_{a+b}(t) \;\;.
\end{aligned}
$$

Therefore we get

$$
E(\tau_{v_2}(t+1)) \;=\; \tau_{v_2}(t) + \left(\tfrac{p_a(t)}{w_{1+2}} + \tfrac{p_b(t)}{w_{2+3}} - \rho\tau_{v_2}(t)\right) \;\;. \tag{19}
$$

For the **evolution of τ_{v_3} and τ_{v_4}** the computations are the same as for τ_{v_2} and τ_{v_1} respectively. Consequently

$$
E(\tau_{v_3}(t+1)) \;=\; \tau_{v_3}(t) + \left(\tfrac{p_c(t)}{w_{3+4}} + \tfrac{p_b(t)}{w_{2+3}} - \rho\tau_{v_3}(t)\right) \tag{20}
$$

and

$$
E(\tau_{v_4}(t+1)) \;=\; \tau_{v_4}(t) + \left(\tfrac{1}{w_{3+4}}\cdot p_c(t) - \rho\tau_{v_4}(t)\right) \;\;. \tag{21}
$$

It is common practice in Genetic Algorithm research to analyse the bahaviour of the algorithm with infinite population size. Therefore, in the following we consider the limit case of $n_s \to \infty$. In this case the law of large number says that the actual value of $\tau_{v_i}(t)$ converges in probability to the expected value. Therefore we use $\tau_{v_i}(t)$ instead of $E(\tau_{v_i}(t))$ in the following. The algorithm at time $t = 0$ starts with $\tau_{v_1}(0) = \tau_{v_2}(0) = \tau_{v_3}(0) = \tau_{v_4}(0) = c > 0$. With (9), (11) and (13) it follows that $p_a(0) = p_c(0)$. With (18) and (21) it follows that $\tau_{v_1}(1) = \tau_{v_4}(1)$ and with (19) and (20) it follows that $\tau_{v_2}(1) = \tau_{v_3}(1)$. In turn this implies with (9) that $p_a(1) = p_c(1)$. By induction it follows that for $t \geq 0$

$$
\tau_{v_1}(t) \;=\; \tau_{v_4}(t) \quad \text{and} \quad \tau_{v_2}(t) = \tau_{v_3}(t) \tag{22}
$$

$$
p_a(t) \;=\; p_c(t) \;\;. \tag{23}
$$

This means that the evolution of $\tau_{v_1}(t)$ is equal to the evolution of $\tau_{v_4}(t)$, the evolution of $\tau_{v_2}(t)$ is equal to the evolution of $\tau_{v_3}(t)$ and the probability to produce tree T^a is equal to the probability to produce tree T^c. This allows us to simplify equations (11), (12) and (13) expressing everything in terms of $\tau_{v_1}(t)$ and $\tau_{v_2}(t)$. Substituting $\tau_{v_3}(t)$ by $\tau_{v_2}(t)$ and $\tau_{v_4}(t)$ by $\tau_{v_1}(t)$ in equations (11), (12) and (13) results in

$$
p_a(t) \;=\; p_c(t) \;=\; \tfrac{\tau_{v_1}(t)}{2(\tau_{v_1}(t)+\tau_{v_2}(t))}\cdot\left(1 + \tfrac{\tau_{v_2}(t)}{\tau_{v_1}(t)+\tau_{v_2}(t)}\right) \tag{24}
$$

$$
p_b(t) \;=\; \tfrac{\tau_{v_2}(t)}{\tau_{v_1}(t)+\tau_{v_2}(t)}\cdot\tfrac{\tau_{v_2}(t)}{\tau_{v_1}(t)+\tau_{v_2}(t)} \;\;. \tag{25}
$$

Theorem 1 *In the settings described above the following holds. The probability $p_b(t)$ to produce T^b (the worst 1-cardinality tree to be found in graph G as defined in equations (5)–(9)) is increasing from the first step of Alg. 1 onward as long as*

$$
\tfrac{1}{w_{2+3}}p_b(t)\cdot\tau_{v_1} > \tfrac{1}{w_{1+2}}\cdot p_a(t)\,(\tau_{v_2} - \tau_{v_1}) \;\;. \tag{26}
$$

Proof: In the following we will write $\tau_{v_1+v_2}(t)$ for $\tau_{v_1}(t) + \tau_{v_2}(t)$. Then

$$
\begin{aligned}
&p_b(t+1) > p_b(t) \\
&\Leftrightarrow \\
&\left(\frac{\tau_{v_2}(t)+\left(\frac{p_a(t)}{w_{1+2}}+\frac{p_b(t)}{w_{2+3}}-\rho\tau_{v_2}(t)\right)}{\tau_{v_1+v_2}(t)+\left(\frac{2p_a(t)}{w_{1+2}}+\frac{p_b(t)}{w_{2+3}}-\rho(\tau_{v_1+v_2}(t))\right)}\right)^2 \\
&> \\
&\left(\frac{\tau_{v_2}(t)}{\tau_{v_1+v_2}(t)}\right)^2 \\
&\Leftrightarrow \text{(taking the square root)} \\
&\left(\tau_{v_2}(t) + \left(\frac{p_a(t)}{w_{1+2}} + \frac{p_b(t)}{w_{2+3}} - \rho\tau_{v_2}(t)\right)\right)\cdot(\tau_{v_1+v_2}(t)) \\
&> \\
&\tau_{v_2}(t)\cdot\left(\tau_{v_1+v_2}(t) + \left(\frac{2p_a(t)}{w_{1+2}} + \frac{p_b(t)}{w_{2+3}} - \rho(\tau_{v_1+v_2}(t))\right)\right) \\
&\Leftrightarrow \\
&\tau_{v_2}(t)\tau_{v_1}(t) + (\tau_{v_2}(t))^2 + \frac{p_a(t)}{w_{1+2}}\tau_{v_1}(t) + \frac{p_a(t)}{w_{1+2}}\tau_{v_2}(t) \\
&+\frac{p_b(t)}{w_{2+3}}\tau_{v_1}(t) + \frac{p_b(t)}{w_{2+3}}\tau_{v_2}(t) - \rho\tau_{v_2}(t)\tau_{v_1}(t) - \rho\,(\tau_{v_2}(t))^2 \\
&> \\
&\tau_{v_2}(t)\tau_{v_1}(t) + (\tau_{v_2}(t))^2 + \frac{2p_a(t)}{w_{1+2}}\tau_{v_2}(t) + \frac{p_b(t)}{w_{2+3}}\tau_{v_2}(t) \\
&-\rho\tau_{v_2}(t)\tau_{v_1}(t) - \rho\,(\tau_{v_2}(t))^2 \\
&\Leftrightarrow \\
&\tfrac{1}{w_{2+3}}p_b(t)\cdot\tau_{v_1} > \tfrac{1}{w_{1+2}}\cdot p_a(t)\,(\tau_{v_2} - \tau_{v_1}) \;\;.
\end{aligned}
$$

As in the first iteration of Alg. 1 all model parameter values are equal and greater than 0, it holds that $p_b(1) > p_b(0)$. **qed**

Remark 1 *With (22) it is now also clear that in the case $p_b(t+1) > p_b(t)$ it holds that*

$$\begin{aligned} p_a(t+1) &< p_a(t), && (27)\\ p_c(t+1) &< p_c(t), && (28)\\ p_a(t+1) - p_a(t) &= -\tfrac{1}{2}\left(p_b(t+1) - p_b(t)\right), && (29)\\ p_c(t+1) - p_c(t) &= -\tfrac{1}{2}\left(p_b(t+1) - p_b(t)\right) && (30) \end{aligned}$$

If we now measure the performance $\mathcal{P}(t)$ of Alg. 1 as the expected average quality of a solution produced at any time $t \geq 0$ as

$$\mathcal{P}(t) = w_{1+2} \cdot p_a(t) + w_{2+3} \cdot p_b(t) + w_{3+4} \cdot p_c(t), \quad (31)$$

then it is clear that for $p_b(t+1) > p_b(t)$ the expected performance of the system is decreasing from time t to time $t+1$. Using Theorem 1 it follows that the expected performance of Alg. 1 is decreasing from $t = 0$ to an unknown time $t_{stop} > 0$ which may be finite or infinite. To summarize, we have shown that it is possible to find circumstances where the expected performance – as formalized in (31) – of Alg. 1 is decreasing. In the next section we will confirm the outcomes outlined in this section empirically.

5 Analytical curves and empirical confirmation

In this section we present analytical and empirical results for Alg. 1 on two different weight settings for the k-cardinality tree problem instance defined in equations (5)–(9). The first setting of the weights is

$$w_1 = w_4 = 1.0 \text{ and } w_2 = w_3 = 2.0 \quad (32)$$

further on referred to as problem instance 1. The second setting is

$$w_1 = w_4 = 1.0 \text{ and } w_2 = w_3 = 100.0 \quad (33)$$

further on referred to as problem instance 2. We used the formulas derived in the last section[4] to produce the analytical curves showing (i) the performance of the system as defined in (31), and (ii) the evolution of the four model parameter values over time. We also run Alg. 1 with 10000 solution constructions per iteration for four different values for parameter $\rho \in \{0.5, 0.1, 0.05, 0.01\}$ on both problem instances to see the empirical behavior of the system. The results are shown in Fig. 3–5. In Fig. 3 and Fig. 4, we observe

[4]Formulas (18)–(21) for the evolution of model parameter values, and formulas (11)–(13) for the evolution of the probabilities to produce the different solutions.

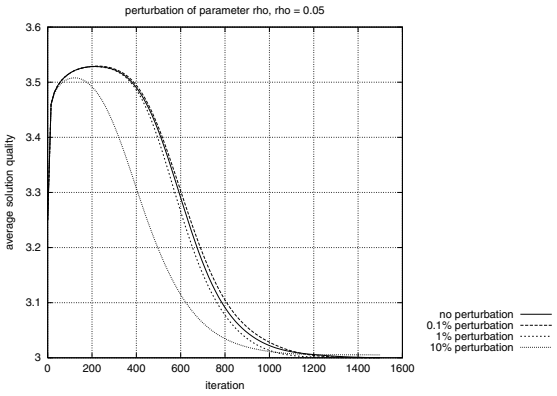


Figure 2: This graph shows four different empirically obtained performance curves for problem instance 1, $\rho = 0.05$ (averaged over 100 experiments). The curve labeled "no perturbation" is the curve for the system starting with model parameter values $\tau_{v_1} = \tau_{v_2} = \tau_{v_3} = \tau_{v_4} = c$, where c is the starting value for the model parameters. The other curves show the performance of the system when there is a random perturbation on the starting model parameters. 0.1% perturbation for example means that for every model parameter value we were tossing a coin: this means that to equal probability we either added $\frac{1}{1000} \cdot c$ to τ_{v_i} or we subtracted $\frac{1}{1000} \cdot c$ from τ_{v_i}. We did that for each of the four model parameter values independently.

that the performance of the system in the analytical and also in the empirical case decreases quite drastically in the first couple of hundred iterations. Then, the analytic curves seem to converge (with still decreasing performance) to a fixed point of the system (not being one of the solutions). This fixed point is implicitly defined by equation (26)[5]. On the other hand, the empirically obtained curves show even a slightly higher decrease in performance at the beginning, but then near the equilibrium the sensitivity to sampling errors seems to increase rapidly. As a result, the system converges to one of the two optimal solutions (T_a or T_b). For problem instance 2, the number of iterations needed for both, the analytical and the empirical curves, to reach there limits is lower than for problem instance 1. Qualitatively the results are the same for the two different weight settings. This is in accordance with the results of the last section, which are not dependent on the relative difference between weights $w_1 = w_4$ and $w_2 = w_3$.

[5]This claim is supported by the fact that if we take the analytically obtained convergence values (exact up to four decimal digits) for the four model parameter values and substitute them in equations (24), (25) and (26) we get equality in (26) for up to decimal digits.

We also compared the evolution of the four model parameter values over time, analytically as well as empirically. The graphs are shown in Fig. 5. In the analytic case, the evolution of model parameter values τ_{v_1} and τ_{v_4} (resp. τ_{v_2} and τ_{v_3}) is the same whereas in the empirical case the evolution is the same at the beginning, until the system nearly reaches the equilibrium and then, due to the sampling error, begins to drift toward one of the two optimal solutions.

The last set of experiments we performed was to investigate the influence of perturbations of the starting model parameter values. To perturb the model parameter values (initially set to a constant c), a value $\frac{1}{x} \cdot c$ was either added or substracted to equal probability. For x we considered values 10, 100 and 1000. The results are shown in Fig. 2. With $x = 1000$ (just a slight perturbation) the convergence of the system seems to be slightly slowed down, but at the end it is reaching the state of convergence slightly faster than the system without perturbation. With the choice of $x = 100$, we notice a bigger advantage in convergence speed in all phases. The choice of $x = 10$ (which corresponds to a strong perturbation) shows a much higher convergence speed until about 1000 iterations at which point the speed of convergence gets really slow and the system does not even converge before the 1500 iteration limit. These results suggest that a slight perturbation of the initial model parameter values is useful (but it must neither be too low nor too high).

6 Conclusions and outlook to the future

In this paper we showed that – unlike what is usually expected – the expected performance of model-based search algorithms using positive feedback can decrease over time in certain settings. We want to make clear at this point, that the results presented in this paper have not uncovered a general weak point of learning systems based on positive feedback as such. We believe that it is rather in the nature of algorithms such as Ant System [3] and related algorithms such as Population Based Incremental Learning (PBIL) and Estimation of Distribution algorithms (EDAs) that such phenomena can occur when applied to a certain kind of problem or problem instances with a certain kind of structure. In the case of the problem and problem instance chosen in this paper we have a structure of two equally good solutions competing in the system and a bad solution taking profit from that. We also point out that the update rule is a crucial component of a model-based search algorithm. If we chose a different update rule in the example presented – for instance a rule only using the best solution found since the start of the algorithm for updating the model parameter values – the algorithm wouldn't show a decreasing performance. In the future we intend to investigate into the interactions between parameter model and model parameter update rule in order to improve the understanding about phenomena related to the one presented.

Acknowledgments: This work was supported by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential program of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

[1] S.Y. Cheung and A. Kumar. Efficient quorumcast routing algorithms. In *Proceedings of INFOCOM'94*, Los Alamitos, USA, 1994. IEEE Society Press.

[2] M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.

[3] M. Dorigo, V. Maniezzo, and A. Colorni. Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics–Part B*, 26(1):29–41, 1996.

[4] L.R. Foulds and H.W. Hamacher. A new integer programming approach to (restricted) facilities layout problems allowing flexible facility shapes. Technical Report 1992-3, University of Waikato, Department of Management Science, 1992.

[5] N. Garg and D. Hochbaum. An $O(\log k)$ approximation algorithm for the k minimum spanning tree problem in the plane. *Algorithmica*, 18:111–121, 1997.

[6] H.W. Hamacher and K. Joernsten. Optimal relinquishment according to the Norwegian petrol law: A combinatorial optimization approach. Technical Report No. 7/93, Norwegian School of Economics and Business Administration, Bergen, Norway, 1993.

[7] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Proceedings of the 4th Conference on Parallel Problem Solving from Nature – PPSN IV*, volume 1411 of *Lecture Notes in Computer Science*, pages 178–187, Berlin, 1996. Springer.

[8] M. Pelikan, D.E. Goldberg, and F. Lobo. A survey of optimization by building and using probabilistic models. Technical Report No. 99018, IlliGAL, University of Illinois, 1999.

[9] G.J. Woeginger. Computing maximum valued regions. *Acta Cybernetica*, 4(10):303–315, 1992.

[10] M. Zlochin, M. Birattari, N. Meuleau, and M. Dorigo. Model-based search for combinatorial optimization. Technical Report TR/IRIDIA/2001-15, IRIDIA, Université Libre de Bruxelles, Belgium, 2001. Submitted to Annals of Operations Research.

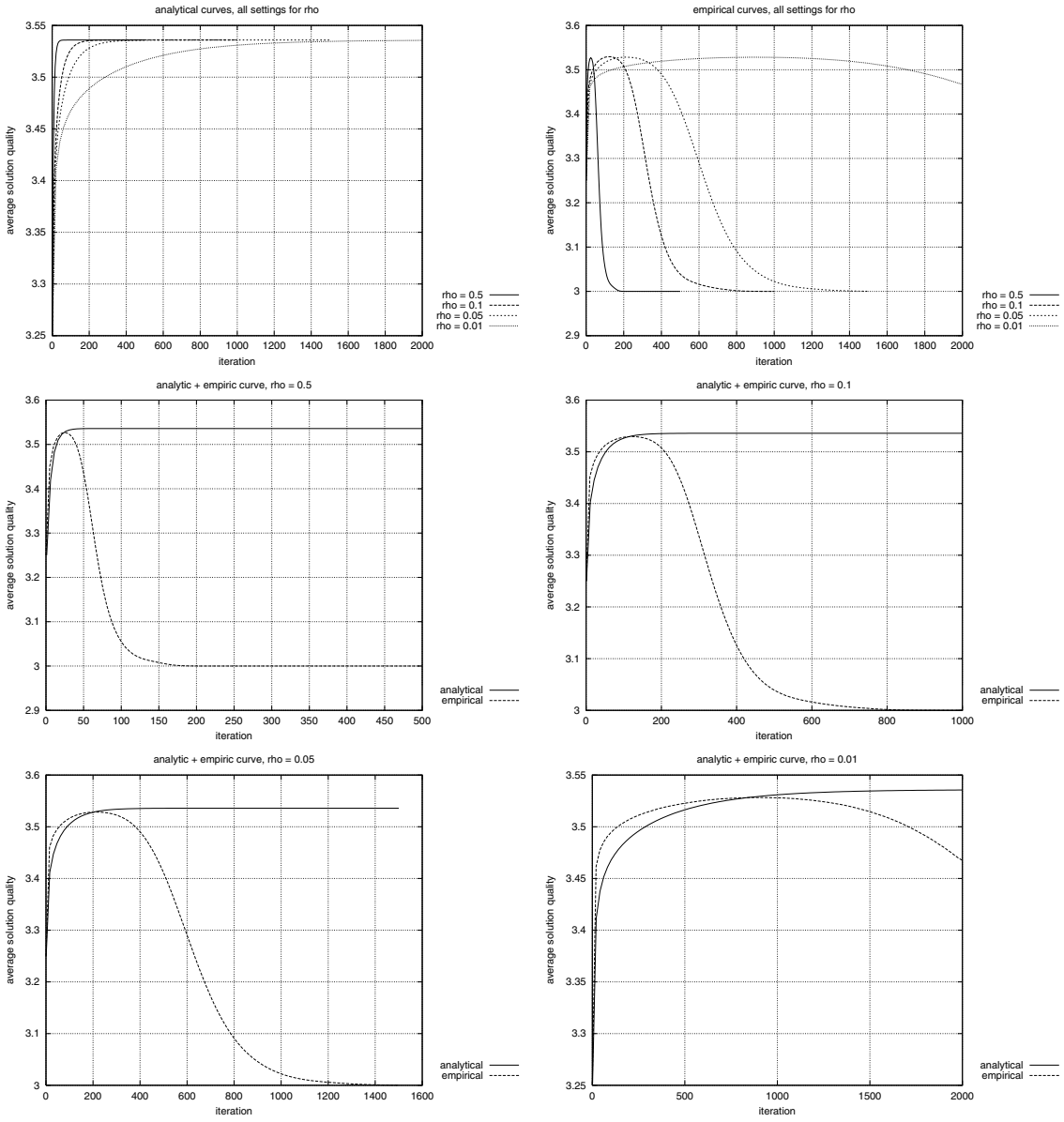


Figure 3: Performance curves for problem instance 1. The graph in the upper left corner shows the analytic curves for four different values for $\rho \in \{0.5, 0.1, 0.05, 0.01\}$. In contrast to that the graph in the right upper corner shows the empirically obtained curves for 10000 solution constructions per iteration (the curves are averaged over 100 experiments). The other four graphs show the comparison of the analytic and the empirical curve for every one of the four settings for ρ.

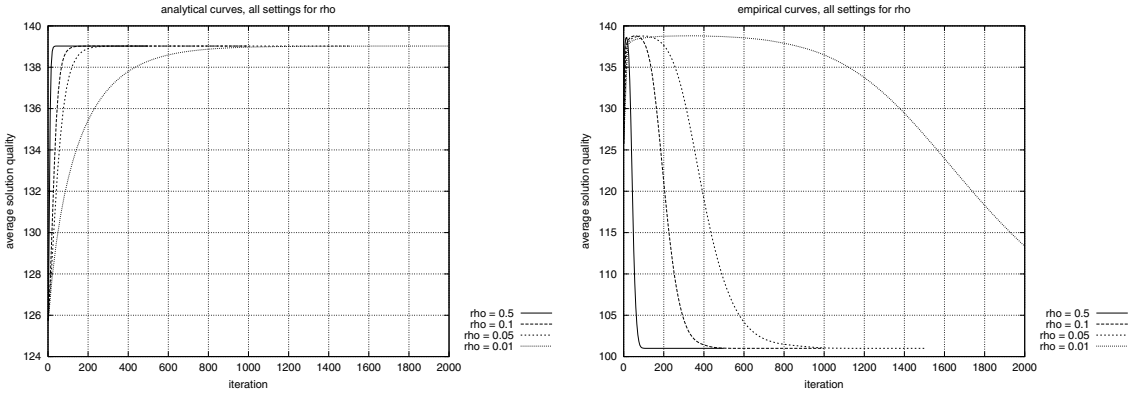


Figure 4: Performance curves for problem instance 2 (higher weights on v_2 and v_3). The graph on the left shows the analytic curves for four different values for $\rho \in \{0.5, 0.1, 0.05, 0.01\}$. In contrast, the graph on the right shows the empirically obtained curves for 10000 solution constructions per iteration (the curves are averaged over 100 experiments).

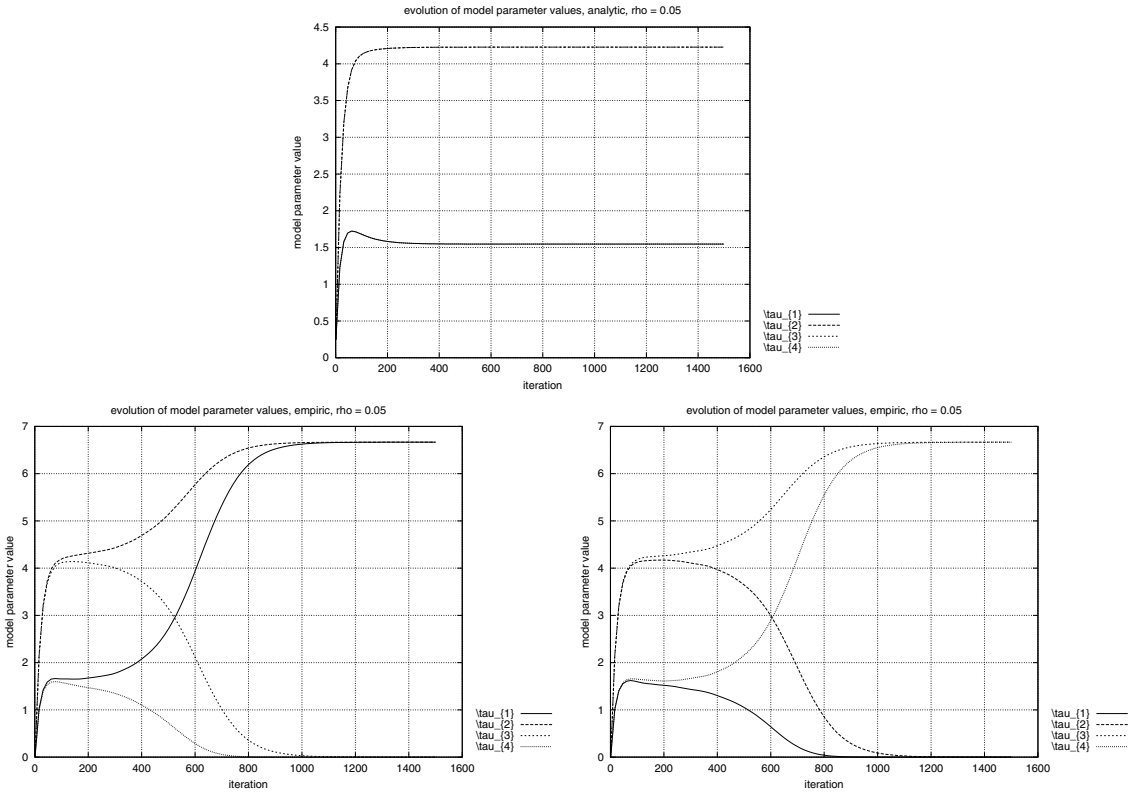


Figure 5: The three graphs show the evolution of the four model parameter values exemplary for problem instance 1, $\rho = 0.05$. The top graph shows the analytic curves (there are just two curves visible, because the curves for τ_{v_1} and τ_{v_4} (τ_{v_2} and τ_{v_3} respectively) are exactly the same). In contrast, the two lower graphs show the evolution of model parameter values empirically obtained. In the graph on the left the system is converging to solution T_a, and in the graph on the right to the solution T_c.

An Ant System Algorithm for Graph Bisection

Thang N. Bui
Dept. of Computer Science
Penn State Harrisburg
Middletown, PA 17057

Lisa C. Strite
Dept. of Computer Science
Penn State Harrisburg
Middletown, PA 17057

Abstract

This paper gives an algorithm for the graph bisection problem using the Ant System (AS) technique. The ant algorithm given in this paper differs from the usual ant algorithms in that the individual ant in the system does not construct a solution to the problem nor a component of the solution directly. Rather the collective behavior of the two species of ants in the system induces a solution to the problem. The algorithm also incorporates local optimization algorithms to speed up the convergence rate and to improve the quality of the solutions. The results achieved by this algorithm on several classes of graphs equal the best known results for the majority of graphs tested, and are very close to the best known results for the remainder.

1 Introduction

Let $G = (V, E)$ be a graph on n vertices, where n is even. A *bisection* of G is a partition of the vertex set V into two disjoint sets A, B of equal size, i.e., $A \cup B = V, A \cap B = \emptyset$, and $|A| = |B|$. Such a bisection is denoted by (A, B). The *cut size* of a bisection (A, B) is the number of edges that have one endpoint in A and the other endpoint in B. The *graph bisection problem* is the problem of finding a bisection of minimum cut size for a given graph. The graph bisection problem is well-known to be $\mathcal{NP}$-hard [13]. It arises in a wide variety of problems including VLSI placement and routing, sparse matrix computation, and processor allocation [6][7][16]. Since the problem is $\mathcal{NP}$-hard, efforts have been concentrated on designing efficient approximation algorithms and heuristics for solving it. In this paper we use ideas from Ant System (AS) [11] to design an algorithm for the graph bisection problem.

Our algorithm incorporates several AS features as well as local optimization techniques and graph preprocessing. The algorithm was tested on five classes of graphs ranging in size from 500 to 5,252 vertices with average degrees from 2 to 36. The results were compared with the best known results for each graph as well as results from several other heuristic algorithms. For the majority of graphs tested, the algorithm produced the best known results. For the remaining graphs the results produced by the algorithm are very close to the best known solutions. A major advantage of this algorithm compared to other existing algorithms for the graph bisection problem is that our algorithm is very amenable to parallelization.

The rest of the paper is organized as follows. In Section 2 we provide some background information on the graph bisection problem and AS. We describe our algorithm in Section 3. The performance of the algorithms on the test graphs is described in Section 4 and conclusions are given Section 5.

2 Preliminaries

2.1 The Graph Bisection Problem

As mentioned above the graph bisection problem is $\mathcal{NP}$-hard and thus we do not expect to have a polynomial time algorithm for solving it. For special types of graphs there are polynomial time algorithms for solving it exactly, e.g., k-outerplanar graphs, for fixed k, or planar graphs whose optimal bisection is of size $O(\log n)$ [8]. However, the complexity of the problem on planar graphs remains an open question.

One approach to $\mathcal{NP}$-hard problems is to find efficient approximation algorithms. Currently, the best known polynomial time approximation algorithm for bisecting graphs can find a solution that is within $O(\sqrt{n} \log n)$ of the optimal [12]. It has been shown that it is $\mathcal{NP}$-hard

to find a bisection that is within an additive factor of $O(n^{1/2-\epsilon})$ of the optimal, for any $\epsilon > 0$ [5].

Since the best approximation algorithm still has a rather large approximation ratio and is rather complicated to implement, heuristics are often used for the graph bisection problem in practice. Heuristics are algorithms that do not have performance guarantee as approximation algorithms do. However, they usually are fast and produce solutions that are very good. Generally, one can classify the heuristic algorithms for bisecting graphs into two main groups: local methods and global methods. Local methods include the greedy algorithm, Kernighan-Lin, simulated annealing and multi-level algorithms [14][15][16]. Global methods include spectral algorithms, flow based algorithms and genetic algorithms [4][7][19]. The Kernighan-Lin algorithm is one of the first efficient algorithms for the graph bisection problem. We briefly describe the Kernighan-Lin algorithm here since we use it in our algorithm later on.

2.2 The Kernighan-Lin Algorithm

Kernighan-Lin is a local optimization algorithm for the graph bisection problem [16]. The algorithm starts with a bisection (A, B), either created randomly or as the result of some other algorithm. The algorithm consists of a number of passes. During one pass of the algorithm it interchanges equal sized subsets of A and B. The subsets to be interchanged are selected by first ordering the vertices of A and B, say $a_1, \ldots, a_{n/2}$ and $b_1, \ldots, b_{n/2}$. The algorithm then selects k such that swapping $a_1, \ldots, a_k$ with $b_1 \ldots, b_k$ will give the greatest reduction in the size of the current bisection over all choices of k. This constitutes one pass of the algorithm. The bisection produced by this pass is then used as input to the next pass. The algorithm may run for a fixed number of passes or until no more improvement can be made from the current bisection.

2.3 Ant System

Ant System (AS) is a heuristic technique that seeks to imitate the behavior of a colony of ants and their ability to collectively solve a problem. For example, it has been observed that a colony of ants is able to find the shortest path to a food source by marking their trails with a chemical substance called pheromone[3][11].

The Traveling Salesman problem was the first problem to which the Ant System (AS) technique was applied [2] [11]. Other problems that have been the focus of AS as well as Ant Colony Optimization (ACO) [10] work include the quadratic assignment, network routing, vehicle routing, frequency assignment, graph coloring, shortest common supersequence, machine scheduling, multiple knapsack and sequential ordering problems [18] [3].

In addition to the idea of finding shortest paths, the idea of territorial colonization and swarm intelligence can also be utilized in ant algorithms. Kuntz and Snyers applied these concepts to a graph clustering problem [17]. The organisms are called *animats*, reflecting the fact that the system draws ideas from several sources, not just ant colonies.

We combine these two ideas of animats following paths and forming colonies, together with the use of graph preprocessing and local optimization to develop an Ant System algorithm for the graph bisection problem, which we call ASGB. Our algorithm is described in the next section.

3 Ant System Algorithm for Graph Bisection

3.1 Main Ideas

The basic foundation of the algorithm is to consider each vertex in the graph as a location that can hold any number of animats. The animats can move around the graph by moving across edges to reach a new vertex. Each animat belongs to one of two species (called species A and B). However, animats of both species follow the same rules. Since movement of animats from vertices to vertices is an important part of the algorithm, the first step in the algorithm is to add edges, called *free edges*, to the input graph to make it connected if it is not. This is accomplished in two steps as follows.

First, we add the necessary free edges to connect all disconnected subgraphs. This is done by randomly selected a starting vertex and performing a depth first search until no new vertices can be reached. If any vertices were not reached, an edge is randomly placed between a vertex that was found in the search and a vertex that was not found. The depth first search then continues again and the process is repeated until all vertices have been joined to the graph.

Next, we add free edges between a number of vertices to improve the animats' ability to move and explore. The number of free edges added in this step is proportional to the number of vertices in the graph, but inversely proportional to the average degree. Thus a graph with a large number of vertices and a very low average degree will have the most free edges added while a graph with few vertices and a high average de-

gree will have the least free edges added. To select the locations of the free edges, a pool of possible pairs of vertices are randomly selected. The number of pairs in the pool is τ times the number of pairs that will actually be chosen. The distance between each pair is determined, in terms of the minimum number of steps needed to move from one vertex to the other, up to a certain maximum distance. Then pairs of vertices are selected randomly from the pool but in proportion to the distance between them. In other words, a pair of vertices that is furthest apart has the greatest chance of being selected.

Once the free edges are added, the animats consider them in the same manner as regular edges in selecting moves. However, when the cut size of a partition is calculated, the free edges are ignored.

The algorithm now starts by distributing α animats on the graph. (Note that the values of all parameters used in the description of the algorithm are given in Table 1.) Their species and location are chosen randomly. At any point throughout the algorithm, the configuration of animats on the graph constitutes a partition of the graph in the following way. Each vertex is considered to be colonized by one species. At a given time, it is said to be colonized by whichever species that has the greater number of animats on it. Ties are broken in a random order by assigning the vertex to the species that results in a lower cut size. The set of all vertices colonized by each species, called *colony*, makes up one half of the partition. This partition is not necessarily a bisection, since one colony may contain more vertices than the other. Thus, other techniques are used at certain points in the algorithm to ensure that the final solution is a bisection.

In addition, each vertex can hold pheromone. The two species produce separate types of pheromone, so each vertex has an amount of A pheromone and B pheromone.

The idea of the algorithm is for each species of animats to form a colony consisting of a set of vertices that are highly connected to each other but highly disconnected from the other colony. The result should be two sets of vertices that are highly connected amongst themselves, but have few edges going between the two sets.

The ASGB algorithm is divided up into σ sets each comprised of γ iterations. In each iteration a percentage of animats are activated. When an animat is activated, it adds an amount of pheromone to the vertex it is currently at based on conditions at the vertex. It then may die with a certain probability or it may reproduce with a certain probability and then moves to a new vertex. In each iteration, these activations are performed in parallel. After each iteration, the graph is updated with the new information.

After each set, the configuration of the graph is forced into a bisection using a greedy algorithm and a local optimization algorithm is run to help speed up the convergence rate. During each set, the parameters, which include probabilities for activation, death, reproduction and birth are varied. The parameters are varied in such a way that at the beginning of the set, the colonies change a great deal and by the end of the set the colonies have converged to a stable configuration. The next set begins at the state where the previous set ended. However, if the animats follow their usual rules immediately, they will not be able to move away from the local optimum that has been reached. So, for all but the initial set, a *jolt* is performed for a certain number of the first iterations to help move the configuration, or distribution of animats on the vertices, away from the local optimal solution to which it had converged. The jolt allows animats to select moves randomly instead of following the normal rules for movement. The length of the jolt is changed during the algorithm. The first jolt lasts for ν iterations and for subsequent jolts the length decreases linearly until the last set where there is no jolt. The idea is that with each successive set, the bisection should come closer to the optimal bisection, and thus shorter and shorter jolts are needed.

After σ sets have been completed, the solution is the best bisection that has been achieved. This is usually the bisection found by the last set; however, occasionally the best bisection is found earlier.

In the following subsections we will describe in detail what occurs in one iteration, what occurs when an animat is activated and what occurs between sets. The full ASGB algorithm is given in Figure 1.

3.2 Iteration

An iteration of the algorithm consists of a percentage of the animats being activated and then performing the necessary operations in parallel. The probability of an animat being activated changes during the set. At the beginning of the set, more animats are activated during each iteration. By the end of the set, only a small percentage of the animats are activated in each iteration. The actual probability of activation is a sigmoid-like function. The function starts at a maximum of $\pi_{a\,\max}$ and ends at $\pi_{a\,\min}$.

After the activations of animats have been completed, ϵ percent of the pheromone on each vertex is evap-

Figure 1: An Ant System algorithm for graph bisection

```
Preprocess the graph to make it connected
Randomly add α animats to graph
For set=1 to σ
  For time=1 to γ
    For each animat do (in parallel):
      Activate animat with probability a(time)
      If activated
        Add p(animat,time) pheromone to
            the animat's location
        Die with probability π_d
        If not dead
          If animat meets reproduction criteria
            Then reproduce with probability π_r
          If time is in a jolt period
            Then select move randomly
          Else select move based on pheromone
               and connectivity
        Endif
      Endif
    Endfor animat
    Evaporate ε percent of the pheromone from
          each vertex
  Endfor time
  Convert configuration to a bisection with
          a greedy algorithm
  Run Kernighan-Lin local optimization
  Reduce total number of animats
  Equalize number of animats in each species
Endfor set
Return best bisection found
```

Table 1: Parameter values

Param.	Value	Description
γ	1000	Number of iterations per set
σ	10	Number of sets
ν	50	Maximum jolt length
α	10000	Initial number of animats
$\pi_{a\,\max}$	0.8	Maximum activation probability
$\pi_{a\,\min}$	0.2	Minimum activation probability
π_d	0.035	Death probability
β_{init}	4	Expected number of animats born in first iteration
β_{final}	2	Expected number of animats born in final iteration
β_{range}	50%	Percentage range from average number of animats born
π_r	0.01	Reproduction probability
η	10	Max number of offspring per animat
μ	5	Number of moves needed before animat can reproduce
ψ_{stay}	20%	Percentage of offspring that stay on old location when not colonized by animat's species
ω_{pmin}	0	Minimum pheromone weight
ω_{pmax}	1	Maximum pheromone weight
ω_{cmin}	250	Minimum connection weight
ω_{cmax}	500	Maximum connection weight
π_{min}	0.1	Minimum probability for moving to a vertex
ρ	0.9	Reduction factor for returning to previous location
ψ_{swap}	75%	Percentage of vertices needed for swap
ψ_{maj}	90%	Percentage of animats needed for majority
ϵ	0.2	Evaporation rate
λ	1000	Pheromone limit
τ	50	Free edge factor

orated. This prevents pheromone from building up too much and highly populated vertices from being overemphasized, which in turn prevents the algorithm from converging prematurely.

3.3 Activation of an Animat

When an animat is activated, it deposits pheromone on its current vertex, dies with a certain probability or reproduces with a certain probability, and then moves to another vertex. These operations are performed by the animat by using local information to make decisions.

3.3.1 Pheromone

The purpose of pheromone is to allow the algorithm to retain a "memory" of good configurations that have been found in the past. The formula for the amount of pheromone to be deposited is:

$$p(a,i) = \frac{a_{col}}{a_{total}} \cdot \frac{i}{\gamma}$$

where a is the animat, i is the iteration number, a_{col} is the number of vertices adjacent to the animat's current location which are colonized by the animat's species, and a_{total} is the total number of vertices adjacent to the animat's current location. The idea here is for an animat to deposit more pheromone at a vertex if that vertex is highly connected to vertices colonized by its own species. Also, less pheromone is used in early iterations to allow for more exploration and more pheromone is used later on to emphasize exploitation. Even though the number of neighbors of a seems to relate more directly to the cut size, the amount of pheromone deposited is made proportional to the fraction a_{col}/a_{total}, to prevent the amount of pheromone at any vertex from growing out of control, even with evaporation.

There is also a limit to the amount of pheromone of each species that can be stored on a vertex. The limit

for a vertex is the product of the degree of that vertex and the pheromone limit parameter (λ). This allows densely connected vertices to accumulate more pheromone. The more highly connected a vertex is, the more essential it is that it is colonized by the right species. This is because a mistake on a highly connected vertex will mean a much greater cut size.

3.3.2 Death

Next, the animat may be selected to die. The animat die with probability π_d, which is fixed throughout the algorithm. However, the activation probability changes throughout the set, so that early in the set, more animats are activated, and therefore more animats die early in the set. The purpose of this is to have shorter life spans in the beginning, which allows more turnover and change in the configuration. Later in the set, the animats live longer and thus there is less change and the solution is able to converge.

3.3.3 Reproduction

If the animat is not selected for death, the algorithm proceeds to the reproduction step. The animat is selected for reproduction with fixed probability π_r. However, the number of new animats that are produced depends on time. In the first iteration of a set, the average number of animats born is β_{init} and it decreases linearly over time to β_{final} in the last iteration. The changing birth rate serves to allow more change in earlier iterations, in which animats live for shorter lengths of time. In later iterations, fewer animats are born, but they live longer. The actual number of animats born is selected uniformly at random over a range centered on the average birth rate for the iteration. The number of animats born can be up to β_{range} more or less than the specified average.

If the vertex on which the parent is located is colonized by its own species, the offspring animats are all placed on that vertex. However, if the vertex is colonized by the opposite species, only ψ_{stay} percent of the offspring animats are placed there. The remaining new animats will be placed on the vertex to which the parent animat moves in the next step. The rationale is that if the parent animat is already in its own colony but moves to another vertex, it should leave its offspring behind to help maintain the majority on that vertex. However, if the parent's species is not in majority, it should take most of its children to the new vertex in which it is trying to create a colony. The parent leaves some of its offspring behind however, so that some of its species remain at the vertex (in case that vertex really should be part of their colony).

There are two other constraints on reproduction. First, there is a limit to how many offspring an animat can produce during its lifetime (η). This value is fixed throughout the algorithm and is the same for each animat. Once the limit is reached, the animat can no longer reproduce. This serves to prevent one species from taking over the graph and forcing the other species into extinction.

To prevent a species from overemphasizing a vertex through reproduction, the animats are not allowed to reproduce until they have made a set minimum number of moves (μ). This ensures that the graph is explored and that new configurations are created by the reproduction and movement rather than being inhibited by these operations.

3.3.4 Movement

An animat can move to any vertex which is connected to its current location by an edge. There are two factors used to select a move from the set of possible moves. For each vertex to which the animat could move, the connectivity to other vertices is examined. The animat should move to a vertex that is highly connected to other vertices colonized by its own species. This factor gives an indication of the current configuration of the graph. In addition, the animat should learn from the past and take into account the pheromone that other animats have deposited. Throughout the course of a set, these two factors are weighted differently. Initially, the pheromone is weighted at ω_{pmin} with the weight increasing linearly to ω_{pmax}. Conversely, the connectivity is weighted at ω_{cmax} to begin and decreases linearly to ω_{cmin}. In this way, the configuration of the colonies changes greatly in early iterations and over time learning is incorporated into the algorithm. These basic factors drive the animats to create colonies of highly connected vertices which are highly disconnected from the vertices colonized by the opposing species.

These factors are the basis of move selection. The probability of moving to an adjacent vertex is proportional to the two combined factors. Specifically, the factors are combined as follows to create a "probability" of moving to a specific vertex v:

$$pr(v) = cv_c + pv_p + \pi_{min}$$

where v_c is the number of vertices adjacent to v that are colonized by the animat's own species, c is the connectivity weight and $\omega_{cmin} \leq c \leq \omega_{cmax}$, v_p is the amount of pheromone of the animat's species on

vertex v, p is the pheromone weight and $\omega_{pmin} \leq p \leq \omega_{pmax}$, and π_{min} is a fixed amount added to prevent any probabilities from being zero.

In addition, one more factor is considered in selecting a move. To encourage the animats to explore more of the graph in the early sets, the probability of selecting the move which would result in the animat returning to its previous location is reduced. The factor it is reduced by starts at ρ and decreases linearly after each set until it reaches zero in the final set. Then the probability of moving to a connected vertex is the resulting value divided by the sum of values over all possible moves.

3.4 Between Sets

After each set of iterations, several other operations are performed. They help nudge the configuration into a bisection, improve the bisection through local optimization and then prepare the configuration for the next set.

First, the algorithm looks for "mistakes" the animats have made. Here the algorithm looks for vertices in which a very high percentage (ψ_{swap}) of the adjacent vertices are colonized by the opposite species. In these cases, the vertex is swapped to the other colony. This is achieved by changing the species of animats on the vertex until the new species attains ψ_{maj} percent of the animats. In most cases, few such vertices are found.

Next the colonies are manipulated to produce a bisection. As was discussed earlier, any given configuration of animats on the graph does not necessarily induce a bisection. Therefore, if one species is colonizing more vertices than the other, some vertices will have to be swapped to the other species. The vertices to be swapped are selected from the set of fringe vertices, that is, vertices that are adjacent to a vertex of the opposite colony. By only changing the colonizing species on fringe vertices, the algorithm continues in the direction the animats were heading. Vertices are selected to be swapped by making the greedy choice from amongst the fringe vertices.

Using the bisection produced by this greedy optimization, a weak version of the Kernighan-Lin algorithm is run. Since the quality of the result produced by the Kernighan-Lin algorithm depends largely on the quality of the bisection used as input, it produces little if any improvement in early sets. However, in later sets, after the animats have begun to converge upon a good solution, it usually improves the solution slightly.

Even though we now have a bisection, the number of animats on the graph may differ from the initial number of animats of both species. To prevent the animat population from growing unchecked the algorithm removes animats at random until the population size reaches its initial value. This may disrupt the colonies, however, this is not a problem since each new set begins with a jolt anyway.

Finally, to prevent one species from dominating the graph, the number of animats in the two species is equalized. This is done by adding animats to equalize the number of animats in each species. Usually this is a very small number and thus is not problematic in consideration of the previous operation (reducing the number of animats to the initial number). The new animats are added only to vertices where their own species is already in majority. Thus, this operation does not significantly alter the configuration of the colonies; it merely gives added strength to the colonies in which animats are added.

Following this operation, a new set is begun. Again, the time is initialized to 0 and all probabilities relating to time are reset. Thus, as the animats have converged on a possible solution, starting a new set allows the animats to move away from that solution in expectation of finding a better solution in case this solution was a local optimum. After σ sets have been completed, the solution is the bisection with minimum cut size.

4 Results

Using the parameter values listed in Table 1, the algorithm was tested on a total of forty graphs of five different types to determine its behavior on a wide selection of inputs. These graphs are used as a benchmark as they have been used to test a number of different graph bisection algorithms [1][7]. Thus the results can be compared with other algorithms. The graphs range in size from 500 to 5,252 vertices and have average degrees from 2 to 36. The algorithm was implemented in C++ and run on a Pentium III 800MHz with 256 MB RAM. For each graph, the algorithm was run for 100 trials. These results are given in Table 2 which also gives average running time in seconds for one trial of each graph. In this section, the five graph types are described and the results for different graph types are discussed.

4.1 Graphs Types

In [14], Johnson *et al.* described two classes of graphs that we use to test our algorithm. The first type, $Gn.p$, is a random graph on n vertices where an edge is placed between two vertices with probability p, independent

of all other edges. The expected vertex degree is then $p(n-1)$. These graphs are a good test case as they have large optimal bisections. The second type, $Un.d$, is a random geometric graph on n vertices with expected vertex degree d. It is generated by selecting n points within the unit square which represent the vertices. An edge is placed between two vertices if their Euclidean distance does not exceed t. It can be shown that the expected vertex degree is $d = n\pi t^2$. This type of graph is highly clustered so it provides a very different test case than the previous class of graphs.

Three other graph types were proposed by Bui *et al.* in [4]. They defined the class of random regular graphs, Breg$n.b$, on n vertices with degree 3 having an optimal cut size b with probability $1 - o(1)$. These graphs provide an interesting test case because of they are sparse and have a provable unique optimal bisection with high probability. A grid graph, Grid$n.b$, on n vertices is a grid with known optimal cut size b. A variation of this type is W-Grid$n.b$ in which the grid boundaries are wrapped around. This class of graphs is highly structured with good connectivity. The last class of graphs used is the caterpillar graph, Cat.n, on n vertices with an optimal cut size of 1. It is constructed by starting with a spine, which is a straight line in which all vertices except the two ends have degree 2. Then to each vertex on the spine, called a node, we add six legs each of which consist of adding a vertex and connecting it to the node on the spine. If the number of nodes on the spine is even, the optimal cut size of 1 is found by dividing the spine in half. In addition, RCat.n is a caterpillar graph in which each node on the spine has degree $\sqrt{n}$. Caterpillar graphs seem simple but are difficult for local bisection algorithms.

4.2 Comparison with other algorithms

The results of the ASGB algorithm are compared with results from three other algorithms in Table 2. The table compares the best result achieved by each algorithm in a fixed number of trials. For the ASGB algorithm, 100 trials were run for each graph with either 10 or 25 sets depending on the difficulty of the graph for the algorithm. Results for other algorithms reflect 1,000 trials as this was the data that was available from the sources. The last three columns of Table 2 contain the average and the standard deviation of the solutions returned by ASGB in 100 trials, as well as the average running time.

Battitti and Bertossi gave a Reactive and Randomized Tabu Search (RRTS) in [1]. The Multi-Start Kernighan-Lin (KL) consists of running the Kernighan-Lin algorithm 65 times on a new random bisection each time. The final result is the minimum cut size of the 65 results. Since the results of KL are greatly affected by the quality of input, it is necessary to run it many more times to achieve good results since random bisections usually have poor cut sizes. This allows us to compare KL with other algorithms which normally would outperform it. The results for Multi-Start KL, Simulated Annealing (SA) and the best known results are taken from [7]. The sources provide results for most graphs in the benchmark set, however, when the results for a graph are not provided by the source, the corresponding entry is left blank.

Overall, ASGB got the best known solution for 27 of the 40 graphs tested. When the best known solution is not 1, the best solution returned by ASGB is less than 5% away from the optimal, usually much less. Generally, ASGB performed best when the input graphs have some clustering structure and enough connectivity that allow the animats to discover a good bisection, e.g., U$n.d$, Breg$n.b$ and grid graphs. The caterpillar graphs have regular structure, but they do not have enough connectivity to allow the animats to explore the graph easily. The effect of random free edges added in the preprocessing step is not enough to overcome this deficiency. We did observe that if the number of sets is increased to 30 ASGB returns the optimal answer for almost all caterpillar graphs. It seems that the larger the caterpillar graphs, the more sets are required to get the optimal solution. For the class G$n.p$, ASGB either produced the best known solution or solutions that are within at most 5% of the best known.

Generally, we can see from Table 2 that ASGB is better than Multi-Start KL and SA and is very competitive with RRTS, noting that the data from these algorithms are from 1,000 trials for each graph.

5 Conclusion

An algorithm, called ASGB, using Ant System techniques with local optimization and graph preprocessing was developed for solving the graph bisection problem. Animats from two different species are placed on a graph and follow a set of local rules. The emergent behavior of the population following these rules, coupled with a local optimization, results in a bisection of the graph with low cut size. The results achieved were equal or very close to the best known results for the set of benchmark graphs. Even though the results achieved by ASGB were not always as good as the results of RRTS it seems that ASGB is more amenable

Table 2: Comparison of ASGB results with other algorithms

Graph	Best known	ASGB	RRTS	Multi-Start KL	SA	ASGB Avg	S.D.	Time
G500.005	49	51	51	52	52	57.52	2.38	139.36
G500.01*	218	218	218	220	219	225.29	3.51	412.72
G500.02	626	626	626	627	628	633.62	3.19	139.47
G500.04	1744	1744	1744	1744	1744	1752.98	3.56	197.14
G1000.0025*	95	97	96	101	102	103.82	3.57	426.96
G1000.005*	445	450	447	457	451	459.25	4.03	460.04
G1000.01*	1362	1367	1362	1376	1367	1377.51	3.89	542.38
G1000.02	3382	3385	3382	3390	3389	3399.88	7.63	222.83
U500.05*	2	2	2	5	4	14.28	5.75	462.01
U500.10*	26	26	26	26	26	61.66	16.93	494.16
U500.20	178	178	178	178	178	209.75	28.92	207.72
U500.40	412	412	412	412	412	461.43	54.13	297.55
U1000.05*	1	3	1	15	3	21.76	6.56	395.02
U1000.10	39	39	39	39	39	116.59	31.74	172.45
U1000.20	222	222	222	222	222	293.48	58.87	249.92
U1000.40	737	737	737	737	737	873.58	145.31	333.40
Breg500.0	0	0	0	0	–	0.00	0.00	132.48
Breg500.12	12	12	12	12	–	13.36	8.06	123.39
Breg500.16	16	16	16	16	–	16.68	4.82	138.87
Breg500.20	20	20	20	20	–	20.00	0.00	123.20
Breg5000.0	0	0	0	0	–	0.00	0.00	323.28
Breg5000.4	4	4	4	4	–	4.00	0.00	324.84
Breg5000.8	8	8	8	8	–	8.00	0.00	309.73
Breg5000.16	16	16	16	16	–	16.00	0.00	328.04
Grid100.10	10	10	10	10	–	10.06	0.45	137.39
Grid1000.20	20	20	20	20	–	23.64	8.69	135.74
Grid500.21	21	21	21	21	–	23.00	4.62	120.69
Grid5000.50	50	50	50	50	–	53.98	12.84	394.90
W-Grid100.20	20	20	20	20	–	20.00	0.00	120.37
W-Grid1000.40	40	40	40	40	–	43.98	13.76	143.98
W-Grid500.42	42	42	42	42	–	44.50	4.31	128.36
W-Grid5000.100	100	100	100	100	–	104.64	16.98	368.78
Cat.352	1	3	1	3	–	6.08	1.64	139.03
Cat.702	1	3	1	13	–	10.18	2.54	18.82
Cat.1052	1	7	1	25	–	13.14	3.01	155.51
Cat.5252	1	14	–	165	–	31.30	4.21	382.69
RCat.134	1	1	1	1	–	2.68	0.97	150.03
RCat.554	1	3	1	1	–	7.00	1.61	194.23
RCat.994	1	5	1	3	–	9.42	1.91	210.48
RCat.5114	1	7	–	17	–	14.50	2.81	528.28

* graph is run with 25 sets instead of 10

to parallelization than the RRTS algorithm.

Acknowledgement

The authors would like to thank Karthik Balakrishnan and the anonymous referees for helpful suggestions.

References

[1] R. Battiti and A. Bertossi, "Greedy, Prohibition, and Reactive Heuristics for Graph Partitioning," IEEE Trans. on Comp., 48(4), 1999, pp. 361–385.

[2] E. Bonabeau, M. Dorigo and G. Theraulaz, *Swarm Intelligence*, Oxford University Press, 1999.

[3] E. Bonabeau, M. Dorigo and G. Theraulaz, "Inspiration for Optimization from Social Insect Behavior," Nature, Vol. 406, July 6, 2000, pp. 39–42.

[4] T. N. Bui, S. Chaudhuri, F. T. Leighton and M. Sipser, "Graph Bisection Algorithms With Good Average Case Behavior," Combinatorica, 7(2), 1987, pp. 171–191.

[5] T. N. Bui and C. Jones, "Finding Good Approximate Vertex and Edge Partitions is NP-Hard," Inf. Processing Letters 42 (1992), pp. 153–159.

[6] T. N. Bui and C. Jones, "A Heuristic for Reducing Fill-In in Sparse Matrix Factorization," Proc. of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, 1993, pp. 445–452.

[7] T. N. Bui and B. R. Moon, "Genetic Algorithm and Graph Partitioning," IEEE Trans. on Computers, 45(7), 1996, pp. 841–855.

[8] T. N. Bui and A. Peck, "Partitioning Planar Graphs," SIAM Journal of Computing, 21(2), April 1992, pp. 203–215.

[9] A. Colorni, M. Dorigo and V. Maniezzo, "Distributed Optimization by Ant Colonies," Proc. of the First European Conference on Artificial Life, New York: Elsevier, 1991, pp. 134–142.

[10] M. Dorigo and G. Di Caro, "The Ant Colony Optimization Meta-Heuristic," in New Ideas in Optimization, D. Corne, M. Dorigo and F. Glover, Editors, McGraw-Hill, 1999, pp. 11–32.

[11] M. Dorigo and L. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem," IEEE Trans. on Evol. Computation, 1(1), 1997, pp. 53–66.

[12] U. Feige, R. Krauthgamer and K. Nissim, "Approximating the Minimum Bisection Size," Proc. of the Thirty-Second Annual ACM Symposium on Theory of Computing, 2000, pp. 530–536.

[13] Garey, M. R. and D. S. Johnson, *Computers and Intractibility : A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[14] D. S. Johnson, C. Aragon, L. McGeoch, and C. Schevon, "Optimization by Simulated Annealing: An Experimental Evaluation, Part 1, Graph Partitioning," Operations Research, Vol. 37, 1989, pp. 865–892.

[15] G. Karypis and V. Kumar, "Analysis of Multilevel Graph Partitioning," Proc. of the 7th Supercomputering Conference, 1995.

[16] B. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graph," The Bell System Tech J., 49(2), 1970, pp. 291–307.

[17] P. Kuntz and D. Snyers, "Emergent Colonization and Graph Partitioning," Proc. of the Third Int. Conference on Simulation of Adaptive Behavior: From Animals to Animats 3, 1994, pp. 494–500.

[18] V. Maniezzo and A. Carbonaro, "Ant Colony Optimization: An Overview", in Essays and Surveys in Metaheuristics, C. Ribeiro editor, Kluwer Academic Publishers, 2001, pp. 21–44.

[19] P. Merz and B. Freisleben, "Fitness Landscapes, Memetic Algorithms and Greedy Operators for Graph Bi-Partitioning," Evolutionary Computation, 8(1), 2000, pp. 61–91.

The Evolution of Variable Learning Rates

John A. Bullinaria
School of Computer Science
The University of Birmingham
Birmingham, B15 2TT, UK
j.bullinaria@physics.org

Abstract

Neural plasticity in humans is well known to be age dependent, with 'critical periods' for the learning of many tasks. It is reasonable to hypothesise that this has some intrinsic advantage over constant plasticity, and that it has arisen as the result of evolution by natural selection. If this is true, then it may also prove useful for building more efficient artificial systems that are required to learn how to perform appropriately. In this paper I explore these ideas with a series of explicit evolutionary simulations of some simplified control systems.

1 INTRODUCTION

Evolutionary algorithms have shown much promise for generating artificial neural networks with performance superior to those formulated directly by human researchers. Factors such as network architecture, learning rules and connection weights have all been successfully optimised by evolution (e.g., Yao, 1999). A similar approach can equally well be applied to optimising the adjustable parameters and learning rates in other systems that learn, such as traditional adaptable controllers (e.g., Levine, 1996; Bullinaria, 2001). In this paper I take this work one stage further by considering how an evolutionary approach might lead to more efficient systems by allowing the emergence of non-constant learning rates.

It is well known that human neural plasticity varies considerably with age, and that there are "critical periods" during which learning must take place if the given task is to be mastered successfully (Julesz & Kovacs, 1995). The idea of variable neural plasticity is also quite common in the field of artificial neural networks where modellers have found it beneficial to vary their network learning rates during the course of training (Jacobs, 1988). For example, near the end of training it may be useful to decrease the learning rates to minimise the weight variations seen after each sample in online training, or to increase them to speed the saturation of sigmoids as the errors become small. Alternatively, if the performance of a task depends crucially on some lower level of processing, it may be sensible to delay the learning of that task until the lower level processes have fully developed. It is not clear to what extent factors such as these have been responsible for the evolution of the patterns of plasticity found in humans, or if it has been more a matter of minimizing the physical overheads of the plasticity. In this paper I shall present a series of explicit simulations of the evolution of some simple adaptable control systems. The evolutionary processes will result in efficient patterns of variable learning rates for these artificial systems that can then be used to develop better learning strategies for real world applications, and perhaps also provide some constraints on our explanations of the critical learning periods found in humans. The overall aim will be to see which learning strategies evolve naturally, and to explore how different strategies evolve under different circumstances.

2 THE CONTROL MODEL

The control system that will form the basis of the current investigation is shown in Figure 1. It is actually a simplified version of the part of the oculomotor control system that focuses and rotates the human eye (Schor et al., 1992), though similar systems can be applied quite generally (Levine, 1996). The input is a sequence of target responses and a feedback loop allows the determination of an error signal. This signal then feeds into standard simple integral and proportional controllers, the outputs of which are added to bias and

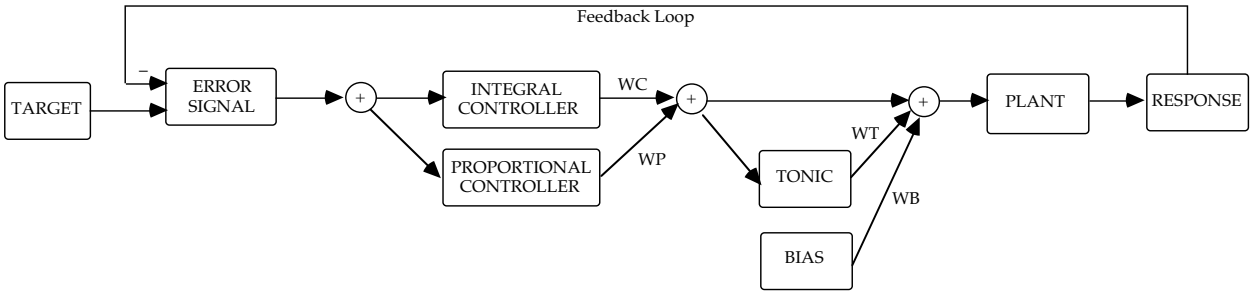


Figure 1: A simplified control model with four learnable parameters: WC, WP, WT, WB.

tonic signals, and fed into the plant to produce the response. The unit bias provides an appropriate resting state, and the leaky integrator tonic allows short timescale adaptation of the resting state during periods of constant demand. In the human eye focusing system, for example, we would have blur being processed to generate signals for the ciliary muscles in the eye appropriate for the distance of the visual target. The system can equally well be regarded as a traditional control system (Levine, 1996), or as a fully dynamical network of leaky integrator neurons.

Simulating the evolution will involve working with a large number of copies of this model, each with four adjustable parameters/connection weights $\mathbf{W}(t) = \{WC(t), WP(t), WT(t), WB(t)\}$ where t is the time/age of the individual model measured in simulated years. These are learned by a simple on-line gradient descent algorithm that minimizes a cost function consisting of response error and regularization (smoothing) components which would be readily available to the system (Bullinaria & Riddell, 2001). Corresponding to the learnable weights, then, each instantiation of the model will have four variable learning rates/plasticities $\mathbf{P}(t) = \{PC(t), PP(t), PT(t), PB(t)\}$. The model will also have various other parameters (time constants, plant characteristics, feedback time delay, and so on) which we take to be the same for all instantiations, with values appropriate for human oculomotor control (Schor et al., 1992). Such a system that has evolved/learned a good set of weights will produce appropriate damped responses to arbitrary discontinuous output requirements such as steps, and smooth pursuit of arbitrary continuous output changes such as ramps (Bullinaria & Riddell, 2001).

For the purposes of this paper, I shall assume that all the learning rates in a given model vary with age in the same manner, and that this variation depends only on the genotype (innate parameters) of the individual, and not on the environment that the individual finds itself in. Naturally, it will be important to relax this condition in the future, but this means that we can write $\mathbf{P}(t) = s(t).\mathbf{P}(0)$, where $\mathbf{P}(0)$ are the four initial learning rates, and $s(t)$ is a simple age dependent scaling factor. Clearly, if there is no plasticity variation, $s(t) = 1$ for all t. A convenient parameterization is simply to take $s(t)$ to be piecewise linear with parameters $\mathbf{S} = \{s(t) : t = 1, \ldots, N\}$. The part of the model's genotype that varies between individuals thus represents the $8 + N$ parameters $\{\mathbf{W}(0), \mathbf{P}(0), \mathbf{S}\}$.

3 EVOLVING THE MODEL

Simulating an evolutionary process for our model involves taking a whole population of individual instantiations and allowing them to learn, procreate and die in a manner approximating these processes in real (living) systems. The genotype of each new individual will depend only on the genotypes of its two parents and random mutation. Then during their life each individual will learn from their environment how best to adjust their weights to perform most effectively. Eventually, perhaps after producing a number of children, each individual dies. Obviously, in nature, the ability of an individual to survive or reproduce will depend on a number of factors that are related in a complicated manner to that individual's performance on a range of related and unrelated tasks (food gathering, fighting, running, and so on). For the purposes of our simplified model, however, I shall consider it to be a sufficiently good approximation to assume a simple linear relation between our single task fitness function and the survival or procreation fitness. In fact, any monotonic relation should result in similar evolutionary trends, but it is easy to lose weak effects in the noise of the rather coarse simulations forced upon us by limited computational resources.

Given that, initially at least, we are aiming to replicate an effect that arises in human evolution, it seems appropriate here to follow a more natural approach to

procreation, mutation and survival than has been used in many evolutionary simulations in the past (e.g. in Belew & Mitchell, 1996). Rather than training each member of the whole population for a fixed time and picking the fittest to breed and form the next generation, our populations contain competing learning individuals of all ages, each with the potential for dying or procreation at each stage. During each simulated year, every individual learns from their own experience with a new randomly generated common environment (i.e. set of training/testing data) and has its fitness measured. Random pairs of individuals are then forced to compete, with the least fit dying (i.e. being removed from the population). Additionally, a random subset of the oldest individuals die of old age. The dead are replaced by children, each having one parent who is the fittest of a randomly chosen pair from the remaining population, who randomly chooses their mate from the rest of whole population. Each child inherits characteristics from both parents such that each innate free parameter is chosen at random somewhere between the values of its parents, with sufficient noise (or mutation) that there is a reasonable possibility of the parameter falling outside the range spanned by the parents. Ultimately, our simulations might benefit from more realistic encodings of the parameters, concepts such as recessive and dominant genes, learning and procreation costs, different inheritance and mutation details, different survival and procreation criteria, more restrictive mate selection regimes, offspring protection, different learning algorithms and fitness functions, and so on, but for the purposes of this paper, our simplified approach seems adequate.

4 SIMULATION RESULTS

A previous study (Bullinaria, 2001), employing a slightly more complex control system and a slightly simpler evolutionary regime, has already explored the Baldwin Effect, i.e. the interaction of learning and evolution (Baldwin, 1896; Belew & Mitchell, 1996), in models of the type considered here. This demonstrated explicitly how genetic assimilation of learned behaviour (i.e. learned parameter values) will occur automatically, without Lamarckian inheritance, to reduce the inherent costs of learning (e.g. periods of poor performance). However, even when a good set of innate parameters have evolved, a control system will still benefit from being plastic since that will allow it to fine tune its performance after a noisy procreation process and/or being born into an unpredictable environment. Many biological systems will also need plasticity to compensate for the changes (e.g. growing size) that naturally take place during their own maturation period. For the current study, such a maturation process was simulated by a simple output scale factor that varies linearly from 0.5 to 1.0 over the first ten years of life for each individual. (It turns out that the precise details of this variation are not crucial.) In humans this maturation might correspond to changes in inter-pupilliary distance for the eye rotation system, or changes in arm length for reaching or pointing. The important consequence is that the appropriate innate/newborn weights will not be the same as the adult values. The pattern of plasticities that evolve will allow the system to learn most efficiently how to optimize its weights throughout its life.

Unfortunately, limited computational resources allowed only a rather coarse simulation of the evolutionary process, but for an initial study it proved sufficient to have a fixed population size of only 100, with around 10 deaths per year due to competition, and around 4 individuals over 30 years old dying each year due to old age. (Such a system coded in C typically simulated around 20,000 years per CPU day on an average UNIX workstation.) The procreation and mutation parameters were chosen to speed the evolution as much as possible without introducing too much noise into the process. These evolutionary details were kept constant across all the simulations I shall now present.

Figure 2 shows the simulation results for a typical run of the basic system described above. First we see that the population means of the initial weights $\mathbf{W}(0)$ and learning rates $\mathbf{P}(0)$ quickly evolve to take on appropriate values. (Note the large variation between the learning rates that emerge for the different weights.) These lead to good values for the weights throughout the individuals' life. All the weights will need an initial fine tuning to remove the noise in the procreation process, then some weights (WC and WP) need to adjust during the maturation period, while others (WT and WB) need little further change. The plots of $WC(t)$ and $WT(t)$ for a typical evolved population show this quite clearly. The plots of the mean weights $\mathbf{W}(t)$ for the whole population show how they differ in magnitude and noise from the initial weights $\mathbf{W}(0)$. Finally, we see how the plasticity scale factor $s(t)$ varies with age t. In particular, we see that the plasticity falls drastically between birth and the end of the maturation period, thus confirming that critical periods for learning will arise as a natural consequence of evolution.

The results from the basic system naturally lead to the question of what happens if an individual needs to adapt or learn later in life, after the standard learning period is over. There is a traditional saying that

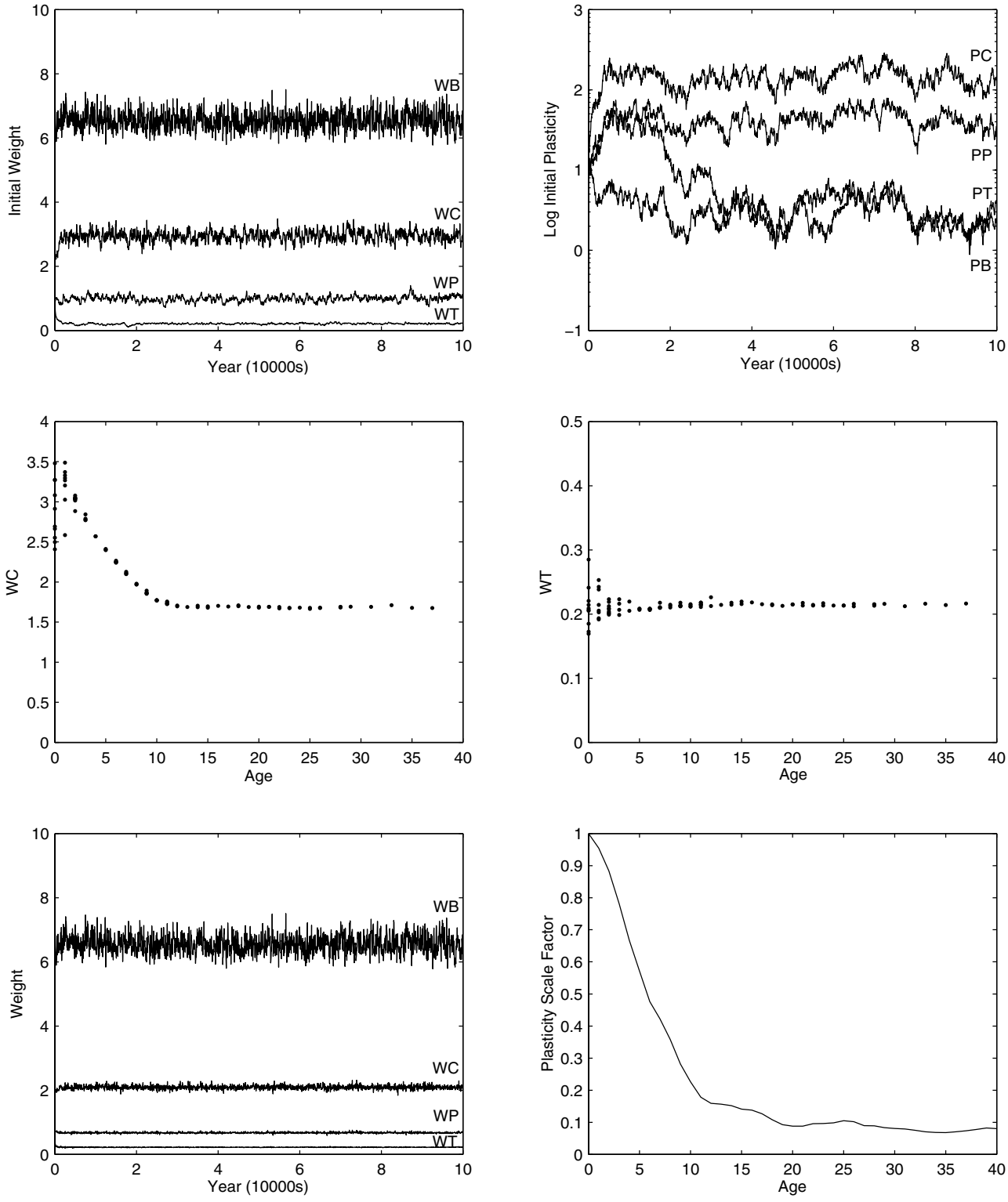


Figure 2: Evolution and learning in a typical simulation of the basic system. Individuals in the evolved population have plasticities that fall rapidly between birth and the end of their maturation period.

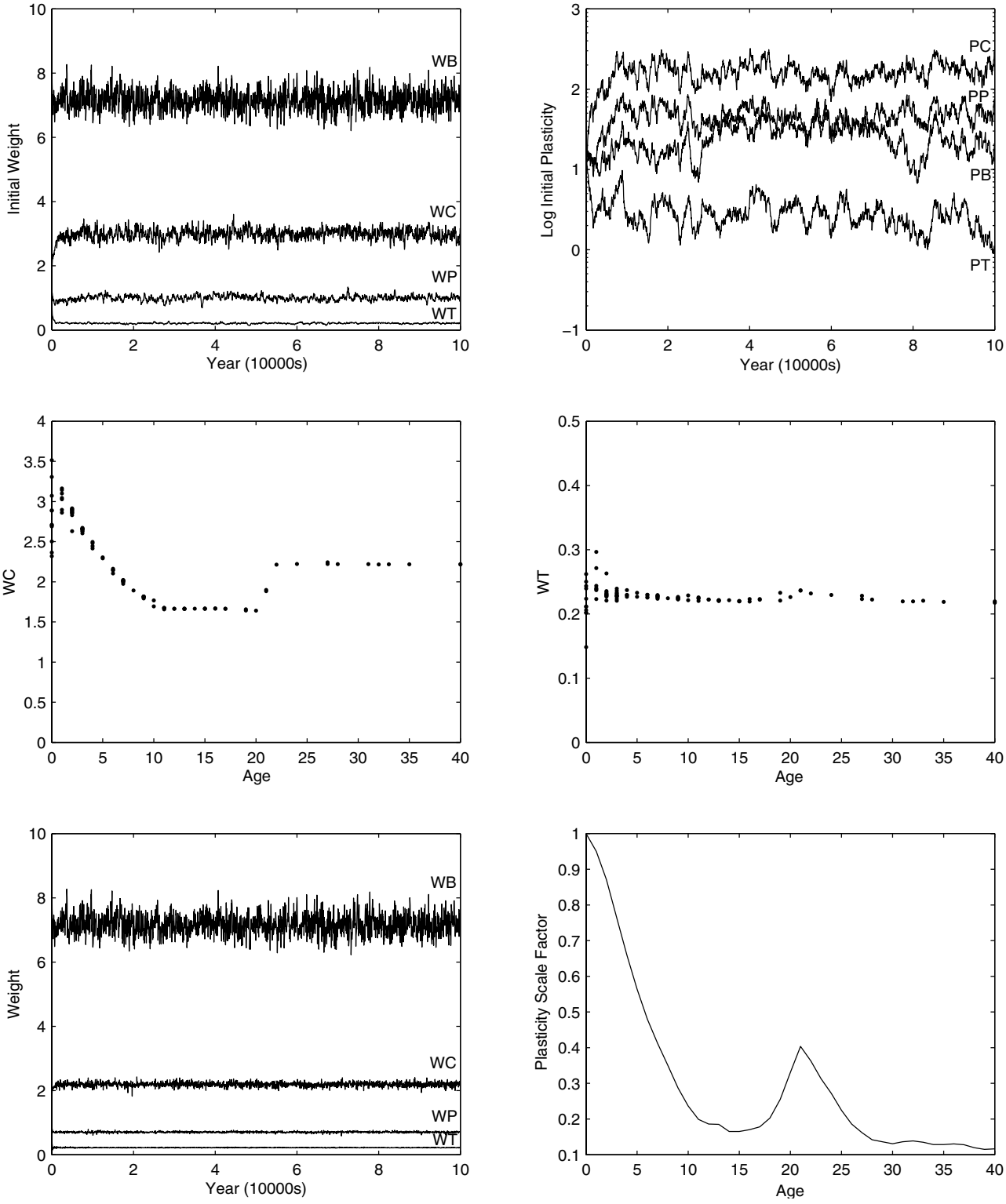


Figure 3: Evolution and learning in a typical simulation when late life adaptation is required. Individuals in the evolved population have plasticities appropriate for the learning or adaptation that is forced upon them.

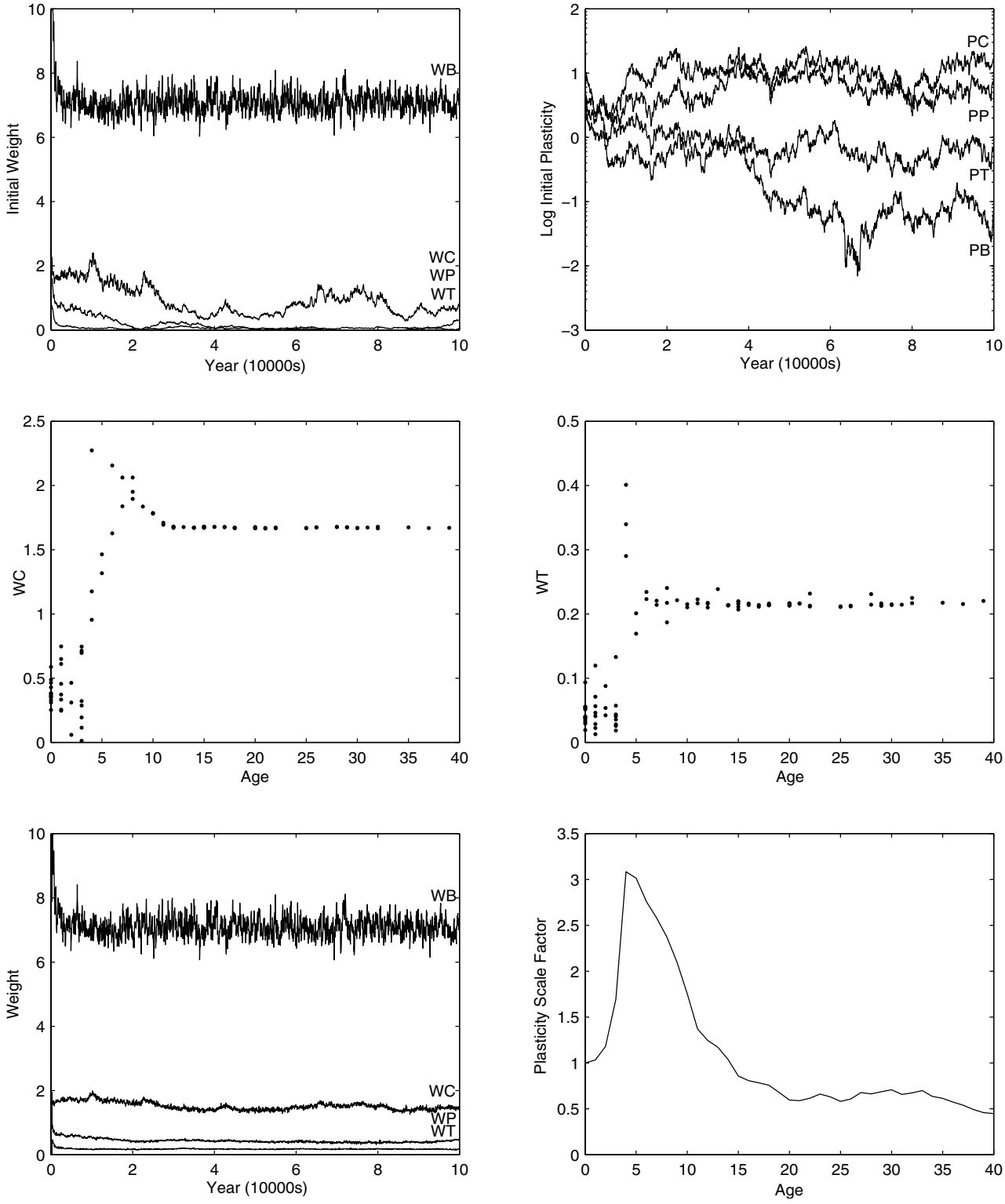


Figure 4: Evolution and learning in a typical simulation when there is a dependency on the development of lower level sub-systems. Individuals in the evolved population have a critical period for learning.

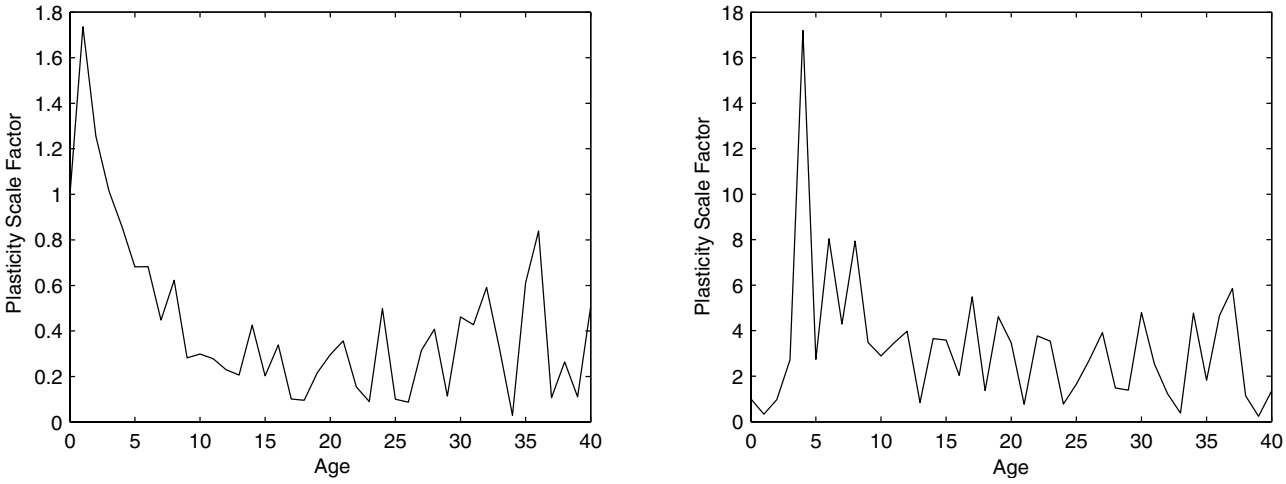


Figure 5: Typical plasticity scale factors arising from a different implementation to that used for Figures 2 and 4. The basic patterns are the same, but considerably noisier.

"old dogs cannot learn new tricks", but it seems unlikely that evolution would allow the plasticities to decay away to small values in situations where late life adaptation is regularly required. To introduce such a requirement, the basic model was modified so that there was a sudden step in the output scale factor from 1.0 to 0.75 at the age of 20. (Again it turns out that the precise details of this variation are not crucial.) There is no need to specify whether this variation corresponds to an internal factor (e.g. compensation for system damage or deterioration) or an external factor (e.g. adaptation to changes in the operating environment), as they will have the same effect. Obviously, the need for real late life adaptation will rarely be so predictable, but the consequences for our model will be similar, and the simplification makes it easier to interpret the results.

Figure 3 shows how this changes the simulation results from those of the basic model in Figure 2. The most obvious difference is in the plot of $WC(t)$ where we see the required step change at age 20 has been learned successfully. The plot of $s(t)$ shows the initial fall as before, but then a sharp rise to give the required increased plasticity at the age of 20. This gives us confidence that our evolutionary simulations really are picking up the requirement for plasticity, and not some confounding factor.

A final situation to consider, that regularly arises in human development, is when one level of processing relies on signals from another system. If the sub-system supplying those signals is not fully developed, it might be sensible to wait until it is before beginning to learn how to use the signals. For example, the adult eye rotation (vergence) system uses an image disparity signal, and humans have to wait until 12-16 weeks of age before this signal relatively suddenly becomes available. To simulate such an effect in our basic model, the error signal was replaced by low level noise for each individual until they reached three years of age.

Figure 4 shows how this affects the standard results of Figure 2. The changes here are rather clear. First, the initial/innate weights WC, WP and WT all drop to very low values, leaving the system with an appropriate constant output driven by the bias WB, and no interference from the noisy input signal. Naturally, the initial learning rates are also all very low, because learning from noise is not a good strategy, but they quickly rise to coincide with the onset of the input signal at the age of three. By the age of seven, the system has caught up with the performance levels of Figure 2. Once again our simplified evolutionary approach leads to a sensible pattern of plasticity variations.

5 SCALE FACTOR MUTATIONS

As with all modelling endeavours, it is important to test the robustness of the results with respect to the implementational details. Naturally, in this case it is the encoding of the plasticity scale factor $s(t)$ that we need to be particularly careful about. If each point $\{s(t) : t = 1, \ldots, N\}$ defining the piece-wise linear function were simply allowed to evolve in isolation in the same manner as the weights and learning rates, we would actually end up with the rather noisy results

shown in Figure 5.

The individual performance advantages that would keep the curves smooth, and reduce any unnecessary plasticity, are rather weak and get lost in the noise of our coarse simulations. This is particularly apparent after the age of about 10. The weakness is partly due to the error signals being relatively low after the maturation period is complete, and partly because it will be relatively unimportant if the fitness starts decreasing again after a number of children have already been produced, or if the majority of individuals normally die before reaching that age.

Fortunately, we can compensate for these limitations by variations of the plasticity scale factor mutations. First, we can prevent unnecessary plasticity (which will surely have an intrinsic cost in real systems) by allowing mutations which set random points $s(n)$ to zero. Then, it is unlikely in real systems to be efficient to have $s(t)$ varying wildly with age, so it is reasonable to encourage smoothness of $s(t)$ by allowing mutations which swap the values of random adjacent points $s(n)$ and $s(n+1)$. It was these simple variations that turned the noisy and inefficient results of Figure 5 into the smooth and efficient results of Figures 2, 3 and 4.

6 CONCLUSIONS

By simulating evolving populations of simple adaptable control systems we have seen that there is a natural propensity for the evolution of plasticities that vary sensibly with age, quite independently of any physical overheads of the plasticity. This is consistent with the well known "critical periods" of human brain development (Julesz & Kovacs, 1995). It is reasonable to expect that such an evolutionary approach will also be a profitable strategy for obtaining improved performance in adaptable systems for real world applications.

There are two competing effects at play. In order to survive in competition with fitter adults and/or a hostile environment, a newborn needs to adapt as quickly as possible to its environment. It also needs to adapt efficiently to its own maturation. Large plasticities will be beneficial for both. In adults, however, large plasticities can lead to an unstable learning system, in which unusual/extreme experiences can potentially result in a large shift of the systems' parameters with a serious reduction in overall fitness. Lower learning rates in this situation will allow smoother optimal parameter estimation and more consistently good responses in a varied environment. In this paper it has been demonstrated how a process of evolution by natural selection can result in a population of individual systems that deal with these conflicting requirements by having plasticities that vary appropriately with age under normal maturation, when late life adaptation is required, and when there is a dependence on the prior development of other sub-systems. We have also seen how appropriate changes to the implementational details (e.g. the plasticity scale factor mutations) can lead to vastly superior results.

In complex systems, such as the human brain, we can expect each of the various sub-systems to evolve appropriately for its own requirements, so there may well be no single global behaviour. The next stage of this work will be to develop and test larger scale and more realistic simulations of specific human sub-systems, and to explore explicitly how these ideas could be applied to the formulation of more efficient artificial adaptable systems for particular real world engineering applications.

References

Baldwin, J.M. (1896). A New Factor in Evolution. *The American Naturalist*, **30**, 441-451.

Belew, R.K. & Mitchell, M. (Eds) (1996). *Adaptive Individuals in Evolving Populations.* Reading, MA: Addison-Wesley.

Bullinaria, J.A. (2001). Exploring the Baldwin Effect in Evolving Adaptable Control Systems. In: R.F. French & J.P. Sougne (Eds), *Connectionist Models of Learning, Development and Evolution*, 231-242. London: Springer.

Bullinaria, J.A. & Riddell, P.M. (2001). Neural Network Control Systems that Learn to Perform Appropriately. *International Journal of Neural Systems*, **11**, 79-88.

Jacobs, R.A. (1988). Increased Rates Of Convergence Through Learning Rate Adaptation. *Neural Networks*, **1**, 295-307.

Julesz, B. & Kovacs, I. (Eds) (1995). *Maturational Windows and Adult Cortical Plasticity.* Reading, MA: Addison-Wesley.

Levine, W.S. (1996). *The Control Handbook.* London: CRC Press.

Schor, C.M., Alexander, J., Cormack, L. & Stevenson, S. (1992). Negative Feedback Control Model of Proximal Convergence and Accommodation. *Ophthalmic and Physiological Optics*, **12**, 307-318.

Yao, X. (1999). Evolving Artificial Neural Networks. *Proceedings of the IEEE*, **87**, 1423-1447.

Adaptive Control utilising Neural Swarming

Alex v.E. Conradie
Department of Chemical Engineering
University of Stellenbosch
South Africa
aconradi@ing.sun.ac.za

Risto Miikkulainen
Department of Computer Sciences
University of Texas at Austin
USA
risto@cs.utexas.edu

Christiaan Aldrich
Department of Chemical Engineering
University of Stellenbosch
South Africa
ca1@ing.sun.ac.za

Abstract

Process changes, such as flow disturbances and sensor noise, are common in the chemical and metallurgical industries. To maintain optimal performance, the controlling system has to adapt continuously to these changes. This is a difficult problem because the controller also has to perform well while it is adapting. The Adaptive Neural Swarming (ANS) method introduced in this paper satisfies these goals. Using an existing neural network controller as a starting point, ANS modifies the network weights through Particle Swarm Optimisation. The ANS method was tested in a real-world task of controlling a simulated non-linear bioreactor. ANS was able to adapt to process changes while simultaneously avoiding hard operating constraints. This way, ANS balances the need to adapt with the need to preserve generalisation, and constitutes a general tool for adapting neural network controllers on-line.

1. INTRODUCTION

The chemical and metallurgical industries face constant demands for greater economic return that requires increased production and greater product purity. Also, environmental concerns call for the use of minimal resources. By addressing these issues, intelligent control techniques add economic value to process plants.

A process' operating point (i.e., the process state) determines the product purity and production rate. The operating point thus has an intrinsic economic value. Control engineers select fixed operating points (i.e., set points) based on their economic value. Process changes, due to process disturbances and drifting dynamics, cause deviations from the set points, requiring corrective action. Optimal set points and effective corrective actions yield greater economic return. Typically, linear controllers (e.g., PID controllers) maintain the set points and provide corrective action to process changes (Seborg et al., 1989).

For example, a chemical reactor has an optimal operating temperature. This temperature determines the production rate that directly impacts on the economic return from the reactor. The control engineer selects this optimal temperature as a set point. A PID controller responds to process disturbances that affect the reactor temperature, by increasing or decreasing the cooling water flow rate, thereby maintaining the set point

PID controllers typically utilise both set points and fixed controller parameters. The PID controller parameters govern the corrective action (i.e., the control response) to process changes. There are three PID controller parameters: gain, integral and derivative. PID control's linear control structure is the industry standard, though not suited to non-linear processes.

Non-linear processes are common in the process industries. In such cases, PID controller parameters are optimal only over a limited operating region. Process changes may cause the operating point to stray far from the set point, whereupon PID controllers may implement sub-optimal corrective actions. Sub-optimal performance may be avoided only by adapting the controller parameters. As the set points largely determine the economic return, the set points must also adapt in response to process changes. Tracking the economic optimum therefore requires adapting both the controller parameters and the set points (Hrycej, 1997).

Effective generalisation and adaptability during process changes are essential to tracking a process' economic optimum. Generalisation tools, such as neural networks, are invaluable in creating non-linear controllers for non-linear processes. Non-linear controllers are near optimal over wider operating regions than possible with PID control (Conradie, 2000). Near optimal performance may be further improved by on-line adaptation of the neural network weights in response to process changes. Robust search techniques are required for effective on-line adaptation of neurocontroller weights.

This paper introduces an adaptive neurocontrol strategy, Adaptive Neural Swarming (ANS). A highly non-linear bioreactor benchmark is used in the control simulation. The bioreactor's dynamic behaviour is

changed continuously, which shifts the operating point with maximum economic return. ANS adapts an existing neurocontroller's weights to reap greater economic return from the changing bioreactor process. ANS emerges as an effective tool for adapting existing neural network strategies, resulting in enhanced performance.

Section 2 outlines basic notions in conventional adaptive control, which remain relevant to an advanced scheme such as ANS. Section 3 describes Adaptive Neural Swarming (ANS). Section 4 outlines the bioreactor case study. The paper concludes with an explanation of ANS' mechanism.

2. ADAPTIVE CONTROL METHODS

Control design requires a dynamic process model. Optimal control design is possible only if the process model is accurate. However, the model and the actual process are invariably mismatched. Also, exact knowledge of possible process changes is seldom available for control design purposes. Despite these shortcomings, robust control remains a control requirement. Generalisation is the ability of a controller to deliver near optimal performance, despite limited process knowledge during its design.

Generalisation may provide robust control, but optimal control is rarely ensured during the control design process. The designed controller frequently requires on-line refinements to the controller parameters and set points. Improved generalisation is difficult to impart on-line, as it involves reconciling past (i.e., design) and current process information into a single control strategy. For example, catalyst decay may cause the optimal temperature of a reactor to change over time. In contrast, adaptation changes controller parameters giving precedence to on-line process information. However, degraded performance may result should past process conditions return. A balance must thus be maintained between retaining generalisation imparted during design, while allowing adaptation to exploit changes in the process conditions (Hrycej, 1997).

On-line process information contains inaccuracies due to sensor noise and short-lived disturbances. Adapting controller parameters based on imperfect process information involves operational risk. The process may become unstable. On-line adaptation to control parameters faces numerous challenges: (1) Balancing the use of past and present process information, (2) Supervising process stability, (3) Implementing emergency procedures should the process become unsafe, due to on-line adaptation (Hrycej, 1997).

The following two sub-sections illustrate the aims of conventional methods for adapting controller parameters (section 2.1) and process set points (section 2.2). ANS has the same aims, though its methodology is dissimilar.

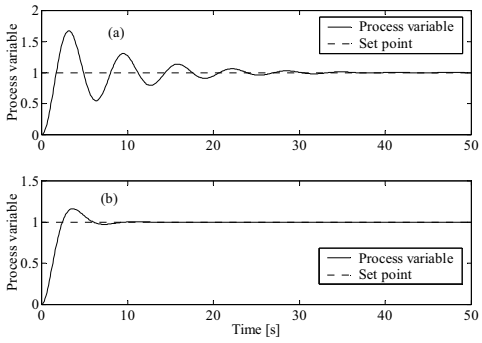


Figure 1: Objective of linear adaptive control. An oscillatory control response around the set point (a) is changed to a specified control response (b). The specified response settles sooner on the set point.

2.1 CONVENTIONAL ADAPTIVE CONTROL

An adaptive linear controller maintains a specified control response (i.e., corrective action) around a set point during process changes. For non-linear processes, a set of PID controller parameters can only maintain the specified control response for a limited range of process conditions. Process changes in non-linear processes may cause the control response to become oscillatory around the set point, as illustrated in figure 1a. Adaptive linear control tunes the PID controller parameters, which corrects the oscillatory response in figure 1a to the specified response in figure 1b. Conventional adaptive control relies on on-line process modelling (i.e., Model Reference Adaptive Control) and heuristic methods (i.e., Ziegler-Nichols) for adapting controller parameters (Ghanadan, 1990). ANS must also ensure that a specified control response is maintained.

2.2 EVOLUTIONARY OPERATION

Adaptive control does not change the set points that largely determine the economic return. Set points are selected during design based on an optimisation of dynamic model equations. The optimisation considers both economic return and controllability. However, process changes during operation may make the current set points economically sub-optimal.

Evolutionary operation (EVOP) challenges the use of constant set points in a continuously changing process. EVOP monitors the process and improves operation by changing the set points towards the economic optimum. EVOP makes a number of small set point changes that do not disrupt production. However, the set point changes need to be sufficiently large to discover potential improvements in the operating point. EVOP uses an experimental design to determine the number of set point change experiments. Pattern search methods use the experimental results to determine whether and in which direction the set points should be changed (Walters, 1991).

Consider figure 2, which graphs the economic return of

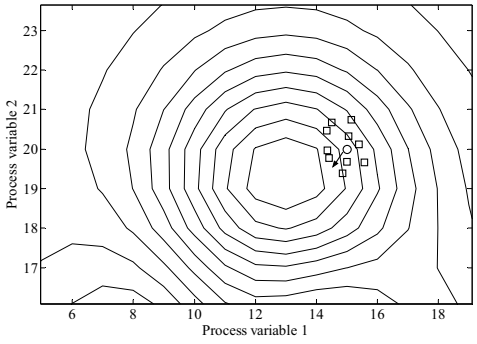


Figure 2: EVOP for a process with two process variables. The current set point (circular marker) is moved along the arrow's trajectory based on the economic return of each set point experiment (square markers). The process operation is thus improved.

a process that has two process variables. The contour lines represent operating points with similar economic returns. The circular marker represents the current set point, which is economically sub-optimal. The set points for both process variables should be reduced for optimal economic return. EVOP conducts a number of set point change experiments (represented by square markers) in the neighbourhood of the current set point. The economic return for each set point experiment is determined. In figure 2, three experiments have greater economic return than the current set point. EVOP adjusts the current set point in the direction of greater economic return. The process is repeated until optimal set points are found (Walters, 1991).

EVOP does not adapt the PID controller parameters for each of the set point experiments. As discussed in section 2.1, using the same controller parameters for all the set point experiments may give oscillatory responses. Poor control responses impact negatively the accurate determination of economic returns.

Adaptive control and EVOP may be combined in a two-step methodology to track a changing economic optimum. EVOP selects a number of set point experiments. An adaptive control method establishes a specified control response for each set point experiment. The economic evaluations for each experiment will consequently be comparable, whereupon EVOP adjusts the current set point. This cumbersome two-step process is repeated until the optimal set point is found. Ideally, a single on-line experiment (evaluation) should provide information on both the economic return and the control response.

3. ADAPTIVE NEURAL SWARMING

This section describes Adaptive Neural Swarming (ANS), which combines adaptive control and EVOP into a single comprehensive step. In ANS, both the economic return and the control response are combined into a single feedback signal. A local PSO uses this sparse reinforcement information to adapt the weights of existing neural network controllers towards greater economic return in response to a changing process.

3.1 NEURAL NETWORK STRUCTURES

Neurocontrollers may originate from various sources. Neural networks may be trained to mimic the control actions of existing PID controllers, thereby distributing the PID functionality over several neurons. Existing fuzzy logic systems may be converted to equivalent neural network architectures (Jong & Sun, 1993). Neurocontrollers are also developed utilising evolutionary reinforcement learning techniques (Conradie et al., 2000). Neural networks possess characteristics that are beneficial to an adaptive scheme, such as generalisation and graceful degradation.

Once a PID controller is adapted, the small number of control parameters prohibits effective generalisation to past process conditions. Neural network controllers are collections of neurons, with each neuron specifying the weights from the input layer (process states) to output layer (control actions). Neurocontroller parameters are the neural network weights. A neurocontroller that is equivalent to a PID controller, has additional degrees of freedom, owing to a larger number of controller parameters. During adaptation, a neural network's distributed functionality preserves greater generalisation to past process conditions. The need for effective generalisation justifies the use of neural networks.

Neural networks also exhibit graceful degradation. Graceful degradation allows small changes to the weights, without causing catastrophic control performance loss (S'euim & Clay, 1990). Process stability is preserved during adaptation.

These neural network characteristics are relied upon in a reinforcement learning framework, described below, to provide process stability and continued generalisation.

3.2 REINFORCEMENT LEARNING

Reinforcement learning (RL) automates the acquisition of on-line performance (i.e., feedback) information and the adaptation process. RL uses on-line performance evaluations to guide adaptation. RL improves controller performance without a need to specify how the control objectives should be reached (Kaelbling et al., 1996).

ANS maintains a population of possible neurocontroller solutions that serve as RL evaluations, similar to EVOP experiments. Each neurocontroller is evaluated individually over a number of sensor sample periods while interacting with a dynamic process as in figure 3. Initially, the process may be at an arbitrary operating point (state, s_t). The neurocontroller observes the current process operating point at sample, t, and selects a control action, a_t. The control action changes

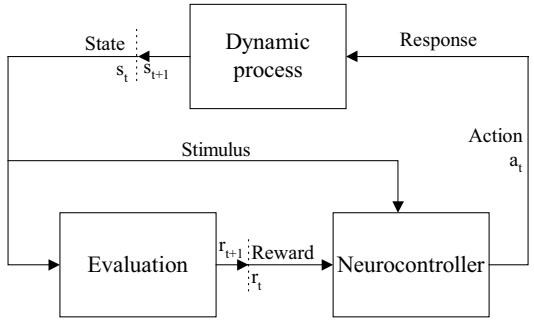


Figure 3: Reinforcement learning framework. A neurocontroller interacts with a dynamic process to learn an optimal control policy from cause-effect relationships.

the operating point to s_{t+1}. A reward, r_t, is assigned based on the economic value of this new operating point. The objective is to maximise the total reward over a series of control actions, while maintaining a specified control response. An optimisation algorithm adapts the neural network weights based the reward feedback from each evaluations.

ANS treats the population of neurocontrollers as a swarm, using a local particle swarm optimisation for adapting the weights of each neurocontroller.

3.3 PARTICLE SWARM OPTIMISATION

PSO is loosely based on the social behaviour of flocks of birds. A population of individuals is updated based on feedback evaluations, gathered from the collective experience of the swarm individuals (Shi & Eberhart, 1999). Equations 1 and 2 determine the velocity and position of the swarm in the solution space:

$$v_{id} := v_{id} + c_1 \cdot rand() \cdot (p_{id} - x_{id}) + c_2 \cdot rand() \cdot (p_{gd} - x_{id}) \quad (1)$$

$$x_{id} := x_{id} + v_{id} \quad (2)$$

where each particle, i, moves through the solution space with dimension, d. Each particles velocity vector, v_{id}, is dynamically adjusted according to the particle's own best experience, p_{id}, and that of the current best particle, p_{gd}, in the swarm. These two knowledge components are blended with each particle's current velocity vector to determine the next position of the particle as per equation 2 (Shi and Eberhart, 1999).

The best swarm particle is a beacon to a region of the solution space that may contain better optimisation solutions. Each particle searches the solution space along its unique trajectory for better solutions. Should a better solution be found, the new best swarm particle moves the swarm in a new direction. The momentum in each particle's current velocity provides some protection against convergence to a local optimum (Shi and Eberhart, 1999).

PSO has been utilised in tracking changing optima in function optimisation problems (Carlisle and Dozier, 2001; Angeline, 1997). PSO's success in these artificial domains motivates its use in complex real-world problems.

3.4 ON-LINE OPTIMISATION

ANS uses a local PSO search as the optimisation algorithm within a reinforcement learning framework. ANS thereby tracks the shifting economic optimum resulting from a changing process. Practical considerations for on-line use relate to the selection of swarm size, swarm initialisation, appropriate PSO parameters and duration of an RL evaluation.

Each on-line experiment is time and resource intensive, since no control improvements are possible during the evaluation phase. The number of reinforcement learning evaluations per PSO adaptation must therefore be minimal. However, the dimensionality of the control task constrains the minimum number of evaluations. More process information (i.e., more evaluations) is required during the evaluation phase, as the dimensionality of the control task increases. Otherwise, effective adaptation based on on-line feedback is not possible. Each neuron in a neurocontroller represents a partial solution to the control task. The number of neuron weights reflects the dimensionality of such partial solutions. For example, to effectively adapt neurons with 12 weights, an absolute minimum of 12 evaluations is required. The number of swarm neurocontrollers *(n)* is thereby selected based on the dimensionality of the control task, as reflected by the number of neuron weights.

In ANS, each swarm particle is an altered version of an existing neurocontroller. The initial swarm consists of the original (i.e., existing) neurocontroller and *(n-1)* altered neurocontrollers. Each altered neurocontroller is initialised with a small gaussian deviation from the existing neurocontroller weights. The maximum weight deviation is 3% from the each original weight, thereby altering the control policy only marginally. A neurocontroller swarm is thus initialised in a local region of the network weight solution space. This slight weight alteration determines the direction in which the swarm should move, without negatively effecting production and inducing process instability. On-line evaluation (experimentation) is thus limited to neighbouring solutions of an existing solution.

Each swarm neurocontroller is evaluated on-line for a limited number of sensor samples. A process' time constant is defined as the process response time to a step change in a control action. The process' time constant determines the number of sensor samples used in each evaluation. Equation 3 is the fitness evaluation that serves as feedback of each swarm neurocontroller's economic return:

$$Fitness = \int_{t_1}^{t_2} t \cdot P(t) \cdot dt - Penalty \quad (3)$$

where the evaluation is conducted for the number of

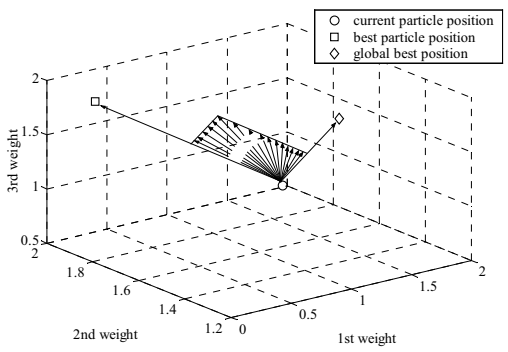


Figure 4: Possible adaptation trajectories of a weight vector based on the swarm's experience. The possible final position after adaptation lies in the plane formed by the arrow lines. The limited trajectories make the search exploitative.

samples between t_1 and t_2 and $P(t)$ is the instantaneous profit at time t.

A higher $P(t)$ for each sample reflects a higher economic return, which increases the fitness value. ANS thus searches for improved economic return. Equation 3 also dictates the specified control response. An ITAE (integral-time-absolute-error) control response has minimal oscillation, which is suited to numerous process control applications. Maximising the integral results in an ITAE control response. The fitness evaluation thus contains information regarding both the economic return and the control response. Also, should hard operating constraints exist for the process, a penalty is assigned should such operating constraints be approached during adaptation. This penalty reduces the fitness and solutions are therefore pursued only within the search boundaries.

An exploitative search preserves generalisation and reduces the risk of inducing process instability. A local (i.e., exploitative) PSO search is implemented by selecting a small inertia weight ($\omega = 0.4$) and the parameters c_1 and c_2 equal to 0.5 (conventionally 2.0) in equation 1. Each neurocontroller, i, adapts each weight, x_{id}, at position d in accordance with equation 2.

A neurocontroller may move only in a limited number of trajectories based on the swarm's experience. Consider a neurocontroller comprised of one neuron with 3 weights with no initial velocity. In figure 4, the circular marker represents the current weight vector. The dashed arrow lines illustrate the possible adaptation trajectories. These trajectories are determined by the global best neurocontroller (square marker) and the neurocontroller's own best experience (diamond marker). These limited trajectories make the search exploitative and are relevant to the optimisation objectives, since the directions are determined by the swarm's collective experience (Shi and Eberhart, 1999).

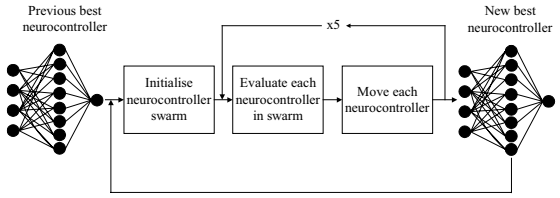


Figure 5: Adaptive neural swarming flow diagram. An effective neurocontroller is initialised into a swarm and adapted based on the evaluation of the swarm.

The local PSO search is run for five iterations, as illustrated in figure 5. The swarm is then re-initialised around the new best neurocontroller. Re-initialisation starts a new search in the neighbouring solution space of the new best neurocontroller. The search thus continues outside the space of the prior initialisations.

ANS was tested in a real-world bioreactor case study. The case study illustrates ANS' ability to adapt the neurocontroller weights towards greater economic return.

4. BIOREACTOR CASE STUDY

4.1 BIOREACTOR CONTROL PROBLEM

A bioreactor is a continuous stirred tank fermenter. It contains a biomass of micro-organisms that grow by consuming a nutrient substrate. The liquid substrate is fed continuously into the reactor, which also determines the reactor's liquid level (i.e., hold-up). The biomass is sold as the product. The bioreactor's dynamic behaviour is highly complex, non-linear and varies unpredictably. Also, the bioreactor process is difficult to quantify, due to unreliable biosensors and long sampling periods (Brengel and Seider, 1992).

Furthermore, the maximum bioreactor liquid level is a hard operating constraint. Should operation exceed the maximum level, the bioreactor is shut down and must then be restarted at great operational cost. However, the maximum instantaneous profit increases as operation approaches the hard level constraint. A trade-off between safety and the maximum economic return is required (Brengel and Seider, 1992).

The operating objective is to maximise the venture profit of the process on-line in response to process changes. This entails tracking the operating point with the maximum venture profit and ensuring acceptable control responses. The bioreactor may be simulated accurately and as such constitutes a benchmark for testing new adaptive methodologies without risking unsafe operation.

4.2 EXPERIMENTAL SET-UP

Typical process changes were simulated to mimic real-world bioreactor operation. The bioreactor's model was changed significantly by reducing the cell mass growth K (figure 6a) and increasing the substrate feed concentration S_F. The increased (i.e., off-set) S_F is also

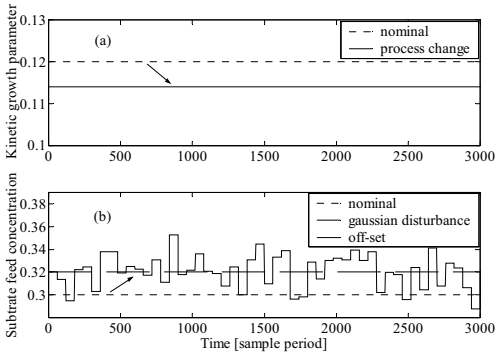


Figure 6: Process changes to the bioreactor. The arrows show the changes from the nominal process conditions (dashed line) for the growth parameter (a) and the substrate feed (b). The process is changed significantly.

disturbed with a gaussian distribution (figure 6b). In addition, the biosensors were inaccurate with a gaussian distribution around the correct reading.

Process search limits ensured that the process operation did not exceed the operation constraints. An adaptation scheme should never induce process shutdown by searching for operating points that are unsafe. The reactor level must remain below a high level alarm, which is a safety margin before bioreactor shutdown is initiated. The high level alarm was set at 5.95 [m] and bioreactor shutdown at 6.2 [m].

An optimal neurocontroller, with 12 neurons comprised of 7 weights each, was developed for the nominal process conditions using methods developed in prior work (Conradie et al, 2000). As discussed in section 3.4, ANS utilised this original neurocontroller to initialise a swarm of 10 neurocontrollers and each swarm neurocontroller was evaluated on-line over 20 sample periods. The inaccurate sensors and randomly changing process conditions make obtaining accurate feedback (i.e., evaluations) for ANS difficult. The ten evaluations, though not based on precise information, determined the direction and velocity of the neurocontroller swarm.

4.3 RESULTS

4.3.1 Adaptation efficiency of ANS

Figure 7 presents the instantaneous profit (IP) for the original neurocontroller and the ANS neurocontroller over a hundred day operating period. Figure 7 illustrates the effect of the process changes on the IP. The average instantaneous profit for the original neurocontroller was 55 [$/min]. As shown in table 1, this is well below the optimal profit of 96 [$/min] expected during design for the nominal process conditions. The original neurocontroller's IP is reduced due to sub-optimal generalisation to the process changes, though it was able to keep the process stable.

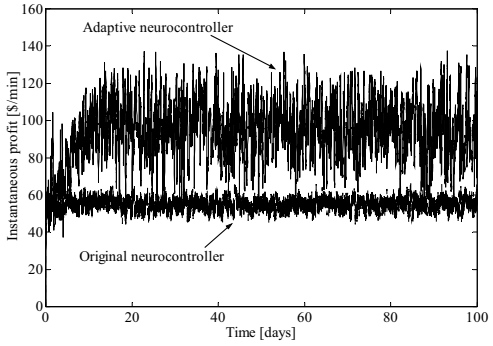


Figure 7: Instantaneous profit for the original and ANS neurocontrollers over 100 days of on-line operation. The adaptive neurocontroller garners greater economic return from the changing process than the original neurocontroller.

Table 1: Maximum IP for changing process conditions

Process condition	*Maximum profit[$/min]*
Nominal process conditions	96
K reduced, Off-set S_F	106
K reduced, Minimum S_F deviation	69
K reduced, Maximum S_F deviation	130

The original neurocontroller incurs an economic opportunity cost. Improved performance over 55 [$/min] is attainable with ANS. The average increase in S_F (i.e. off-set) presents an opportunity for greater venture profit. ANS achieves a substantially increased average profit of 94 [$/min] (figure 7), which is only slightly below the attainable 106 [$/min] possible for the increased S_F (table 1).

As seen in figure 7, the ANS neurocontroller has a larger IP standard deviation than the original neurocontroller. ANS tracks the optimal IP that is due to the gaussian disturbance in S_F. For high S_F values over extended periods (figure 6b, between samples 2000-2250), an IP of 120 [$/min] was attained, though a maximum of 130 [$/min] is attainable (table 1). For unusually low S_F values over extended periods, the swarm attained a minimal profit of 60 [$/min]. The optimal profit for this unfavourable process condition is 69 [$/min] (table 1). ANS thus approximates the changing optimal IP. A small difference remains, because S_F changes substantially over time periods that are too short for the swarm to adapt completely. The swarm is thus essentially tracking the moving average of S_F. Nevertheless, the IP for ANS control exceeds the highest IP for the original neurocontroller at all times (figure 7). ANS offers considerable benefits over the generalisation offered by the original neurocontroller.

4.3.2 Avoiding Hard Process Constraints

Figure 8 illustrates the swarm's ability to avoid the process search limits. Recall that the IP increases as the bioreactor level increases. The swarm neurocontrollers thus searched for control policies that increased the

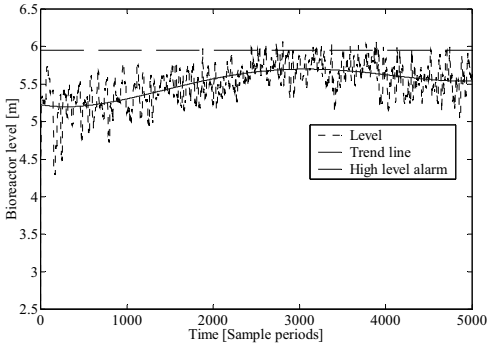


Figure 8: Avoiding the hard level constraint. The trend line (solid) illustrates the swarm moving away from high level alarm set at 5.95 [m]. Process shutdown is thereby avoided.

bioreactor level. Consequently, the swarm moved towards the high level alarm during on-line operation. The high level alarm of 5.95 [m] should never be exceeded; preserving the safety margin before bioreactor shutdown. A neurocontroller's fitness was penalised severely for exceeding the high level alarm. Such a penalised fitness was always lower than the fitness of a neurocontrol policy that remained within the search boundaries. Neurocontrollers, with a penalised fitness, no longer guided the swarm and the swarm moved away from the high level alarm. In Figure 8 at 3000 sample periods, the trend line indicates a move away from the high level alarm. Shutdown at a reactor level of 6.2 [m] was thus safely avoided in ANS' on-line search.

4.3.3 Neuron Weight Adaptations

Each neuron in a neurocontroller has a particular functionality that is a partial solution to the control task. A neuron's weight vector determines its functionality. The changes to a neuron's weight vector during adaptation, provides insight into how its functionality changed in response to the changing process conditions. Principal component analysis allows visualisation of neuron weight vectors and therefore neuron functionality.

Figure 9 is a principal component plot of the weight vector of each neuron in the swarm's current best neurocontroller. After each adaptation, all the neuron weight vectors for the best swarm neurocontroller were plotted in figure 9 as circular markers. The markers thus represent the history of adapted neuron functionalities.

In figure 9, the clusters indicate the different neuron functionalities that solve the control task. A cluster that is distributed over a larger region of the neuron weight space, had undergone a greater degree of on-line adaptation to its functionality. The extent of each neuron's adaptation is determined by the reigning process changes.

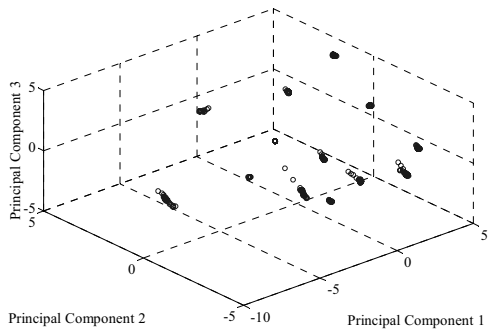


Figure 9: Principle component analysis of neuron functionality (85% variance explained). Circular markers represent neuron weight vectors. Each cluster represents the change in neuron functionality due to adaptation. The extent of each neuron's adaptation is determined by the reigning process changes.

5. DISCUSSION AND FUTURE WORK

ANS' exploitative search preserves the existing neurocontroller's generalisation. For the bioreactor, adaptation failure (i.e., shutdown) never occurs during extensive implementation. Also, instability is never induced in the control response. The bounded nature of each neuron cluster in figure 9 provides insight into how ANS preserves generalisation. Each neurocontroller retains memory of its best position (eq. 1) during the five iterations between initialisations. As the fitness landscape changes, the fitness value of a neurocontroller's best position is no longer valid. A neurocontroller's best position rather serves as an example of where previous good solutions have been found. Memory of past neurocontroller positions biases the search in the direction of good past solutions. This memory function preserves generalisation by considering both past and current process information in the search. Re-initialisation, which clears the swarm's memory, limits prolonged bias to past solutions. Without limiting memory of past solutions, a drifting optimum would be difficult to track.

ANS' search for optimal control policies in a changing process works as follows. Process changes affect each neuron's functionality differently. Some neurons consequently no longer contribute to optimal economic return. The functionality of such a neuron needs to be updated, while retaining information in its weight structure that is still valid.

Consider a neuron weight that is optimal once adapted to a fixed value, despite continued process changes. Such fixed weights correspond to process conditions that remain constant (e.g., fixed growth parameter). As described in section 3.4, the possible directions for adaptation are limited to the positional experiences of all the swarm neurocontrollers. In ANS, the swarm neurocontrollers align along such a fixed weight, preventing (as per eq. 1) the swarm from moving along that particular weight dimension. After several ANS

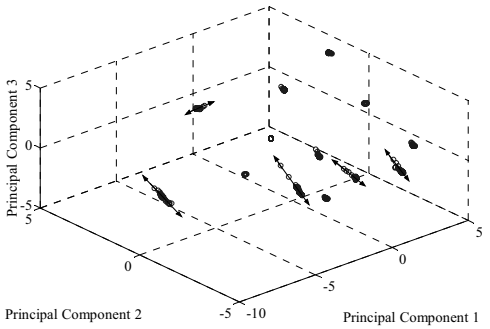


Figure 10: Arrow lines indicate the trajectories of neuron functionalities in response to common (re-occurring) process changes such as S_F. ANS implicitly takes advantage of common process changes, which facilitates effective adaptation.

iterations, only weights still relevant to improving the IP are implicitly changed. Re-occurring process changes (e.g., S_F) govern which specific neuron weights are continuously changed to track the economic return. The dimensionality of the search is thus somewhat reduced. Figure 10 is a copy of figure 9, except that adaptation trajectories are emphasised by drawing arrow lines through the clusters. Each neuron functionality (cluster) moves along a fixed trajectory in response to re-occurring process changes. ANS establishes these trajectories implicitly and exploits this swarm knowledge for greater economic return.

Future work will explicitly identify neuron functionalities that require adaptation. Such explicit knowledge may be used to further speed adaptation using fewer on-line evaluations. As ANS is a robust means for adapting neurocontrollers, it will be tested in other complex domains such as robotics and gaming.

6. CONCLUSIONS

Although neurocontrollers generalise their control actions in a changing process, such generalisation (though robust) may be economically sub-optimal. Adaptive Neural Swarming augments neurocontroller weights on-line, thereby garnering greater economic return from the changing process. ANS balances the need to adapt with the need to preserve generalisation. ANS also effectively avoids hard operating constraints during its on-line search. ANS implicitly identifies re-occurring process changes and uses this knowledge to speed adaptation. ANS is therefore a robust general tool for adapting of neural network controllers on-line. The greater economic return for the bioreactor case study suggests that the process industries would benefit significantly by implementing Adaptive Neural Swarming.

Acknowledgements

This work was supported in part by the South African Foundation for Research and Development, the Harry-Crossley Scholarship Fund, the National Science Foundation under grant IIS-0083776 and by the Texas Higher Education Coordinating Board under grant ARP-003658-476-2001.

References

Angeline, P.J., (1997). *Tracking Extrema in Dynamic Evironments*. Proceedings of the 6th Int. Conference on Evolutionary Programming, Vol. 1213: 335-345.

Brengel, D.D., and Seider, W.D., (1992). Coordinated design and control optimization of nonlinear processes. *Computers and Chemical Engineering* 16(9): 861-886.

Carlisle, A., and Dozier, G., (2000*). Adapting Particle Swarm Optimization to Dynamic Environments.* Proceedings ICAI 2000, Las Vegas, Vol. I: 429:434.

Conradie, A.v.E, (2000), *Neurocontroller development for nonlinear processes utilising evolutionary reinforcement learning*. M.S.c. thesis, Univeristy of Stellenbosch, South Africa.

Conradie, A., Nieuwoudt, I., and Aldrich, C., (2000). *Nonlinear neurocontroller development with evolutionary reinforcement learning*. 9th National Meeting of SAIChe, Secunda, South Africa.

Ghanadan, R., (1990). *Adaptive PID control of nonlinear systems*, M.Sc. thesis, University of Maryland, USA.

Hrycej, T., (1997). *Neurocontrol: Towards an industrial control methodology*. John Wiley & Sons (New York): 223-242.

Jang, J.S.R., and Sun, C.T., (1993). *Functional equivalence between radial basis function networks and fuzzy inference systems*. IEEE Transactions on Neural Networks, 4(1): 156-159.

Kaelbling, L.P., Littman, M.L., and Moore, A.W., (1996). *Reinforcement Learning: A Survey*. Journal of Artificial Intelligence Research, 4: 237-285.

Seborg, D.E., Edgar, T.F., and Mellicamp, D.A., (1989). *Process Dynamics and Cont*rol. John Wiley & Sons (New York).

S'equim, C.H., and Clay, R.D., (1990). *Fault tolerance in artificial neural networks*. Int'l Joint Conference Neural Networks (San Diego), vol. 1: 703-708.

Shi, Y., and Eberhart, R.C., (1999). *Empirical study of particle swarm optimization*. Proceedings of the 1999 Congress on Evolutionary Computation. IEEE Service Center (Piscataway, NJ): 1945-1950 .

Walters, F.H., (1991). *Sequential simplex optimization*. CRC Press (Florida): 55-60.

Particle Swarm Optimization Applied to the Atomic Cluster Optimization Problem

R.J.W. Hodgson

Physics Department
University of Ottawa
Ottawa, Ontario, Canada K1N 6N5
rhodgson@physics.uottawa.ca

Abstract

The Particle Swarm Optimization method has proven quite successful in treating a variety of applied problems. Here we further test its capabilities by studying its behavior when applied to a challenging problem, namely the search for energy conformations of atomic clusters. In its simplest form this is known as the Lennard-Jones Problem. Results are compared with those achieved using simple Genetic Algorithms.

1 INTRODUCTION

The problem of minimizing the potential energy function of clusters of atoms is generally known as the molecular conformation problem. Cluster sizes can range from a few atoms up to several hundred atoms. Physical and chemical characteristics vary with size. The determination of the global minima or ground states of these energy functions is of particular interest to researchers in chemistry, biology, physics and optimization methods. One particular class of these problems in molecular conformation is that where the interaction potential is the pure Lennard-Jones potential function. This turns out to be a very difficult problem to solve since the number of local minima has been estimated to increase exponentially with the number of atoms N (Tsai and Jordan, 1993; Stillinger, 1999). Studies have shown that at N = 13 there are 988 local minima, and for N = 98 the number grows to the order of 10^{40}. In spite of this formidable hurdle, success has been found in locating what are believed to be the global minima (ground states) for systems with N as large as 250 (Hartke, 1993 and 2001).

While Genetic Algorithms (GA) have played an important role in treating this problem (Barron et al. 1999; Deaven and Ho, 1995; Zeiri, 1995), the generic GA has not proven to be very successful beyond small numbers of atoms, requiring a large number of generations in order to locate the global minima. In place of it, modifications to the GA operations of crossover, and mutation have had to be made in order to bring about more rapidly converging sequences. These variations have been based on physical insight into the problem (Hartke, 2001) incorporating crossover and mutation procedures based on the physical geometry of the clusters. In addition, improvements have been achieved by incorporating local search algorithms into the GA as well (Deaven et al., 1996; Doye et al., 1999; Neisse & Mayne, 1996; Radcliffe and Surry, 1995; Wales and Doye, 1997). These investigations have shown that the problem can be treated using problem-specific GAs.

Here we investigate and compare the success of the Particle Swarm Optimization (PSO) method with a fairly generic GA when applied to the Lennard-Jones problem. There has already been some discussion of the similarities and differences between the PSO method and genetic algorithms (Eberhart and Shi, 1998; Angeline, 1998). The PSO method has proven to be successful in a variety of applications (Kennedy and Eberhart, 2001). Our objective is to see how well it can handle quite a challenging function in order to better understand its capabilities. Certainly the success achieved using special treatments tailored to the cluster problem will not be achieved using a simple approach such as the PSO. However, it's ability to seek out the global minimum in a function with a large number of local minima will be severely tested. If it can accomplish this test as well if not better than simple generic genetic algorithms, then it should merit recognition as a valuable tool for treating other global optimization problems.

2 THE LENNARD-JONES PROBLEM

The Lennard-Jones problem is concerned with determining the lowest-energy configuration of a cluster of neutral atoms interacting via the Lennard-Jones potential. The function to be minimized is the total energy (in reduced units) of the Lennard-Jones cluster as computed from

$$\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left(r_{ij}^{-12} - r_{ij}^{-6} \right)$$

where N is the number of atoms in the cluster, and r_{ij} represents the distance separating atom *i* and atom *j*. This problem has a long history (Hoare, 1979; Leary, 1997). It has served as a test-bed for a wide variety of optimization algorithms, primarily due to the exponentially increasing number of local minima. It is significant in the field of chemical physics for the insight gained by studying the structure of the clusters as the size increases. It also serves as a reasonably accurate mathematical model of a real physical system, namely that of low-temperature microclusters of heavy rare-gas atoms such as argon, krypton and xenon.

Much of the foundation for the study of the Lennard-Jones problem was laid by the well-known results of Northby (1987). He established multilayer icosahedral conformations as the dominant structural motif for the optimal microclusters and produced global optima for all $N \leq 147$. Other authors (e.g. Xue, 1994; Deaven et al. 1996; Leary & Doye, 1999) have since found new configurations having lower energies than those of Northby for some values of N. Many of these studies employed hybrid genetic algorithms with local search. More recently (Hartke, 2001) the PHENIX method, also based on a GA, has extended the values of N successfully treated up to 250. It is believed that the lowest-energy configurations have now been determined for all $N \leq 250$.

3 PARTICLE SWARM OPTIMIZATION

The Particle Swarm Optimization (PSO) method has evolved from a purely qualitative social optimization scheme to a truly numeric optimization scheme (Kennedy and Eberhart, 1995; Eberhart and Kennedy 1995). In its most applied form, the algorithm is designed to search for global optima in an n-dimensional search space of real numbers. An excellent review of the background and philosophy behind this method can be found in a recently published book (Kennedy and Eberhart 2001).

As in the more common Genetic Algorithm, a population of individuals is formed. Each individual in the PSO method is considered as a "particle" which is free to move about the search space. In the case of the Lennard-Jones problem each particle of the population is characterized by a set of 3N real numbers corresponding to the (x,y,z) positions of each atom. The fitness of a particle corresponds to the energy of the collection of N atoms, with the objective to make this energy as small as possible. In practice this energy is negative, corresponding to a 'bound' set of N atoms.

If particle i in the population, at time t, is represented as the vector $\vec{x}_i(t)$, then the change of position as the particle goes from one time step to the next is defined to be its velocity,

$$\vec{v}_i(t) = \vec{x}_i(t) - \vec{x}_i(t-1)$$

How this velocity changes at each time step is determined by the history of the particles past motion as well as that of its neighbours. This information is encapsulated in two parameters, the previous best position $\vec{p}_i$ for this individual particle, and the previous best position $\vec{p}_g$ for all those particles in the neighbourhood of this particular particle. We will define this neighbourhood later. The formulas to adjust the particle's velocity and position are then given by

$$\vec{v}_i(t) = w\vec{v}_i(t-1) + \varphi_1\left(\vec{p}_i - \vec{x}_i(t-1)\right) + \varphi_2\left(\vec{p}_g - \vec{x}_i(t-1)\right)$$

for the velocity, and

$$\vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t)$$

for the position. Here *w* is a weighting factor which is set to 0.9 at the beginning of the process, and decreases linearly to 0.4 at the end of the specified number of time steps (Shi and Eberhart 1998). The other two parameters, φ_1 and φ_2 are positive random numbers with upper bounds φ_1^{max} and φ_2^{max}.

The other constraint on the system requires that the velocity is limited within a certain range so that all the particles will not escape from the search area. This is defined by limiting each velocity to $\pm v_{max}$. Finally, the neighbourhood of each particle, mentioned above, refers to those particles which are adjacent in the population. For example a neighbourhood of 2 particles means that the particle in question has one neighbour on either side of it. A neighbourhood of 4 means that there are 2 particles on either side. It is understood that the population is considered to be a loop with the first and last particles in the population connected. The algorithm is then reasonably straightforward.

begin PSO
  g := 0 (generation counter)
  Initialize population P(g)
  Evaluate population P(g) (i.e. the cluster energy)
  while not done **do**
    g := g + 1
    Evaluate new velocities
    Evaluate new positions
    Determine local and global best
    Evaluate new population P(g)
  end while
end PSO

4 GENETIC ALGORITHMS

In order to do some sort of a comparison we have also carried out a number of calculations of the Lennard-Jones energies using a basic generic genetic algorithm, as well as a slightly more involved one. The first GA we refer to as BasicGA, which is described by the pseudocode shown below:

```
begin BasicGA
    g := 0 (generation counter)
    Initialize population P(g)
    Evaluate population P(g)
    while not done do
        g :=g+1
        Select P(g) from P(g-1)
        Crossover P(g)
        Mutate P(g)
        Evaluate P(g)
    end while
end BasicGA
```

This algorithm employed Roulette wheel selection, single-point crossover, and the mutation consisted of randomly modifying one parameter. The crossover probability was 0.8 and the mutation probability was 0.15.

In addition we ran a more general genetic algorithm based on the Genetic Algorithm Optimization Toolbox (GAOT) (Houck) which incorporated three different kinds of crossover functions and also three different mutation operators. Geometric selection was used, the crossover methods used were simple, arithmetic, and heuristic, and the mutation operators were boundary, uniform, and non-uniform (Michaleweiz 1992). In each generation two sets of parents were chosen for each crossover method, and four individuals were chosen for each of the mutation methods. Hence each generation is able to provide a wider search of the parameter space than in the case of the BasicGA above.

5 CALCULATIONS

In order to initially investigate the effect of the various parameters in the PSO calculation for the Lennard-Jones problem, we have focussed on a fairly simple case, that of the 8-atom cluster. As the best energies for clusters having up to 250 atoms are well known, we have examined the convergence of the iterations to the best known energy for this case (-19.82). Hence in the first part of this study we allow the iterations to proceed until the energy of the cluster (the best energy in the swarm) is within a small distance of the known value. Here we have required the value of this energy to converge to 4 significant figures.

The first calculations examined the effect of the neighbourhood size on the rate of convergence. We varied the size of the neighbourhood from 0 (meaning that all particles were in the neighbourhood) up to 14. The size of the swarm was kept at 30 particles. The maximum velocity was set at 0.2 and the bounds for $\varphi_1^{\max}$ and $\varphi_2^{\max}$ were both set at 2.0. If the swarm did not converge after 10,000 iterations, it was terminated. We repeated the runs 100 times and recorded the number of failures (i.e. the number of times that convergence was not achieved in the maximum allowed number of iterations), as well as the average of the number of iterations required for those cases where convergence was achieved. Figure 1 shows the behaviour of these two measures as a function of the neighbourhood size

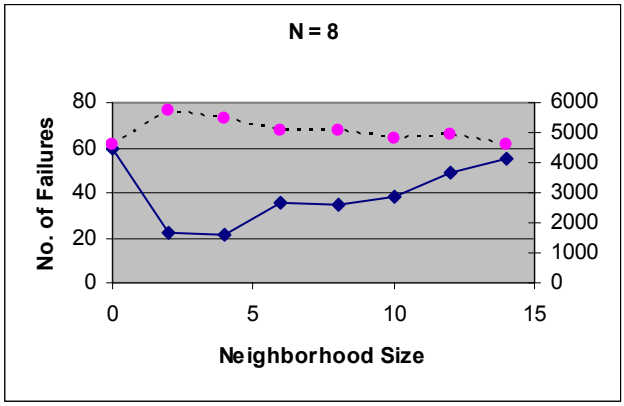


Figure 1: Variation with the neighbourhood size

The solid line in Figure 1 represents the number of failures, and the dashed line the average number of iterations. Here we see that the average number of iterations for the runs which converged did not vary

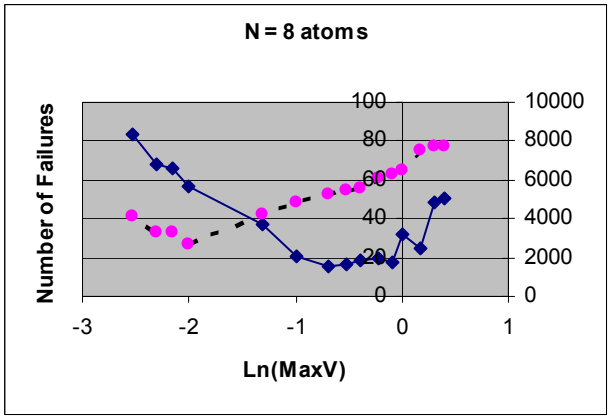


Figure 2: Variation with the log of Max V.

significantly from around 5,000. However there is a clear minimum for the number of failures when the neighbourhood size was either 2 or 4.

We proceeded next to investigate the effect of varying the value of the maximum velocity (MaxV).

The neighbourhood size was set to 4, and all other parameters were as above. The maximum velocity was varied between 0.003 and 2.5. With such a large range, we plotted the results as a function of the logarithm of the maximum velocity. The same two measures were used as above, and are shown in Figure 2.

The solid line, showing the number of failures for the 100 trials, shows a broad minimum with the number of failures to converge being less than or equal to 20 for MaxV in the range from 0.1 to 0.8. The number of average iterations has a minimum when MaxV equals 0.01. Hence again there is no common minimum. However, with the primary objective of achieving convergence, we have tended to focus more on the number of failures as opposed to the average number of iterations. The latter quantity does not vary over a large range. Selecting a maximum velocity anywhere in the range from 0.1 to 0.8 does not significantly affect the number of iterations to convergence. Hence for the next computation, we chose to select a value of 0.1 for MaxV.

The other variables which we looked at were those for the bounds φ_1^{max} and φ_2^{max}. Once again for the 8-atom cluster we took the neighbourhood size as 4, and the maximum velocity was set at 0.1. Setting the two bounds to be equal to each other, we obtained the results shown in Figure 3.

The value of 2.0 gives the best results in the sense that the number of failures is the least and the average number of iterations appears to increase with increasing φ_{max}.

While we recognize that there could be some dependence on the number of atoms and the nature of the energy surface in determining these results, we took these best parameters and ran calculations of cluster sizes for N varying from 4 to 15. We wanted to test the ability of the

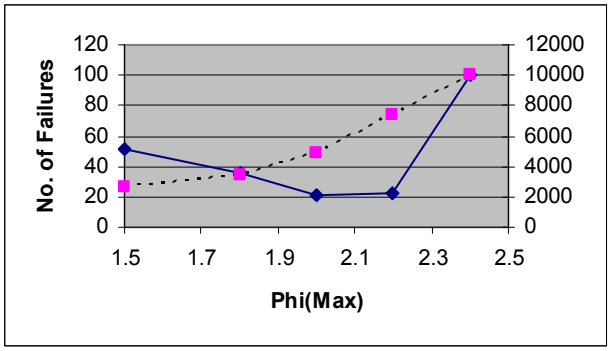


Figure 3: Variation with φ_{max}

PSO to achieve convergence over this range, and compare the results with fairly simple Genetic Algorithms applied to the same problem. In addition to looking for convergence, we also kept track of the number of function evaluations required for each run. In all cases we took the number of particles to be 30 and ran for 20,000 iterations. 100 trials were run for each value of N and the iterations were terminated if the energy matched the best known value for that value of N, up to 4 significant figures. If

Table 1.

		PSO			GA			
N	**Cluster Energy**	**No. Of Failures**	**Av. No. iterations**	**Av. No. Function Evaluations**	**No. Of Failures**	**Av. No. Generations**	**Av. No. Function Evaluations**	**Best Energy (GA)**
4	-6	0	2500	75008	57	11634	150016	
5	-9.104	0	5982	179487	95	16464	212581	
6	-12.71	96	7439	223177	100	20000	257500	-12.69
7	-16.51	52	8547	256424	100	20000	256720	-16.49
8	-19.82	29	9346	280388	100	20000	257482	-19.76
9	-24.11	67	10143	304293	100	20000	258916	-23.99
10	-28.42	91	9428	282846	100	20000	256877	-28.16
11	-32.77	95	11309	339294	100	20000	257706	-30.68
12	-37.97	96	10728	321855	100	20000	258336	-36.89
13	-44.33	100	20000	600000	100	20000	256950	-41.94
14	-47.84	99	16704	501120	100	20000	258426	-42.95
15	-52.32	97	16885	506550	100	20000	258444	-48.92

the run did not converge to this limit under the 20,000 iteration limit, then the run was counted as a failure.

Table 1 provides a summary of these results for N = 4 to 15. The second column is the known lowest energy for each of these cluster sizes (Leary, 1997). The next three columns show the PSO results: the number of failures to converge out of the 100 total trials; the average number of iterations required for those trials which did converge; and the average number of function evaluations used in the converged runs. The number of failures to converge is also graphed in Figure 4 as a function of N. Here two plots are shown, the solid line shows runs which were terminated after 10,000 iterations if convergence was not achieved, whereas the dashed line shows the same results for 20,000 iterations. It can be seen that there is not a great deal of difference between these two results.

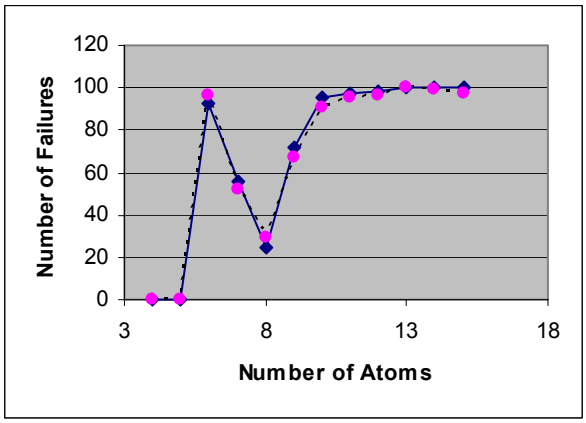


Fig. 4. Failures vs N

Table 1 also shows the results of runs using the modified GA. The basic GA was not successful in the sense that after 500,000 generations, convergence could not be achieved even for N = 4 atoms. Limiting the runs to 20,000 generations and using a population size of 30, the modified GA did find the known energies for N=4 and N=5, but was unsuccessful in 100 trials for the other values of N. The best energies achieved are shown in the table, and for the smaller values of N are relatively close to the known energies, indicating that allowing the GA to continue would likely achieve convergence.

The PSO method was able to achieve convergence in at least one of the trials for all values of N studied here, except for N = 13. In many cases only a few of the trials converged, seeming to indicate that the particle "swarm" could not get close enough to the fixed point in the set number of iterations.

6 SUMMARY AND CONCLUSIONS

The Particle Swarm Optimization method has been applied to a variety of areas since its introduction only a few years ago. As far as we are aware this is the first time that it has been applied to the energy conformation problem for atomic clusters. There has also been some other work on comparing the PSO with genetic algorithms (Eberhart and Shi 1998; Angeline 1998).

Genetic algorithms incorporate selection, crossover and mutation schemes in order to search the parameter space. In the PSO there is no specific selection process, however each individual carries with it a copy of its personal best value, which serves a somewhat similar role to that of a parent. The offspring of an individual is a function of this best value. The PSO is the only evolutionary algorithm that does not incorporate selection of the fittest.

The role of the crossover function in the GA is to select information from parents (usually two) to create offspring. In the PSO the influence on one particle by the others comes only in the value of the best position of the particles in the defined neighbourhood.

Mutations are an important aspect of the GA process in that they help to break out of the genetic sequence generated by the parents and offspring. However a common limitation of the GA is that as the population converges, the average fitness value becomes high so that mutations will usually result in a low-fitness chromosome which will be rejected by the selection process. The result is that the process may converge to a local optimum instead of finding the global one. There are variations which try to circumvent this, including the incorporation of local optimization methods.

Particle swarm uses a highly directional mutation operation as each individual's velocity vector is modified using a vector whose direction lies between the personal best and the neighbourhood best. As a consequence the PSO may have difficulties when the average local gradients point away from the global optima or are constantly changing.

Our calculations here have shown that the PSO is indeed an effective optimization method. Significantly better results have been found for the location of the global optimum energy value for a cluster of atoms interacting via the Lennard-Jones potential than for the case of a relatively generic GA. The potential energy surfaces for these problems are known to contain large numbers of local minima, often very close to the global minima for certain values of N. Hence it is not unexpected to find that many of the runs converge to one of these local minima.

The use of a low value of the maximum velocity in our calculations undoubtedly resulted in slow convergence in most cases, however it was somewhat necessary in the sense that the parameter values for the sizes of clusters studied here are in the range [-1, 1]. Small variations in these parameter values can shift the energy significantly, so that it is important to search over a relatively fine mesh.

The runs with the genetic algorithms quickly converged to a limit, and then tended to stay near that value for most of the subsequent generations. Here it would help to have

more flexible mutation and selection schemes to prevent this from happening.

Future work will focus on studying possible variations to the PSO which can help improve the success rate for this type of problem.

Acknowledgments

This investigation was supported in part by the National Science and Engineering Research Council of Canada (NSERC).

References

Angeline, P.J. "Particle Swarm - Evolutionary Optimization Versus Particle Swarm Optimization: Philosophy and Performance Differences", Lecture Notes in Computer Science **1447** (1998) 601-610.

Barron, C., Gomez, S., Romero, D. and Saaverdra, A. "A genetic algorithm for Lennard-Jones atomic clusters", App. Math. Letts. **12** (1999) 85-90.

Deaven, D.M. and Ho, K.M. "Molecular Geometry Optimization with a Genetic Algorithm", Phys. Rev. Letts. **75** (1995) 288-291.

Deaven, D.M., Tit, N., Morris, J.R. and Ho, K.M. "Structural optimization of Lennard-Jones clusters by a genetic algorithm", Chem. Phys. Letts. **256** (1996) 195-200.

Doye, J.P.K., Miller, M.A. and Wales, D.J., "The double-funnel energy landscape of the 38-atom Lennard-Jones cluster", J. Chem. Phys. **110** (1999) 6896-6906.

Eberhart, R.C. and Kennedy, J. "A new optimizer using particle swarm theory", Proc. Sixth Int. Symp. on Micro Machine and Human Science (Nagoya, Japan) (1995) 39-43.

Eberhat, R.C. and Shi, Y. "Comparison between Genetic Algorithms and Particle Swarm Optimization", Lecture Notes in Computer Science **1447** (1998) 611-618.

Hartke, B. "Global Geometry optimization of clusters using genetic algorithms", J. Phys. Chem. **97** (1993) 9973-9976

Hartke, B. "Global geometry optimization of atomic and molecular clusters by genetic algorithms", Proc. Of the Genetic and Evolutionary Computation Coference (GECCO) 2001, 1284-1291.

Hoare, M. " Structure and dynamics of simple microclusters", Adv. Chem. Phys. **40** (1979) 49-135.

Hoare, M.R. and Pal, P. Adv. Physics **20** (1971) 161.

Houck, C.R., Joines, J.A. and Kay, M.G. Genetic Algorithm Optimization Toolbox (GAOT) for Matlab. http://www.ie.ncsu.edu/mirage/GAToolBox/gaot/

Kennedy, J. "The particle swarm: social adaptation of knowledge", Proc. 1997 IEEE Int. Conf. on Evolutionary Computation, 303-308.

Kennedy, J. and Eberhart, R.C. "Particle swarm optimization", Proc. 1995 IEEE Int. Conf. on Neural Networks (Perth, Australia) 1942-1948.

Kennedy, J. and Eberhart, R.C. "Swarm Intelligence", Morgan Kaufmann, 2001.

Leary, R.H. "Global Optima of Lennard-Jones Clusters", J. Global Optim. **11** (1997) 35-53

Leary, R.H. and Doye, J.P.K. "Tetrahedral global minimum for the 98-atom Lennard-Jones cluster", Phys. Rev. **E 60** (1999) R6320-22.

Michalewicz, Z. (1992) "Genetic Algorithms + Data Structures = Evolution Programs" Springer-Verlag,.

Niesse, J.A. and Mayne, H.R. "Global geometry optimization of atomic clusters using modified genetic algorithm in space-fixed coordinates", J. Chem. Phys. **105** (1996) 4700-4706.

Northby, J.A., "Structure and binding of Lennard-Jones clusters: 13 ≤ N ≤ 147", J. Chem. Phys. **87** (1987) 6166-6177.

Peterson, C., Sommelius, O. and Soderberg, B. "Variational appraoch for minimizing Lennard-Jones energies", Phys. Rev. **E 53** (1996) 1725-1731.

Radcliffe, N.J. and Surry, P.D., "Formal Memetic Algorithms", Evolutionary Computing. AISB Workshop. Selected Papers. Springer-Verlag. 1994, pp.1-16. Berlin, Germany.

Shi, Y. and Eberhart, R.C. "Particle Swarm - Parameter Selection in Particle Swarm Optimization", Lecture Notes in Computer Science **1447** (1998) 591-600.

Stillinger, F.H., Phys. Rev. E **59**, 48 (1999).

Tsai, C.J. and Jordan, K.D., J. Phys. Chem. **97**, 11 227 (1993).

Wales, D.J. and Doye, J.P.K. "Global optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters up to 110 Atoms", J. Phys. Chem. A 101 (1997) 5111-5116.

Wille, L. and Vennik, J., "Computational Complexity of the ground-state determination of atomic clusters", J. Phys A 18 (1985) L419-L422.

Wolf, M.D. and Landman, U. "Genetic Algorithms for Structural Cluster Optimization", J. Phys. Chem. A 102 (1998) 6129-6137.

Xue, G. "Improvement on the Northby Algorithm for Molecular Conformation: Better Solutions", J. Global Optim. **4** (1994) 425-440.

Zeiri, Y., "Prediction of the lowest energy structure of clusters using a genetic algorithm", Phys. Rev. E **51**, R2769 (1995).

Option Valuation with Generalized Ant Programming

Christian Keber
University of Vienna
Department of Business Administration
Bruenner Strasse 72, A-1210
christian.keber@univie.ac.at

Matthias G. Schuster
University of Vienna
Department of Business Administration
Bruenner Strasse 72, A-1210
matthias.schuster@univie.ac.at

Abstract

For the valuation of American put options exact pricing formulas haven't as yet been derived We therefore determine analytical approximations for pricing such options by introducing the *Generalized Ant Programming* (GAP) approach applicable to all problems in which the search space of feasible solutions consists of computer programs. GAP is a new method inspired by Genetic Programming as well as by Ant Algorithms. Applying our GAP-approximations for the valuation of American put options on non-dividend paying stocks to experimental data as well as huge validation data sets we can show that our formulas deliver accurate results and outperform other formulas presented in the literature.

1 INTRODUCTION

In their seminal papers Black and Scholes (1973) and Merton (1973) derived an analytical solution for the valuation of European call options on stocks paying no dividends during the time to expiration. Merton additionally showed that premature exercising of American call options on this type of stocks is never optimal and that the valuation can also be made using the Black/Scholes–Merton method. If in the case of an American call option, dividends are paid, and these are known with certainty, its value can be obtained using the analytically exact pricing model of Roll (1977).

In comparison to American call options, premature exercise of American put options on non-dividend paying stocks may produce benefits, if the stock price falls below a certain, permanently variable critical value (*killing price*). As a result of this difference between the premature exercising of American call options and of American put options, ascertaining the optimal time for premature exercising, or the killing price, is a part of the problem to be solved, for which there was previously no exact model. Thus, the valuation of American put options is based on numerical procedures or analytical approximations. The best-known numerical procedures are the *lattice approach* of Cox, Ross, and Rubinstein (1979) and the *finite difference method* of Brennan and Schwartz (1977). However, getting accurate results by using numerical procedures normally requires long calculation time. A simple analytical approximation for the valuation of American put options on non-dividend paying stocks was presented by Johnson (1983). This simple analytical approximation, however, is not very accurate, so that many more ambitious analytical approximations have been developed. The best-known of these come from MacMillan (1986) and Geske and Johnson (1984).

MacMillan's analytical approximation for the valuation of American put options consists of raising the value of a suitable European put option by the value of the approximately calculated premature exercising of the option. This method forms the basis for analytical approximations used to evaluate a range of other American options, such as index options, currency options, and options on futures (see, e.g., Barone-Adesi and Whaley (1987)). Geske and Johnson's analytical approximation assumes that premature exercise is possible only at particular, discrete points in time. Accordingly, for each of these moments an option value is calculated, and, based on this, the value of the American put option is ascertained using the polynomial extrapolation method. This method treats American put options both with and without dividends. The MacMillan method, however, deals only with put options on stocks paying no dividend during the maturity of the options, although it was extended by Barone-Adesi and Whaley (1988) as well as Fischer (1993) to

include the dividend paying case. It is characteristic of most approximations for the valuation of American put options found in the literature that they approximate either the stochastic stock price process, or the partial differential equation (with the appropriate boundaries) which implicitly describes the option price. In contrast, Geske and Johnson (1984) as well as Kim (1990) formulate a valuation equation which represents an exact solution of the partial differential equation, and then solve it either by analytical approximation or by numerical techniques.

During the last few decades an increasing number of researchers of various disciplines has been impressed by the problem-solving power of nature. They developed optimization algorithms and heuristics based on the imitation and simulation of admirable natural phenomena. For example, *Artificial Neural Networks* imitate the principle of human brains and *Evolutionary Algorithms* make use of the *Darwinian principle* of the *survival–of–the–fittest*. Both methodologies are used to solve problems which are normally not amenable to traditional solution techniques and have been applied to option pricing problems. For instance, Hutchinson, Lo, and Poggio (1994) employ an *artificial neural network* for the valuation of European options whereas Chen, Lee, and Yeh (1999), Chidambaran, Lee, and Trigueros (2000), and Keber (2000) use Genetic Programming to solve option pricing problems.

A further example and one of the most recent developments of nature-based solution techniques is the *Ant Colony Optimization* meta-heuristic. *Ant Algorithms* originally introduced by Dorigo and colleagues (Dorigo (1992) and Dorigo, Maniezzo, and Colorni (1991)) as a multi-agent approach to difficult combinatorial optimization problems like the travelling salesman problem (TSP) are inspired by the foraging behaviour of real ant colonies, in particular, how ants can find shortest paths between two points. Real ant colonies are capable of solving such problems using *collective behaviour* and indirect communication via a chemical substance called *pheromone* deposited on the ground. It is hardly surprising that for the first time ant algorithms have been applied to the travelling salesman problem because the analogy between the real ants' problem and the TSP is obvious. In the meantime, ant-based algorithms have been applied successfully to a broad field of combinatorial optimization problems (see, e.g., Dorigo, Caro, and Gambardella (1999)).

In this contribution we develop the *Generalized Ant Programming* (GAP) approach as a new variant of ant algorithms and apply the proposed method to the option valuation problem. GAP enables computers to solve problems without being explicitly programmed. It is applicable to all problems in which the search space of feasible solutions consists of computer programs. GAP works by using artificial ants to automatically generate computer programs. As an acid test for GAP we derive analytical approximations for the valuation of American put options on non-dividend paying stocks. We focus on this problem for several reasons. Firstly, analytical exact solutions for pricing American puts have not as yet been derived. Secondly, testing new techniques should not be based on simple problems because any assessment of the proposed method would be open to criticism. Furthermore, our results have to be compared with other approximations presented in the literature. This can be easily done by using the most frequently quoted approximations mentioned above. Using experimental data as well as huge validation data sets we can show that the GAP based formulas for the valuation of American put options on non-dividend paying stocks deliver accurate approximation results and outperform other approximations presented in the literature.

In the second section we focus on the concept of the Generalized Ant Programming approach. The third section presents the GAP-approximations for the valuation of American put options on non-dividend paying stocks. In the fourth section we show the experimental results. The contribution concludes with a summary.

2 GENERALIZED ANT PROGRAMMING

2.1 INTRODUCTION

The Generalized Ant Programming (GAP) approach is a new method inspired by the Genetic Programming approach introduced by Koza (1992) as well as by Ant Algorithms originally presented by Dorigo (1992) as a multi-agent approach to difficult combinatorial optimization problems like TSP. GAP is an approach designed to generate computer programs by simulating the behaviour of real ant colonies. When travelling real ants deposit pheromone on the ground which influence the choices they make. Ants tend to choose steps marked by strong pheromone concentrations. Pheromone trails can be seen as "public information" which is modified by ants to reflect their experience while solving a problem, e.g., to find shortest paths between the nest and food sources. The quantity of pheromone left by an ant depends on the amount of food found. Within a given interval of time, shorter paths can be travelled more often, which causes a stronger pheromone concentration. In return, this

increases the probability of the path to be chosen.

2.2 METHODOLOGY

Generalized Ant Programming is an algorithmic framework which enables computers to solve problems without being explicitly programmed. It is applicable to all problems in which the search space of feasible solutions consists of computer programs. GAP works by using artificial ants to automatically generate computer programs. Similar to real ants, the artificial ants explore a search space now representing the set of all feasible computer programs which we describe as paths through a graph. The pheromone amount deposited by an artificial ant depends on the quality of the solution found. In other words, it depends on which path (computer program) was chosen. The quality of a path is measured using the corresponding computer program as an "input parameter" to an "algorithmic regression problem". These transpositions lead to our proposed GAP approach which we describe in a more detailed way in the next few paragraphs.

Computer programs are usually based on a well defined programming language. In our GAP–application we therefore use a programming language $\mathcal{L}$ specified by the context-free grammar $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \mathcal{R}, \mathcal{S})$, see, e.g., Aho and Ullmann (1972). $\mathcal{L}(\mathcal{G})$ is to be seen simply as the set of all analytical expressions which can be produced from a start symbol $\mathcal{S}$ under application of *substitution rules* $\mathcal{R}$, a finite set of *non-terminal* symbols $\mathcal{N}$, and a finite set or vocabulary of *terminal symbols* $\mathcal{T}$. Thus,

$$\mathcal{L} = \{p \mid \mathcal{S} \Longrightarrow p \ \wedge \ p \in \mathcal{T}^*\} \tag{1}$$

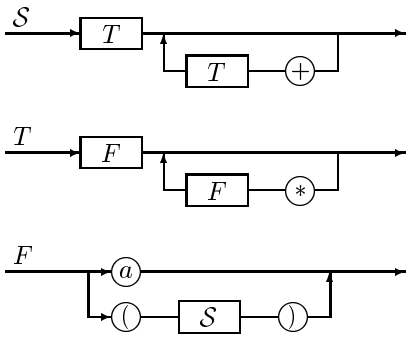
where $\mathcal{T}^*$ represents the set of all analytical expressions which can be produced from the symbols of the vocabulary $\mathcal{T}$. Using the grammar $\mathcal{G}$ a derivation of an analytical expression $p \in \mathcal{L}$ consists of a sequence $\mathfrak{t}_1, \mathfrak{t}_2, \ldots, \mathfrak{t}_\mathfrak{p}$ of terminal symbols and the corresponding derivation steps (productions) $\mathfrak{t}_i \to \mathfrak{t}_{i+1}$ (for $i = 1, \ldots, \mathfrak{p} - 1$). This derivation is denoted by

$$\mathcal{S} \underset{\mathcal{G}}{\overset{*}{\Longrightarrow}} p. \tag{2}$$

To take a simple example, assume

$$\begin{aligned}\mathcal{G} = (&\mathcal{N} = \{\mathcal{S}, T, F\},\\ &\mathcal{T} = \{a, +, *, (,)\},\\ &\mathcal{R} = \{\mathcal{S} \to \mathcal{S} + T | T, T \to T * F | F, F \to (\mathcal{S}) | a\},\\ &\mathcal{S})\end{aligned}$$

and let us express this grammar in an equivalent graphical representation (*syntax diagram*).



Each derivation in this grammar represents a simple arithmetic expression including the symbols $a, +, *, (,$ and $)$ and can be interpreted as a path through the syntax diagram. An example of a derivation in our simple grammar would be

$$\begin{aligned}\mathcal{S} &\Rightarrow \mathcal{S} + T \Rightarrow T + T \Rightarrow F + T \Rightarrow a + T\\ &\Rightarrow a + T * F \Rightarrow a + F * F \Rightarrow a + a * F \Rightarrow a + a * a\end{aligned}$$

In the sense of GAP, $\mathcal{L}$ is the *search space* of all potential analytical expressions to be generated, $p \in \mathcal{L}$ is a *path* which can be visited by ants, and $\mathcal{J}(t) \subset \mathcal{L}$ is a set of paths already visited at time t. Furthermore, each path $p \in \mathcal{L}$ consists of a sequence of terminal symbols $\mathfrak{t}_1, \mathfrak{t}_2, \ldots, \mathfrak{t}_\mathfrak{p}$ and the corresponding derivation steps $\mathfrak{t}_i \to \mathfrak{t}_{i+1}$ (for $i = 1, \ldots, \mathfrak{p} - 1$).

In GAP each path $p_i \in \mathcal{J}(t)$ can be seen as a derivation

$$\mathcal{S} \underset{\mathcal{P}_i}{\overset{*}{\Longrightarrow}} p_i, \tag{3}$$

where $\mathcal{P}_i \subset \mathcal{G}$. While walking each ant forms a new path p' which is a derivation based on $\mathcal{P}' = \cup_{i=1}^{n} \mathcal{P}_i \subset \mathcal{G}$,

$$\mathcal{S} \underset{\mathcal{P}'}{\overset{*}{\Longrightarrow}} p', \tag{4}$$

and where all derivation steps contained in p' are selected according to the pheromone amounts of the corresponding paths p_i. In the proposed GAP-application the pheromone trail is put on whole paths implying that the derivation steps describing a path are equally weighted. This is just for simplification and can be extended to different pheromone amounts along a path. We are going to present this extension in one of our next papers.

The amount of pheromone trail on path p at time t is given by

$$\tau_p(t) = (1 - \rho) \cdot \tau_p(t) + \Delta\tau_p(t), \tag{5}$$

where $0 < \rho \leq 1$ is the coefficient representing pheromone evaporation, and

$$\Delta\tau_p(t) = \sum_{k=1}^{\mathcal{K}} \Delta\tau_p^k(t)$$

is the pheromone increase obtained by cumulating the contributions $\Delta\tau_p^k(t)$ of each ant $k = 1, \ldots, \mathcal{K}$. In other words, this is the amount of pheromone deposited on path p by the k^{th} ant at time t. This quantity of pheromone trail is given by

$$\Delta\tau_p^k(t) = \begin{cases} Q \cdot L_k(t,p) & \text{if } k^{\text{th}} \text{ ant takes path } p \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

where Q is a constant and $L_k(t,p)$ is the value of the *objective function* obtained by ant k at time t. As GAP is similar to the Genetic Programming approach each path $p \in \mathcal{L}$ represents a computer programm (or an analytical expression) and can therefore be seen as a function $p\colon \mathcal{E} \to \mathcal{A}$ transforming input data $\mathcal{E}$ into a solution or output data $\mathcal{A}$. Accordingly, the function $L_k : \mathcal{A} \to I\!R$ has to be defined in a way that it awards higher values to those paths (computer programs) which represent a good solution to the task in hand, and lower values to less suitable paths (computer programs). The value of the objective function is measured using a representative set of test records $\mathcal{E}_i$, for $i = 1, \ldots, \mathcal{D}$. If these input data are processed using the computer program $p \in \mathcal{J}(t)$, the result will be the output data $\mathcal{A}_i$, which can then be compared to the target output data $\mathcal{A}_i^S$. Using a deviation function $\delta(\mathcal{A}_i, \mathcal{A}_i^S)$ in a way that it delivers higher values the more the output data differ from the target output data, the aggregated deviation is given by

$$F(p) = \sum_{i=1}^{\mathcal{D}} \delta(\mathcal{A}_i, \mathcal{A}_i^S) \quad \text{with} \quad \mathcal{A}_i = p(\mathcal{E}_i). \quad (7)$$

The objective function $L_k(t,p)$ can be formulated as

$$L_k(t,p) = \frac{1}{1 + F(p)} \quad (8)$$

so that the values of the objective function lie between zero and one, and larger values represent better paths (computer programs).

For the (new) path $p' \in \mathcal{L}(\mathcal{P}')$ being built by the k^{th} ant, see (4), the probability of selecting derivation steps describing old paths $p_i \in \mathcal{J}(t)$ is given as

$$P_{p_i}^k(t) = \begin{cases} \dfrac{[\tau_{p_i}(t)]^\alpha \cdot [\eta_{p_i}]^\beta}{\sum_{\pi \in \mathcal{P}'_k(t)} [\tau_\pi(t)]^\alpha \cdot [\eta_\pi]^\beta} & \text{if } p_i \in \mathcal{P}'_k(t) \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

where $\mathcal{P}'_k(t) \subseteq \mathcal{P}'(t)$ is the set of derivation steps of path p_i that the k^{th} ant has not visited yet. η_{p_i} is a heuristic value of including derivation steps of p_i. The parameters α and β control the relative importance of pheromone trail versus visibility.

Using the above definitions, GAP can be outlined by the following (pseudo) computer program.

```
[ 0] program Generalized AntProgramming;
[ 1]   t = 0;
[ 2]   Init α, β, ρ, Q, J(t), τ_p(t);
[ 3]   repeat
[ 4]     t = t + 1;
[ 5]     for each ant k do
[ 6]       Build a path p' according to (4) and (9);
[ 7]       Calculate L_k(t, p') using (8);
[ 8]       J(t) = J(t − 1) ∪ p';
[ 9]       end;
[10]     Save the best solution found so far;
[11]     Update trail levels τ_p(t) according to (5);
[12]     Shrink J(t);
[13]     Perform global shaking on τ_p(t);
[14]   until termination;
[15] end.
```

While most of the programming steps are already discussed, programming step [12], Shrink $\mathcal{J}(t)$, is to be seen in conjunction with step [8], $\mathcal{J}(t) = \mathcal{J}(t-1) \cup p'$. Executing [8] repeatedly implies that the set of paths already visited, $\mathcal{J}(t)$, becomes bigger and bigger or could go to infinity in worst case which is highly undesirable. On the other hand, $\mathcal{J}(t)$ also contains paths where pheromone trails are completely evaporated so that they can be excluded in further exploration, i.e., when ants choose a new path. Hence, in step [12] we shrink $\mathcal{J}(t)$ accordingly. A second step not yet mentioned is the global shaking procedure in [13]. In (1) we choose a specification (of our language) which is of finite size, although the language being specified is not finite. Hence, GAP is comparable to dynamic problems such as the TSP with the insertion or deletion of cities, see, e.g., Bonabeau, Dorigo, and Theraulaz (1999) and Guntsch, Middendorf, and Schmeck (2001). If, in GAP, the pheromone amount on a derivation step (or path) becomes much higher than all others, this step (or path) will almost certainly always be chosen. This would be fine in a static model but is a problem in GAP because it prevents ants from taking new derivation steps. Similar to dynamic approaches we therefore apply the "shaking technique" to normalise the pheromone levels. The formula used in our application is a logarithmic one and is given by

$$\tau_p(t) = \tau_p^0 \cdot \left[1 + \ln\left(\frac{\tau_p(t)}{\tau_p^0}\right)\right], \quad (10)$$

where τ_p^0 is the minimum value for $\tau_p(t)$, forced by the algorithm, so that $\tau_p(t) \geq \tau_p^0$ (Eyckelhof (2001) and Stützle and Hoos (2000)). In our GAP-application we use the initial value as the lower boundary.

2.3 RELATED WORK

To our knowledge, the first attempt using ants for automatic programming comes from Roux and Fonlupt (2000). At first glance one tends to believe that their approach can solve symbolic regression problems. But on a closer view it is doubtful whether the approach provides accurate approximation results. Hence, we have retraced and implemented their method. Testing the approach on several problems we have got poor approximation results. This is in accordance with the results presented in their own paper which, from our point of view, are not very promising. Furthermore, in the present version of the proposed method the authors focus only on symbolic regression problems. In comparison, our approach can be used for general "*algorithmic* regression problems". Thus, we refer to our approach as *Generalized* Ant Programming.

3 APPLICATION

3.1 EXPERIMENTAL DESIGN

By applying the GAP approach to option pricing we have to specify some parameters. For the aggregated deviations used in the objective function (8) we used a randomly generated training sample of 1,000 American put options on non-dividend paying stocks described by the tuple $\langle P_0, S_0, X, r, T, \sigma\rangle$ where $S_0 > S^*$. S^* represents the killing price calculated using MacMillan's (1986) procedure, P_0 refers to the "exact" value of an American put option on a non-dividend paying stock at $t = 0$, S_0 denotes the stock price at $t = 0$, X is the exercise price, r represents the annual continuous risk-free interest rate (in %), T is the time to expiration (in years), and σ is the annual volatility of the stock price (in %). Each option price P_0 was calculated by using the finite difference method[1] and served as the exact value of the American puts. Applying the property that option prices are linear homogenous in S_0 and X, i.e., $\gamma \cdot f_0(S_0, X, r, T, \sigma) = f_0(\gamma \cdot S_0, \gamma \cdot X, r, T, \sigma)$ where γ is a constant and f_0 denotes a function for calculating the option price, we are able to use a stock price $S_0 = 1$ for all the options of the training sample. The remaining parameters were drawn randomly based on uniform distributions. In accordance with the literature as well as realistic circumstances we defined the following domains: $2 \le r \le 10$, $1/360 \le T \le 1/4$, $10 \le \sigma \le 50$, and $0.8 \le \alpha \le 1.2$. The domain of the moneyness ratio $\alpha = S_0/X$ is based on the consideration that option trading always starts near-the-money. Additionally, Stephan and Whaley (1990) look at a sample of 950,346 stock option transactions and report that the moneyness is between 0.9 and 1.1 in about 78 % of cases.

In accordance with the Generalized Ant Programming approach we transformed each tuple $\langle P_0, S_0, X, r, T, \sigma\rangle$ into an input data record $\mathcal{E}_i$ ($\widehat{=}\ \langle S_0, X, r, T, \sigma\rangle$) and a corresponding target output data record $\mathcal{A}_i^S$ ($\widehat{=}\ \langle P_0\rangle$), for $i = 1, \ldots, 1000$. For the aggregated deviations used in (7) we used the sum of the squared errors $\sum_{i=1}^{1000}(\mathcal{A}_i - \mathcal{A}_i^S)^2$. In the terminal symbol set we included the variables S_0, X, r, T, σ, α, ephemeral constants, the commonly used mathematical operators $+$, $-$, $*$, $\div$, $\sqrt{x}$, $\ln(x)$, x^2, x^y, and the cumulative distribution function of the univariate standard normal distribution $\Phi(x)$. For the substitution rules, $\mathcal{R}$, and the non-terminal symbol set, $\mathcal{N}$, we used subsets of the grammar defining Jensen and Wirth's "Standard Pascal" (see Jensen and Wirth (1975), pp. 110–118). This subset was chosen so that simple analytical expressions can be derived. For the other GAP related parameters we used the following settings resulting in a balanced relationship between convergence speed, calculation effort and effectiveness of the GAP algorithm. The ant colony includes $\mathcal{K} = 50$ members and for the relative importance of pheromone trails we used $\alpha = 1$. The remaining parameters are defined as follows: $\beta = 1, \rho = 0.5$ and $Q = 1$. Among other things, η is used to restrict the formula complexity. Furthermore, we stopped the GAP algorithm after 100,000 cycles.

3.2 APPROXIMATION

Applying the GAP approach as described we get various approximations for the valuation of American put options on non-dividend paying stocks. One of our best approximations (for options with $S_0 = 1$ and $S_0 > S^*$ as mentioned above) is given by

$$P_0 \approx P_0^{GAP} = X \cdot e^{-\bar{r}\cdot T} \cdot \Phi(-d_2) - S_0 \cdot \Phi(-d_1) \qquad (11)$$

where

$$d_1 = \frac{\ln(S_0/X)}{\sigma\sqrt{T}}, \qquad d_2 = d_1 - \sigma\sqrt{T}.$$

The above formula is as derived by the GAP algorithm without possible simplification. Looking at (11) it is amazing that our formula can be characterised as a simple approximation when compared to other analytical approximations presented in the literature. Furthermore, our formula shows a strong structural similarity to the Black/Scholes–Merton equation for valuing European put options. Both formulas are identical except for the parameters d_1 and d_2 as well as $\bar{r}$ given in the appendix.

[1] With $\Delta t = (1/365)/5$, $\Delta S = 0.01$, and $S_{\max} = 2 \cdot S_0$.

4 EXPERIMENTAL RESULTS

To give a first impression of the accuracy of the GAP based analytical approximations for the valuation of American put options on non-dividend paying stocks we use the data sets of Geske and Johnson (1984) and Barone-Adesi and Whaley (1988) because these data sets are often used as comparisons. Furthermore, we use a huge sample of 50,000 randomly generated American put options (validation data set). This ensures that the assessment of the approximations will be highly accurate. The corresponding parameters are independent from those of the training sample and their domains are as follows: $10 \leq S_0 \leq 100$, $2 \leq r \leq 8$, $1/360 \leq T \leq 1/4$, $5 \leq \sigma \leq 50$, and $0.8 \leq \alpha \leq 1.2$. The numerically exact put option and killing prices P_0 and S^* were calculated using the finite difference method (with $\Delta t = (1/365)/5$, $\Delta S = 0.01$, and $S_{\max} = 2 \cdot S_0$) and MacMillan's (1986) procedure, respectively. For the *method-related* comparison we use the most frequently quoted analytical approximations P_0^J of Johnson (1983), P_0^{GJ} of Geske and Johnson (1984), and P_0^{MM} of MacMillan (1986).

In Table 1 and 2 the GAP based approximation P_0^{GAP} as well as the most frequently quoted approximations P_0^J, P_0^{GJ}, and P_0^{MM} are applied to the data sets of Geske and Johnson (1984) and Barone-Adesi and Whaley (1988), respectively. P_0 denotes the numerically exact put option price. At the end of each table the three error measures mean absolute error (MAE), mean squared error (MSE) and mean absolute percentage error (MAPE) are given for each approximation. The approximation results can be summarised as follows:

- From Tables 1 and 2 it can be seen that Johnson's (1983) put pricing formula delivers less accurate approximations. The mean absolute errors are about 7 and 71 pence, respectively. The next best approximation comes from the GAP approach having mean absolute errors of about two and three pence, respectively. MacMillan's (1986) and Geske and Johnson's (1984) approximations are better than the others because their mean absolute errors are between about four tenths of a penny and two pence. With respect to the accuracy of the GAP based approximation we have to keep in mind that only 40 % of the options are consistent with the parameter domains used in the GAP application. However, if we look at these options exclusively the GAP based formula delivers the best approximations.

The application results just shown cannot be used for a general assessment of the accuracy of the approximation formulas because the underlying data sets are too small. If we use the 1,000 data records used in the GAP approach as a basis for a general assessment the problem arises that this data sample represents training data, and thus the assessment would be open to criticism of being a "self-fulfilling prophecy". Therefore, the definitive judgement of the approximation formulas is to be made from the above mentioned validation data set which is independent of the training data set. Based on the validation data set Table 3 gives the error measures and the graph in Figure 1 shows the accuracy of the GAP based approximations as well as Johnson's (1983), Geske and Johnson's (1984), and MacMillan's (1986) put pricing formulas. The accuracy is shown in terms of cumulated frequencies of the absolute deviations between the numerically calculated exact put option price and the approximations just mentioned.

- Johnson's (1983) put pricing formula delivers the weakest approximation results. This can be seen from Table 3 as well as the graph in Figure 1. While for the other approximations it can be said with almost 100 % probability that the approximated option prices differ from the numerically

Table 1: Approximations for the value of American put options on non-dividend paying stocks ($S_0 = 40, r = 4.88$, data from (Geske and Johnson 1984, page 1519)).

T	σ	X	P_0	P_0^J	P_0^{MM}	P_0^{GJ}	P_0^{GAP}
0.0833	0.20	35.00	0.0063	0.0062	0.0065	0.0062	0.0063
0.3333	0.20	35.00	0.2001	0.1969	0.2044	0.2000	0.2042
0.5833	0.20	35.00	0.4323	0.4205	0.4415	0.4318	0.4582
0.0833	0.20	40.00	0.8509	0.8406	0.8503	0.8521	0.8533
0.3333	0.20	40.00	1.5787	1.5262	1.5768	1.5759	1.5937
0.5833	0.20	40.00	1.9894	1.8916	1.9888	1.9827	2.0551
0.0833	0.20	45.00	5.0000	4.8403	5.0000	4.9969	5.0000
0.3333	0.20	45.00	5.0875	4.7882	5.0661	5.1053	5.1069
0.5833	0.20	45.00	5.2661	4.8584	5.2364	5.2893	5.3536
0.0833	0.30	35.00	0.0777	0.0777	0.0780	0.0772	0.0773
0.3333	0.30	35.00	0.6967	0.7056	0.7014	0.6972	0.7012
0.5833	0.30	35.00	1.2188	1.2390	1.2281	1.2198	1.2506
0.0833	0.30	40.00	1.3081	1.3047	1.3078	1.3103	1.3104
0.3333	0.30	40.00	2.4810	2.4757	2.4783	2.4801	2.4952
0.5833	0.30	40.00	3.1681	3.1634	3.1667	3.1628	3.2331
0.0833	0.30	45.00	5.0590	4.9910	5.0470	5.0631	5.0578
0.3333	0.30	45.00	5.7042	5.6090	5.6794	5.7017	5.7232
0.5833	0.30	45.00	6.2421	6.1265	6.2150	6.2367	6.3292
0.0833	0.40	35.00	0.2466	0.2499	0.2472	0.2461	0.2461
0.3333	0.40	35.00	1.3447	1.3886	1.3491	1.3461	1.3421
0.5833	0.40	35.00	2.1533	2.2475	2.1619	2.1553	2.1698
0.0833	0.40	40.00	1.7659	1.7763	1.7659	1.7688	1.7670
0.3333	0.40	40.00	3.3854	3.4494	3.3825	3.3863	3.3902
0.5833	0.40	40.00	4.3506	4.4728	4.3494	4.3475	4.3943
0.0833	0.40	45.00	5.2856	5.2706	5.2735	5.2848	5.2847
0.3333	0.40	45.00	6.5078	6.5535	6.4875	6.5015	6.5211
0.5833	0.40	45.00	7.3808	7.4874	7.3597	7.3695	7.4498
MAE				0.0692	0.0082	0.0040	0.0210
MSE ($\times 10^{-5}$)				1351.8420	15.4237	4.4433	117.0473
MAPE				0.0217	0.0045	0.0025	0.0096

Table 2: Approximations for the value of American put options on non-dividend paying stocks ($X = 100$, data from (Barone-Adesi and Whaley 1988, page 315)).

T	σ	S_0	P_0	P_0^J	P_0^{MM}	P_0^{GJ}	P_0^{GAP}
$r = 8.00$							
0.25	0.20	80.00	20.0000	18.0909	20.0000	20.0012	20.0000
0.25	0.20	90.00	10.0353	9.0470	10.0130	10.0730	10.0456
0.25	0.20	100.00	3.2217	3.0378	3.2201	3.2115	3.2262
0.25	0.20	110.00	0.6642	0.6406	0.6810	0.6647	0.6725
0.25	0.20	120.00	0.0888	0.0865	0.0967	0.0879	0.0863
$r = 12.00$							
0.25	0.20	80.00	20.0000	17.1344	20.0000	20.0112	20.0000
0.25	0.20	90.00	10.0000	8.2620	10.0000	9.9811	10.0000
0.25	0.20	100.00	2.9225	2.6265	2.9251	2.9110	2.9050
0.25	0.20	110.00	0.5541	0.5189	0.5781	0.5541	0.5549
0.25	0.20	120.00	0.0685	0.0654	0.0789	0.0676	0.0609
$r = 8.00$							
0.25	0.40	80.00	20.3196	19.7588	20.2478	20.3699	20.3078
0.25	0.40	90.00	12.5635	12.4222	12.5142	12.5511	12.5769
0.25	0.40	100.00	7.1049	7.1204	7.0999	7.1018	7.1162
0.25	0.40	110.00	3.6968	3.7476	3.7120	3.7017	3.7002
0.25	0.40	120.00	1.7885	1.8310	1.8068	1.7892	1.7824
$r = 8.00$							
0.50	0.20	80.00	20.0000	16.6555	20.0000	19.9402	20.0000
0.50	0.20	90.00	10.2890	8.8392	10.2348	10.3712	10.4406
0.50	0.20	100.00	4.1885	3.7889	4.1933	4.1519	4.3474
0.50	0.20	110.00	1.4095	1.3140	1.4459	1.4121	1.5142
0.50	0.20	120.00	0.3969	0.3768	0.4244	0.3961	0.4239
MAE				0.7083	0.0184	0.0173	0.0256
MSE ($\times 10^{-4}$)				14886.3928	7.3752	8.2725	29.2733
MAPE				0.0721	0.0218	0.0034	0.0177

Table 3: Error measures for the approximations for the value of American put options on non-dividend paying stocks based on a sample of 50,000 put options.

	P_0^J	P_0^{MM}	P_0^{GJ}	P_0^{GAP}
MAE	0.0674	0.0059	0.0031	0.0025
MSE ($\times 10^{-3}$)	27.0546	0.1197	0.1008	0.0201
MAX	1.8806	0.0827	1.2102	0.0584

exact option prices by no more than 5 pence, deviation to this level is only found in Johnson's (1983) solution in 72 % of cases. Put another way, in 28 % of cases there is a deviation of more than 5 pence. Due to these poor results, Johnson's (1983) approximation is not considered in further discussions.

- The next best approximations come from MacMillan (1986) and Geske and Johnson (1984) having MAEs of about 6 and 3 tenths of a penny and maximum absolute deviations of about 8 and 120 pence, respectively. In comparison, the GAP based formula delivers the best approximations having a MAE of two and a half tenths of a penny and a maximum absolute deviation of about 6 pence.

- In option pricing we usally look at the penny accuracy of an approximation. From the graphs in Figure 1 it can be seen that the penny accuracy

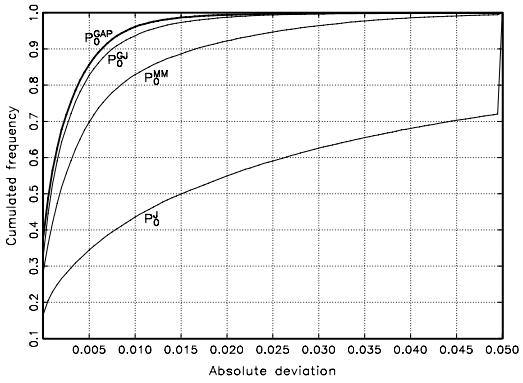


Figure 1: Cumulated frequencies of the absolute deviations between P_0 and the approximations based on a sample of 50,000 put options.

of MacMillan's and Geske and Johnson's, approximations is achieved in about 83 and 94 % of cases, respectively. In comparison, the penny accuracy of the GAP approximation is already achieved in about 96 % of cases.

Summing up, for realistic and frequently observed option parameters we can conclude that the GAP based approximation clearly outperforms the approximations of Johnson, MacMillan as well as Geske and Johnson. Additionally, our approximation consists only of fundamental mathematical operations and is therefore easy to use whereas, e.g., Geske and Johnson's formula requires at least the distribution function of the trivariate, possibly also the multivariate, standard normal distribution which is normally calculated using numerical integration. Moreover, the results presented so far represent work in progress and seem to be very promising.

5 CONCLUSION

In this paper we have introduced the Generalized Ant Programming approach as a new method for solving problems in which the search space of feasible solutions consists of computer programs. We have shown that Generalized Ant Programming can be used to derive accurate analytical approximations for the valuation of American put options on non-dividend paying stocks. Based on experimental data as well as huge validation data sets we have shown that our formula delivers accurate approximation results and outperforms other formulas presented in the literature.

Appendix

$$\bar{r} = -r \cdot \Phi\left(\frac{c_6 \ln(r)}{T}\right)\left[\frac{\ln(X)^2}{\sigma^2} C \sqrt{\Phi\left(\frac{r}{\ln(X)\sigma^2}\right)\frac{r}{T}} - \Phi\left(\Phi\left(\ln(r) D\right)\right)\right]$$

where

$$\begin{aligned}
A &= (r - c_1) X / \Phi\left(T\left[\ln(r) + \frac{r\sqrt{\Phi\left(r/\ln(X)\right) r/T}}{T\sqrt{\alpha X}} - \frac{\ln(X)}{\sigma}\right]\right) \\
B &= \sqrt{X\Phi\left(A\sigma X \Phi\left(-c_2 + T/B^*\right)\right)/B^*}, \quad B^* = \sigma(\ln(X) S_0/\sigma)^2 \\
C &= \sqrt{B + \Phi(-\sigma) - \sqrt{T} + \alpha/\sigma + \Phi(S_0 + C^*) - T} \\
C^* &= \left(\sqrt{\ln(\sigma)\ln(r)} - c_3\right)(T + \ln(X)/\sigma) \\
D &= c_4 \sigma / T\left[\sigma\Phi\left(\ln(X)/\sigma^2 + T\right) + \left(\sqrt{T}\sigma^2 \ln(r)\right)^2 - \Phi(-T)\right] - c_5
\end{aligned}$$

and

$$\begin{array}{lll}
c_1 = 2.190564158 & c_2 = 1.757516557 & c_3 = 7.556576426 \\
c_4 = 0.219427468 & c_5 = 0.119453070 & c_6 = -0.158050093
\end{array}$$

References

Aho, A. V., and J. D. Ullmann, 1972, *The Theory of Parsing, Translation, and Compiling* vol. I Parsing. (Prentice Hall Englewood Cliffs).

Barone-Adesi, G., and R. E. Whaley, 1987, Efficient Analytic Approximation of American Option Values, *Journal of Finance* 42, 301–320.

Barone-Adesi, G., and R. E. Whaley, 1988, On the Valuation of American Put Options on Dividend–Paying Stocks, *Advances in Futures and Options Research* 3, 1–13.

Black, F., and M. Scholes, 1973, The Pricing of Options and Corporate Liabilities, *Journal of Political Economy* 81, 637–659.

Bonabeau, E., M. Dorigo, and G. Theraulaz, 1999, *Swarm Intelligence: From Natural to Artificial Systems.* (Oxford University Press New York).

Brennan, M., and E. Schwartz, 1977, The Valuation of American Put Options, *Journal of Finance* pp. 449–462.

Chen, S.-H., W.-C. Lee, and C.-H. Yeh, 1999, Hedging Derivative Securities with Genetic Programming, *International Journal of Intelligent Systems in Accounting, Finance and Management* 8, 237–251.

Chidambaran, N. K., C. W. J. Lee, and J. R. Trigueros, 2000, Option Pricing via Genetic Programming, in *Computational Finance 1999*, ed. by Y. S. Abu-Mostafa et al. pp. 583–598 Cambridge, Massachusetts. The MIT Press.

Cox, J., S. Ross, and M. Rubinstein, 1979, Option Pricing: A Simplified Approach, *Journal of Financial Economics* pp. 229–263.

Dorigo, M., 1992, Optimization, Learning, and Natural Algorithms, Ph.D. thesis Politecnico di Milano Milano.

Dorigo, M., G. Caro, and L. M. Gambardella, 1999, Ant Algorithms for Discrete Optimization, *Artificial Life* pp. 137–172.

Dorigo, M., V. Maniezzo, and A. Colorni, 1991, Positive feedback as a Search Strategy, Working paper, 91–016 Politecnico di Milano.

Eyckelhof, C. J., 2001, Ants Systems for Dynamic Problems: The TSP Case – Ants caught in a traffic jam, Working paper, University of Twente.

Fischer, E. O., 1993, Analytic Approximation for the Valuation of American Put Options on Stocks with Known Dividends, *International Review of Economics and Finance* pp. 115–127.

Geske, R., and H. E. Johnson, 1984, The American Put Option Valued Analytically, *Journal of Finance* pp. 1511–1524.

Guntsch, M., M. Middendorf, and H. Schmeck, 2001, An Ant Colony Optimization Approach to Dynamic TSP, in *Proceedings GECCO–2001*, ed. by L. Spector et al. pp. 860–867 San Fransico. Morgan Kaufmann.

Hutchinson, J. M., A. W. Lo, and T. Poggio, 1994, A Nonparametric Approach to Pricing and Hedging Derivative Securities Via Learning Networks, *Journal of Finance* pp. 851–889.

Jensen, K., and N. Wirth, 1975, *Pascal, User Manual and Report.* (Springer Verlag New York) 2nd edn.

Johnson, H. E., 1983, An Analytic Approximation for the American Put Price, *Journal of Financial and Quantitative Analysis* pp. 141–148.

Keber, C., 2000, Option Valuation with the Genetic Programming Approach, in *Computational Finance 1999*, ed. by Y.S. Abu-Mostafa et al. pp. 689–703 Cambridge, Massachusetts. The MIT Press.

Kim, I. J., 1990, The Analytic Valuation of American Options, *Review of Financial Studies* pp. 547–572.

Koza, J. R., 1992, *Genetic Programming. On the Programming of Computers by Means of Natural Selection.* (The MIT Press Cambridge, Massachusetts).

MacMillan, L. W., 1986, Analytic Approximation for the American Put Option, *Advances in Futures and Options Research* 1, 119–139.

Merton, R. C., 1973, Theory of Rational Option Pricing, *Bell Journal of Economics and Management Science* 4, 141–183.

Roll, R., 1977, An Analytic Valuation Formula for Unprotected American Call Options with Known Dividends, *Journal of Financial Economics* pp. 251–258.

Roux, O., and C. Fonlupt, 2000, Ant Programming: Or How to Use Ants for Automatic Programming, in *Proceedings of ANTS'2000*, ed. by M. Dorigo et al. pp. 121–129.

Stephan, J. A., and R. E. Whaley, 1990, Intraday Price Changes and Trading Volume Relations in the Stock and Stock Option Markets, *Journal of Finance* 45, 191–220.

Stützle, T., and H. H. Hoos, 2000, MAX-MIN Ant System, *Future Generation Computer Systems Journal* 16, 889–914.

Effects of Agent Representation on the Behavior of a Non-Reciprocal Cooperation Game

Nicole P. Leahy

Graduate Program in Bioinformatics and Computational Biology
Iowa State University
Ames, IA 50011

Abstract

Game theoretic models are often used to simulate phenomena observed in the natural world. It is generally assumed that the implementation (or representation) of the agents within a game has no significant effect on the outcome of simulations. To test this assumption the effect of changing the representation of agents in the non-reciprocal cooperation game studied by Riolo et al. (2001) was used. In addition to the implementation used by Riolo et al. (2001), agents were also represented as higher dimension real vectors, integer values, at bit-strings. It was found that the method of agent implementation used has a highly significant effect on the cooperation rate and general behavior of the simulation.

1 INTRODUCTION

Biologists and social scientists have long been interested in the evolution of cooperation (e.g. Roy, 2000; Sigmund and Nowak, 1999; Mouston et al., 2000; Hemesath, 1994). It is not understood how selfishly acting individuals could evolve cooperative behavior (Axelrod, 1984). Scientists studying this phenomenon often use theoretical games to investigate the origin of cooperation. It is often tacitly assumed that agent representation has only minor effects on the long-term behavior and results of the model. The purpose of this paper is to present the results of a study in which the method of representing the agent in the model published by Riolo, et al. (2001) was altered.

In Riolo et al. (2001)'s game individuals (agents) learned to cooperate despite the cost of assisting another agent and without reciprocity. In brief, this game was designed so that an agent assisted another agent if identification tags were sufficiently similar. The agents had no memory of previous encounters and were unlikely to play other agents more than once (a substantial departure from the usual iterated game used to study the evolution of cooperation). Riolo et al. (2001) reported agents to assist other agents at a rate of 73.6%, after evolution. A more detailed description of the Riolo et al. (2001) simulation is provided in the Experimental Design section.

Agents in Riolo et al. (2001) consisted of two real numbers: the first served as an identity tag and the other determining the altruistic behavior. This study used three alternate implementations of the agents: multi-dimensional real vectors, integer value, and bit-strings. Using a multi-dimensional vector representation expands the tag space (the number of possible tags). The real number representation of computers is a fine-grained approach since tags may vary by very small amounts. When using integer value tags with a maximum value, the difference between agents becomes more pronounced. As the size of the maximum allowable integer increases, the results should approach those of the single dimensional real vector case. Using bit-strings examines the effect of both higher dimensionality and the granularity of the environment.

2 EXPERIMENTAL DESIGN

The experiments in this paper were conducted in the same manner as described in Riolo et al. (2001). Specifically, each simulation used a population of 100 agents which were evolved for 30,000 generations. Each agent had a tag (τ), tolerance threshold (T), and a score. Every generation, each agent played three agents selected at random with replacement. For each place the first agent (agent A) determines if it will assist the other agent (agent B). If the distance between tags of agent A and agent B was lower than the threshold of agent A, the score of agent A was lowered by 0.1, and the score of agent B increased by 1.0. This represents a costly contribution by agent A to agent B.

The next generation was determined with tournament selection of size two, where the population was randomly shuffled and two agents were select without replacement until there were no remaining agents. if the agents in each pair had equal scores, each was copied to the next generation. If one agent had a higher score, that agent was copied twice to the next generation and the agent with the lower score was not copied. Each copy was then

subjected to mutation. Mutation of the tag occured at a frequency of 0.1, and was performed by generating a new tag uniformly at random in the tag space. At the same frequency, the tolerance threshold mutated by a representation-dependent function with a distribution designed to leave the simulation as close as possible to the original study (Gaussian with mean zero and a standard deviation of 0.01). If the new $T<0$, then it was set to 0.

2.1 MULTI-DIMENSION REPRESENTATION

An agent tag in Riolo et al. (2001) was represented as a real number, $\tau \in [0,1]$, with tag distance calculated as the absolute value of the difference between two tags. Similarly, the agent tolerance was initialized as a real number, $T \in [0,1]$. To examine the effect of a larger tag space on the behavior of the game, tags were represented as real vectors, $\tau \in [0,1]_n$, where n is the dimension of the tag space. The tolerance was represented as a single real variable with $T \in [0,1]$. The distance between the tags of two agents, A and B, is the Euclidean distance,

$$D_{\tau_A \tau_B} = \sqrt{\sum_{i=1}^{n} \left(\tau_A(i) - \tau_B(i)\right)^2}$$

Player A assists player B if $D_{\tau_A \tau_B} \leq T_A$.

The mutation of T was the same as in Riolo et al. (2001) except that the standard deviation (σ) of the mutation distribution was adjusted so that the ratio of the hypervolume of the hypersphere of radius σ (V_σ) to the hypervolume of the tag space (V_t) was constant across dimensions (Table 1). This was done in order to insure that the tolerance mutation was equivalent to theat used in Riolo et al. (2001). Runs were performed for n = 1, 2, 3, 4, 5, where n is the number of dimensions of the tag space. 30 replicates were run for each tag space. Note that the case where n=1 is identical to the setup for Riolo et al. (2001).

Table 1: σ for Higher Dimensions

n	σ	$V_\sigma : V_t$
1	0.01	0.02
2	0.079788456	0.02
3	0.16838903	0.02
4	0.25231352	0.02
5	0.32805473	0.02

2.2 INTEGER REPRESENTATION

Using real values for τ and T is a relatively fine-grained environment since it is extremely unlikely that any two agents will have the same τ. In Riolo et al. (2001), T most frequently changed by small values relative to the tag space, creating a fine distinction between agents with an approximately continuous tag space. By employing integer representation of the tag space and tollerance, the simulation becomes more discrete. Both τ and T were initialized as τ, $T \in [0,1,2,...,I]$, where I is the maximum integer. Constraining the tag space and randomly selecting new tags may result in duplication of the tags of other agents, especially at low values for I. The mutation of T was performed by adding $\Delta = \sum_{i=1}^{I} \delta_i$, for $\delta_i \in [-1,0,+1]$, $\delta_i \in$ [-1, 0, +1], with $p(-1) = p(+1) =$ 0.1, and $p(0) = 0.8$. Experiments were run for I=10, 100, 1000, and 10,000.

2.3 BIT-STRING REPRESENTATION

Implementing the tag and tolerance as bit strings both increases the tag space by using larger strings and constrains the tag space, reducing the values of each dimension to one of two states. The distance between two agents was measured as the Hamming distance. The value of T was implemented as the weight of the tolerance binary string. Mutation of the tolerance was performed as bit flips determined by a Poisson distribution with the expected probability of each bit-flip set equal to 0.01. Since it is not possible for $T < 0$, there were no boundary effects as there were in the other two representations. Experiments were performed for string lengths, L of 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100 bits. The bit-string for the tag and tolerance were of equal length in all simulations.

3 RESULTS

An interesting difference between the results of these experiments and those reported by Riolo et al. (2001) was the presence of failed states. A *failed state* is an occurrence of donation rates less that 10% for at least one generation. The value of 10% was selected from examination of the reproduction of the results of Riolo, et al. (2001). In only one replicate did a population fall to a donation rate of 10% or lower one without persisting in a failed state for multiple generations. The comparisons of a typical run from Riolo et al. (2001) compared to a failed state are shown in Figures 1 and 2. The *donation rate* is the proportion of plays per generation that result in one agent assisting another. A *cooperative state* is defined as an occurrence of donation rates greater than 10%. Significance levels were calculated using ANOVA and student-t tests.

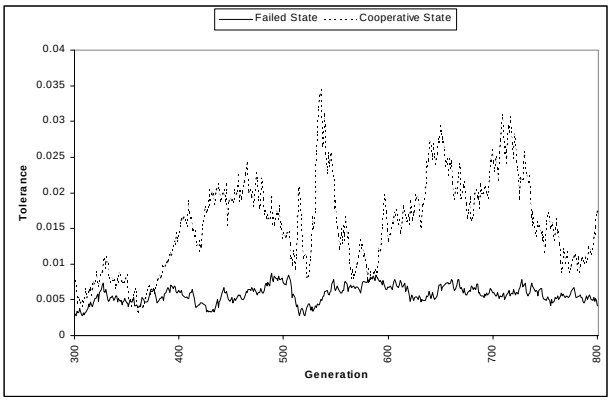


Figure 1: Tolerance Examples For Cooperative States (n=1, Replicate 6) And Failed States (n=1, Replicate 0)

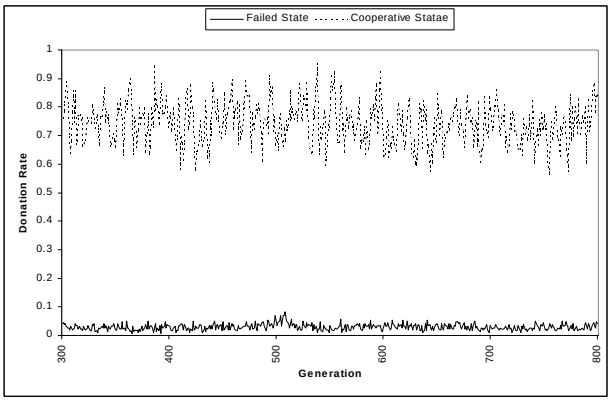


Figure 2: Example Donation Rate For Cooperative States (n=1, Replicate 6) And Failed States (n=1, Replicate 0)

3.1 MULTI-DIMENSION REPRESENTATION

Results of the one-dimensional case, a replicate of Riolo et al. (2001) were consistent with those they reported. In this study the mean donation rate was 73.4 ± 0.3%, whereas Riolo et al. (2001) foud 73.6%. The average tolerancewas identical at 0.019.

As the number of dimensions increased the average number of failed states increased significantly (Table 2) with $p < 0.0001$. The dimensions in which significantly different proportions of occured failed states are given in Table 3.

Table 2: Mean Values for Experiments

Experiment	Failed States	Tolerance	Donation Rate
n=1	18.6	0.0185	0.7344
n=2	1853.2	0.1071	0.6837
n=3	3898.3	0.1816	0.6340
n=4	4949.9	0.2394	0.6086
n=5	5630.2	0.2937	0.5913
I=10	0	0.0951	0.7217
I=100	127.7	0.0481	0.7096
I=1000	164.1	0.0229	0.7263
I=10000	23.2	0.0115	0.7420
L=10	0	0.0027	0.9898
L=20	0	0.0068	0.9807
L=30	0	0.0180	0.9730
L=40	0	0.0756	0.9702
L=50	0	0.2557	0.9817
L=60	0	0.4185	0.9953
L=70	0	0.4602	0.9983
L=80	0	0.4675	0.9987
L=90	0	0.4760	0.9988
L=100	0	0.4785	0.9989

The mean donation rate was found to decrease with increasing dimension ($p < 0.0001$) with the same significant groupings as for the average number of failed states (Table 3). The mean tolerances of the dimensions were significantly different ($p < 0.0001$).

Table 3: Significantly (*) and Insignificantly (·) Different Occurrences of Failed States Between Multi-Dimension Representations

n	1	2	3	4	5
1	-	*	*	*	*
2		-	*	*	*
3			-	·	*
4				-	·
5					-

3.2 INTEGER REPRESENTATION

The results of the integer representations were compared with the results of the one dimensional real vector representation, *R* (the replication of Riolo et al. (2001)). ANOVA results indicated that the average number of

failed states was significantly different from R ($p < 0.0230$). Only the mean number of failed states for I=1000 was significantly higher than that for R. The significant comparisons are shown in Table 4.

Table 4: Significantly (*) and Insignificantly (·) Different Occurrences of Failed States for Integer Representations

	***I*=10**	**100**	**1000**	**10000**	***R***
***I*=10**	-	*	*	·	·
100		-	·	·	·
1000			-	*	*
10000				-	·
R					-

The mean tolerance of the integer representations decreased with increasing I ($p < 0.0001$), with each mean tolerance significantly lower than the previous (Table 2). At I=10,000, the tolerance was less than R. The mean donation rate decreased from I=10 and I=100 and then increased with I=10,000 having a greater donation rate than R ($p < 0.0001$).

3.3 BIT-STRING REPRESENTATION

There were no failed states in any of the bit-string experiments, so there was no significant difference across simulations with different values of L. The mean tolerance increased with increasing L ($p < 0.0001$) beginning at L=30 until L=70, with no significant difference between R and L=10, 20, and 30. The mean donation rate gradually decreased from L=10 to L=50 and then increased for L=50 to L=70 ($p < 0.0001$). The mean donation rate for bit-strings was never below 97.0%, significantly greater than R ($p < 0.0001$). See Table 2 for the data.

4 DISCUSSION

4.1 MULTI-DIMENSION REPRESENTATION

The results demonstrated that the frequency of failed states increased as the tag space increased in dimension. In the one-dimensional space when a tag becomes the most prevalent tag the selection for low tolerance is weakened and the population will move towards higher tolerance. Eventually, an agent with a different tag will fall within the tolerance of the dominant tag. This can occur through increasing the average tolerance of the agents with the dominant tag or the generation of agents with a new tag within the tolerance threshhold of the dominant tag type. Over time, agents with the rarer and less tolerant tag will become dominant through exploiting the previously dominant agents, as in Riolo et al. (2001).

In the one-dimensional case an exploitative agent will have a tag either less than or greater than the most common tag. When the number of dimensions was increased there were more axes along which tags may vary, allowing multiple agents with differing tags to exploit the dominant tag. Agents with the dominant tag will quickly be removed from the population with no new dominant type to take its place. This failed state will persist while numerous agents with low tolerances fail to assist one another. Eventually, another dominant type will emerge. From the duration of failed states, it is thought that the average amount of time until a new type takes over increases as the dimension of the tag space increases. It is conjectured that the increasing dimension makes it more difficult for a new tag to be within the tolerance threshold of an extant tag.

The decrease in donation rate observed as the number of dimensions in the tag space increased was due to a reduced donation rate in the failed states compared to the cooperating states. Adjusting for failed states by dividing the mean donation rate by the proportion of cooperative generations over all generations results in an increase in the average donation rate (Table 5). However, the one-dimensional case is still significantly higher than those of higher dimensions ($p < 0.0001$) with the higher dimensions having mean donations rates of 72.9% or 72.8%. This seems to indicate that the occurrence of failed states accounts for most of the decrease in donation rate.

Table 5: Adjusted Tolerance and Donation Rates for the Multi-Dimensional Representations

n	***Tolerance***	***Donation Rate***
1	0.0185	0.7349
2	0.1144	0.7289
3	0.2100	0.7290
4	0.2887	0.7284
5	0.36583	0.7288

The increase in mean tolerance is more puzzling. Since the standard deviation of the tolerance mutation function increased with increasing dimension, it was thought that there may be a consistent ratio between the two. Calculating the ration of the standard deviation of the mutation function to the mean tolerance demonstrate that this was not the case (Table 6). The ratios were calculated with both the mean tolderances from Table 2 and the adjusted tolerance from Table 5 While the mutation function likely has some effect, the selection of the game acts to decreasee the tolerance, weakening the correlation between the mutation function and the mean tolerance.

Table 6: Ratio of the Standard Deviation of the Tolderance Mutation Function to the Mean Tolerance

n	$E(T_n):\sigma_n$ *unadjusted*	$E(T_n):\sigma_n$ *adjusted*
1	0.0185	0.7349
2	0.1144	0.7289
3	0.2100	0.7290
4	0.2887	0.7284
5	0.36583	0.7288

4.2 INTEGER REPRESENTATION

The results of the integer representations were expected to approach those of the one-dimensional real vector simulations (*R*) with increasing *I*. It was found that the mean number of failed states and donation rate initially diverged from *R* and then approached that of *R* (Talbe 2). Examination of the data indicated that the failed states occurred early in the run and persisted for several generations before evolving to a cooperative state (Figure 3). For *I*=10,000, the initial failed states resemble the dip in donations rates observed by Riolo, et al. (2001). Given the decreased cooperation and increased failed states for *I* = 100 and 1000, it is unlikely that the cooperative behavior seen at *I*=10 and *I*=10,000 were due to the same dynamics. For *I*=10, a mutated tag may take on one of only ten values, so it was likely that the new tag would be the same as existing agents. Therefore, if the tag of an agent with low tolerance was mutated it would cooperate with agents already present. This coarse granularity makes it unlikely for an agent to be in a position to exploit others. For *I*=10,000, the dynamics are similar to those for *R*.

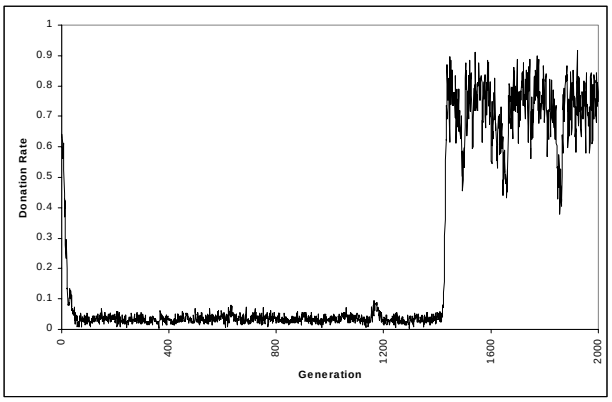


Figure 3: Example of the Initial Behavior for Integer Representations (I = 1000, Replicate 0)

4.3 BIT-STRING REPRESENTATION

Of the three types of representations, bit-strings produced the greatest average donation rate. All tested string lengths had mean donation rates exceeding 97%. In the case of runs with many agents of near zero tolerance, this must have been the result of identical or nearly identical agents. When the average tolerance is large ($T \geq 0.47$), it is not necessary for agents to be highly similar. The process of generating a new tag results in a tag that by chance will have 50% similarity to all possible tags. With an average $T \geq 0.47$, it is likely a tag will be in the tolerance threshold of the majority of the agents. Furthermore, since the agent with the new tag will have a *T* close to 0.5, most of the agents will be within its own threshold.

The pattern of populations of short strings evolving low tolerance while long strings evolve high tolerance remains to be explained. For intermediate values of *L* (lengths of 40, 50, and 60), the populations begin with a tolerance level similar to those of the longest strings, but changed to low tolerance levels (Figure 4). This suggests that a cooperative state may not be stable if the simulation is run for a large number of generations. The stability of high mean tolerance may be due to the fact that a single bit-flip in the tolerance string would have a greater impact on the population dynamics for low values of *L* than for high values.

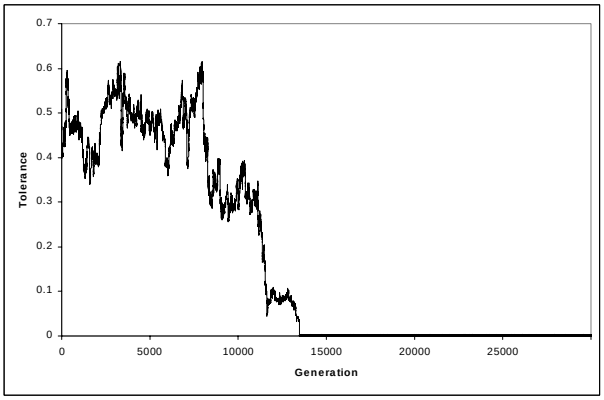


Figure 4: Behavior of Bit-Strings for Intermediate Values of *L*=50 (Replicate 0)

At the initialization of a population there will tend to be more exploitative ($T < 0.4$) and exploitable ($T > 0.6$) agents for shorter strings than longer strings (Table 7). The greater effect of a single bit flip at shorter lengths often results in a population in which agents with low tolerances take advantage of agents with high tolerances. In time, only the agents with low T values remain. When a tag is mutated, there is a greater likelihood of agents being more than 50% dissimilar at shorter lengths. As length increases, the number of exploitative and exploitable agents decreases, leading to the observed trend of more generations passing before *T* becomes small. Eventually, long strings (a length of 40 or greater) maintain high values for *T* for the duration of the run, though this may not be the case if run for more generations than was done in this study.

Table 7. Expected Proportion of Exploitable and Exploitative Agents at Initialization of Bit-String Populations

L	*Proportion*
10	0.363
20	0.263
30	0.200
40	0.154
50	0.119
60	0.092
70	0.072
80	0.057
90	0.045
100	0.035

5 CONCLUSIONS

The results of the experiments described in this paper demonstrate that altering the representation of the agents in the Riolo et al. (2001) simulation has significant effects on the outcome. In the literature, when game theoretic models of the evolution of cooperation are examined, only the rules of the game rather than the digital representation of the agents are modified. The results described above caution that generic statements of a game should not be made from a single implementation. Furthermore, since the outcome of this game is partially dependent upon the representation used, it would not be prudent to present data as a model for the natural world unless it is either confirmed by multiple representations or shown that the chosen representation is a reasonable approximation, such as fitting the model to data from the natural world. With the variety of models used to simulate aspects of the nature, it would be of interest to test the effect of agent representation on the behavior of other game theoretic models.

5 ACKNOWLEDGEMENTS

I would like to thank Dan Ashlock (Department of Mathematics, Iowa State University) for introducing me to game theoretic models of cooperation and his advice on this project.

6 REFERENCES

Axelrod, R. (1984). *The Evolution of Cooperation*. Basic Books, New York.

Hemesath, M. (1994). Cooperate or defect? Russian and American students in a prisoner's dilemma. *Comparative Economic Studies* **176**:83-93.

Mouston, J.M., J. Kinnie, B. Luop, C. Terry, and S.S. Ho (2000). Competitiveness and conflict behavior in simulation of social dilemma. *Psychological Reports* **86**:1219-1225.

Nowak, M.A., and K. Sigmund (1998). Evolution of indirect reciprocity by image scoring. *Nature* **393**:573-577.

Riolo, Rick L., Michael D. Cohen, and Robert Axelrod (2001). Evolution of cooperation without reciprocity. *Nature* **414**:441-443.

Roy, D. (2000). Learning and the theory of games. *Journal of Theoretical Biology* **204**:409-414.

Sigmund, K., and M.A. Nowak (1999). Evolutionary game theory. *Current Biology* **9**:R503-R505.

Intelligent Packets for Dynamic Network Routing Using Distributed Genetic Algorithm

Suihong Liang
abacus@cs.dal.ca

A. Nur Zincir-Heywood
zincir@cs.dal.ca
Faculty of Computer Science
6050 University Avenue,
Halifax, NS, Canada, B3H 1W5

Malcolm I. Heywood
mheywood@cs.dal.ca
+1 902 4942951

Abstract

A distributed GA is designed for the packet switched network routing problem under minimal information. The requirements of such a problem mean that agents are required to possess more intelligence than was previously the case. To this end a distributed GA approach is developed and benchmarked against the AntNet algorithm under the same information constraints.

1 GENERAL FORMATTING INSTRUCTIONS

Network information systems and telecommunication in general rely on a combination of routing strategies and protocols to ensure that information sent by a user is actually received at the desired remote location. In addition, the distributed nature of the problem means that multiple users can make requests simultaneously. This results in delayed response times, lost information or other reductions to the quality of service objectives on which users judge network service. Routing is the process used to determine how a packet travels from source to destination. Protocols are used to implement handshaking activities such as error checking and receiver acknowledgements. In this work, we are interested in the routing problem on computer networks.

The routing problem has several properties, which make it particularly challenging. The problem is distributed in nature; hence a solution that assumes access to any form of global information is not desirable. The problem is also dynamic; hence a solution that is sufficient for presently experienced network conditions may well be inefficient under other loads experienced by the network. Moreover, the traffic experienced by networks is subject to widely varying load conditions, making 'typical' network conditions unrepresentative.

Traditionally, routing strategies are implemented through the information contained in routing tables available at each node in the network (Forouzan, 2001). That is, a table details the next 'hop' a packet takes based on the overall destination of the packet. This should not be taken to imply that a routing table consists of an exhaustive list of all destinations – a form of global information. Instead, the table consists of specific entries for the neighboring nodes and then a series of default paths for packets with any other destination – such as OSPF or BGP4 (Halabi, 1997). Application of a classical optimization technique to such a problem might take the form of first assessing the overall pattern of network traffic, and then defining the contents of each routing table such that congestion is minimized. This approach does not generally work in practice as it simply costs too much to collect the information *centrally* on a regular basis, where regular updating is necessary in order to satisfy the dynamic nature of network utilization. We, therefore, see the generic objectives of a routing strategy to be both dynamically reconfigurable and be based on locally available information, whilst also satisfying the user quality of service objectives (i.e. a global objective).

Several approaches have been proposed for addressing these objectives including: active networking (Tennenhouse *et. al.*,1997), social insect metaphors (Di Caro, Dorigo, 1998), (Heusse *et al.*, 1998) cognitive packet networks (Gelenbe *et. al.*,1999), and what might be loosely called other 'adaptive' techniques (Corne *et. al.*, 2000). The latter typically involve using evolutionary or neural techniques to produce a 'routing controller' as opposed to a 'routing table' at each node, where the controller may require knowledge of the global connectivity to ensure a valid route. The global information assumption may be avoided by framing the problem as a reinforcement-learning context (Boyan, Littman, 1994). However, the Q-learning method, on which this is based, results in single path solutions for each destination. Both the social insect metaphor and the cognitive packet approach provide a methodology for routing, without such constraints; by utilizing probabilistic routing tables and letting the packets themselves investigate and report network topology and performance.

All methods as currently implemented, however, suffer from one drawback or another. Cognitive packet networks and active networking algorithms attempt to provide routing programs at the packet level, hence achieving scalable run time efficiency becomes an issue. To date, implementations of 'adaptive' techniques and social insect metaphors have relied, at some point, on the availability of global information (Liang, *et al*., 2002).

The purpose of this work is to investigate the application of a genetic algorithm (GA) to build on lessons learnt from the social insect metaphor. This represents a major departure from previous works attempting to utilize GAs to solve the dynamic routing problem e.g. (Corne D.W., et al. 2000). In particular, a distributed GA is defined in which populations associated with each node of the network are required to co-evolve to solve the problem as a whole. Moreover, the GA interaction with the environment drives the features measured by the routing tables, as opposed to the tables predefining the features for measurement (a form of *a priori* information). Section 2 introduces the 'ant' based social insect metaphor against which the proposed approach is compared. Section 3 introduces the proposed GA-agent scheme. Section 4 summarizes the network on which experiments are performed. Results are presented in section 5 and conclusions drawn in section 6.

2 ROUTING USING A SOCIAL INSECT METAPHORE

As indicated above, active networking (Tennenhouse *et. al.*,1997) and cognitive packet (Gelenbe *et. al.*,1999) based approaches emphasize a per packet mechanism for routing. The aforementioned 'adaptive' techniques (Corne *et. al.*, 2000) tend to emphasize adding 'intelligence' to the routers leaving the packets unchanged. A social insect metaphor provides a middle ground in which the concepts of a routing table and data packet still exist, but in addition, intelligent packets – *ants* – are introduced that interact to keep the contents of the routing tables up to date. To do so, the operation of ant packets is modeled on observations made regarding the manner in which worker ants use chemical trails as a method of indirect *stigmergic* communication. Specifically, ants are only capable of simple stochastic decisions influenced by the availability of previously laid stigmergic trails. The chemical denoting a stigmergic trail is subject to decay over time, and reinforcement proportional to the number of ants taking the same path. Trail building is naturally a bi-directional process, ants need to reach the food (destination) and make a successful return path, in order to significantly reinforce a stigmergic trail (Forward only routing has also been demonstrated (Heusse *et al.*, 1998)). Moreover, the faster the route, then the earlier the trail is reinforced. An ant on encountering multiple stigmergic trails will *probabilistically* choose the route with greatest stigmergic reinforcement. Naturally, this will correspond to the 'fastest' route to the food (destination). The probabilistic nature of the decision, however, means that ants are still able to investigate routes with a lower stigmergic trial.

This approach has proved to be a flexible framework for solving a range of problems including the traveling sales man problem (Dorigo *et al*., 1996) and the quadratic assignment problem (Maniezzo *et al.*, 1999). The work reported here follows the 'AntNet' algorithm of Di Caro and Dorigo (Di Caro, Dorigo, 1998), and is informally summarized as follows,

- Each node in the network retains a record of packet destinations as seen on data packets passing through that node. This is used to periodically, but asynchronously, launch 'forward' ants with destinations stochastically sampled from the collected set of destinations;
- Once launched, a forward ant uses the routing table information to make probabilistic decisions regarding the next hop to take at each node. While moving, each forward ant collects time stamp and node identifier information where this is later used to update the routing tables along the path followed;
- If a forward ant re-encounters a node previously visited before reaching the destination, it is killed;
- On successfully reaching the destination node, total trip time is estimated and the forward ant converted into a backward ant;
- The backward ant returns to the source using exactly the same route as recorded by the forward ant. Instead of using the data packet queues, however, the backward ant uses a priority queue;
- At each node visited by the backward ant the corresponding routing table entries are updated to reflect the relative performance of the path;
- When the backward ant reaches the source, it 'dies'.

Although providing for a robust ant routing algorithm under simulation conditions, an assumption is made, which inadvertently implies the use of global information - knowledge of the number of nodes in the network (Di Caro, Dorigo, 1998). The definition of routing tables is, such that it is assumed that every node has a unique location in the routing table, see Table 1, or a total of L (number of neighboring nodes) by K (number of nodes in the entire network) entries. In practice, this is never the case. To do so would assume that it is first feasible, and secondly, should the network configuration ever change, then all nodes should be updated with the new configuration information. Moreover, as forward ants propagate across the network, the amount of information they need to 'carry' also increases (node identifier and time stamp).

Table-1 Original Routing Table at any Network Node k on the NTTnet

	All Network Nodes (Possible Destinations)			
Outgoing Links (Each node has L neighboring links)	$P_{1,1}$	$P_{1,2}$	------	$P_{1,55}$
	$P_{2,1}$	$P_{2,2}$	------	$P_{2,55}$
	------	------	------	------
	$P_{L,1}$	$P_{L,2}$	------	$P_{L,55}$

In order to avoid the use of global information, the authors modify the information provided at the routing tables such that the routing tables only detail the neighboring nodes, see Table 2, or a total of 2 by L entries. Such a limitation, therefore, places greater emphasis on the learning capacity of the ant. This is particularly significant during step (2) of the ant forward pass (section 2.1). Tables 1 and 2 illustrate the difference in available information for a node in the commonly used Japanese benchmark backbone (NTTNet) routing problem.

Table-2 Proposed Routing Table at any Network Node on the NTTnet

	Neighbor Node	If used for other nodes
Outgoing Links (Each node has L neighboring links)	$P_{1,1}$	$P_{1,d}$, $d \neq 1$
	$P_{2,2}$	$P_{2,d}$,$d \neq 2$
	------	------
	$P_{L,L}$	$P_{L,d}$, $d \neq L$

The following section summarizes the AntNet algorithm.

2.1 ANTNET ALGORITHM

It is assumed that routing tables, T_k, exist at each node, k, in which a routing decision is made. Tables consist of 'n' rows, one row for each neighboring node/link. As far as a normal data packet is concerned, if the destination, d, from current node, k, is a neighbor then the routing is still a stochastic decision. In all other cases, a route is selected based on the neighbor node probabilities.

1. New forward ants, F_{sd}, are created periodically, but independently of the other nodes, from source, s, to destination node, d, in proportion to the destination frequency of passing data packets. Forward ants travel the network using the same priority structures as data packets, hence are subject to the same delay profiles.
2. Next link in the forward ant route is selected stochastically, $p'(j)$, in proportion to the routing table probabilities and length of the corresponding output queue.

$$p'(j) = \frac{p(j) + \alpha l_j}{1 + \alpha(|N_k| - 1)}$$

where $p(j)$ is the probability of selecting node j as the next hop; α weights the significance given to local queue length verses global routing information, $p(j)$; l_j is proportional to the inverse of queue length at destination 'j' normalized to the unit interval; and N_k is the number of links from node k.

3. On visiting a node different from the destination, a forward ant checks for a buffer with the same identifier as itself. If such a buffer exists, the ant must be entering a cycle and dies. If this is not the case, then the ant saves the previously visited node identifier and time stamp at which the ant was serviced by the current node in a buffer with the forward ant's identifier. The total number of buffers at a node is managed by attaching "an age" to buffer space and allowing backward ants to free the corresponding buffer space.
4. When the current node is the destination, $k = d$, then the forward ant is converted into a backward ant, B_{ds}. The information recorded at the forward ant buffer is then used to retrace the route followed by the forward ant.
5. At each node visited by the backward ant, routing table probabilities are updated using the following rule,

   IF (node was in the path of the ant)
   THEN $p(i) = p(i) + r\{1 - p(i)\}$
   ELSE $p(i) = p(i) - r\,P(i)$
   where $r \in (0, 1]$ is the reinforcement factor central to expressing path quality (length), congestion and underlying network dynamics.

As indicated above, the reinforcement factor should be a factor of trip time and local statistical model of the node neighborhood. To this end (Di Caro, Dorigo, 1998) recommend the following relationship,

$$r = c_1\left(\frac{W_{best}}{t_{ant}}\right) + c_2\left\{\frac{I_{\sup} - I_{\inf}}{(I_{\sup} - I_{\inf}) + (t_{ant} - I_{\inf})}\right\}$$

where W_{best} is the best case trip time to destination d over a suitable temporal horizon, W; t_{ant} is the actual trip time taken by the ant; $I_{inf} = W_{best}$; $I_{sup} = \mu_{kd} + \{\sigma_{kd} / [W(1 - \gamma)]^{0.5}\}$.

The estimates for mean, μ_{kd}, and variant, σ_{kd}, of the trip time are also made iteratively, using the trip time information, o_{kd}. Thus,

$$\mu_{kd} = \mu_{kd} + \eta(o_{kd} - \mu_{kd})$$
$$(\sigma_{kd})^2 = (\sigma_{kd})^2 + \eta\{(o_{kd} - \mu_d)^2 - (\sigma_{kd})^2\}$$

Trip time information is now updated incrementally based on the recorded trip duration between current node, k, and ultimate destination, d. This means that it is no longer

necessary to carry all node and duration information as a 'stack' to the target duration as in the original model (Di Caro, Dorigo, 1998). Only the previous node information is therefore carried by each ant.

3 GENETIC ALGORITHM BASED SCHEME TO ROUTING

Simulation of the above AntNet scheme has been shown to provide a robust alternative to six standard routing algorithms – OSPF, SPF, BF, Q-R, P-QR and Daemon (Di Caro, Dorigo, 1998). However, this is not without utilizing routing tables in the AntNet algorithm, which provide entries for all nodes on the network. In practice, such (global) information is not actually available. In (Liang *et al*. 2002), the AntNet algorithm is benchmarked with routing tables configured with information regarding their neighbors alone; Table 2 as opposed to Table 1. The performance of such a system is then deemed unacceptable. Specifically 55% of packets are lost where 'lost' in this work is defined as *any* packet (data or ant) that visits the same node more than once. In order to address this problem, we are, therefore interested in the ability of route finding packets learning to find paths independently from the routing table information. By doing so, we do not rely on the capacity of the routing tables alone to retain information regarding all nodes in the network.

The objective of this work is to investigate a scenario in which the entries themselves are identified dynamically. This will be a first step towards a co-evolutionary model capable of evolving solutions to the packet switched routing problem. The ants, in this case, take the form of individuals from a distributed Genetic Algorithm (GA), hereafter referred to as GA-agents. Individual chromosomes travel the network using a string of next hop offsets, e.g., {1, 5, 0, 4, 2, 3, 5} over the interval [0, [0, *L*], where '*L*' is selected to enable indexing of node connectivity. In all the experiments of section 5, '*L*' is set to 6. On entering a node, genes (offsets) are used to identify the next link using a clockwise count, with respect to the port the GA-agent entered the node i.e. the next link is selected as a modulus of (gene % # of links). Such a representation is then independent of the specific network connectivity, unlike say the GA approach in (Munetomo *et al*., 1997). For each node encountered, the gene, used to select the next link, is incremented and a record is made of the node ID. The process naturally continues until the GA-agent executes its last gene, at which point it becomes a backward agent, returning to its original source node. In the special case of a GA-agent attempting to return down the same link as it entered a node, the router randomly selects the next hop from the available links, and changes the gene to the new value (deterministic mutation). If no next hop is available, then the chromosome is truncated, and the GA-agent becomes a backward agent (see the algorithm "processing agents"). Note, unlike the AntNet algorithm, modification of routing tables only takes place once the GA-agents have returned to their original source, and modifications only affect the source node routing table. The above representation supports single point crossover, resulting in variable length individuals. Mutation randomly selects a gene and adds/ subtracts an integer such that the new gene is still in the interval [0, 6].

Table 3 – GA-agent Routing Table

Agent ID	Agent Fitness	Trip Time (ms) and node ID
95	0.32	(3, J), (9, C), (21, W)
234	0.39	(1,B), (7, A),…, (432, Y)
…	…	…
31	0.71	(5,C), (9, K), …, (871, X)

At initialization, a router sends out half of the population of GA-agents to explore the network. Once the number of returned GA-agents reaches four, the fitness of the four agents is evaluated; the best two agents are then chosen – as in a steady state tournament (See algorithm "updating routing table & population").

The fitness function measures the popularity of nodes visited as well as the time taken to reach nodes encountered by GA-agents. Both of these properties are measured with respect to the original source node. Popularity of destination '*i*' at node '*k*' ($NP_k(i)$) is a dynamic property, measured at the original source node by recoding the frequency of different data packet destinations as seen by the source node over a fixed time window (50 seconds in this case), or

$$NP_k(i) = Dest(i) / TD_k$$

Where TD_k is the total number of data packets passing through node '*k*'; and $Dest(i)$ is the number of data packets with destination '*i*'.

Chromosomes, which find shortest paths to frequently used destinations, are therefore favored. The ensuing fitness function taking the form,

$$\frac{\sum_{\text{for each explored node } i} NP_k(i) \times trip\_time_i}{\sum_{\text{for each explored node } I} trip\_time_i} \quad (1)$$

The routing table in the GA approach consists of a short list of returned agents, every entry corresponds to an evaluated returned agent path. On routing a data packet, the router checks the table for a path that had experienced shortest trip time to the desired destination (third column of Table 3); if such an entry is not found, the entry with the highest fitness, Table 3 column 2, will be selected as the default next node for this data packet. The first two columns in the routing table are used during ranking and replacement of winning chromosomes.

The above constitutes our basic GA-agent approach. In addition, three further concepts are introduced. The first is that of demes. This provides a mechanism for passing useful chromosomes between *neighboring* nodes. To do so, every node will propagate best-case chromosomes to

neighboring nodes every 500 or 700ms (tunable parameter, see "propagate freq" in section 5). Secondly, in order to avoid stagnation in the routing tables, an incremental penalty is applied to each entry of the routing table (see the algorithm "updating routing table & population"). The motivation for such an aging mechanism is to ensure that routing tables remain sensitive to the dynamic nature of the environment (e.g., changes to network topology, network node/link failure, network congestion). Such a mechanism is introduced during updates to routing tables: making every routing table entry a bit smaller in fitness, and a bit longer in trip time, or

$$\text{fitness}_{\text{agent in routing table}} = \text{original fitness} \times c2;$$
$$\text{trip\_time}_{\text{every node of every entry}} = \text{original\_trip\_time} / c2;$$
$$\text{where } c2 \text{ is a constant} \in (0.0, 1.0) \qquad (2)$$

Finally, when initializing the populations of chromosomes at each node, nodes with a higher connectivity naturally represent a larger search problem. Thus, the number of chromosomes per population is initialized in proportion to the square of the number of neighbors.

The algorithm is outlined as follows: (c_1, c_2, and c_3 are constants.)

init

initialize first generation of agents;
#agents = #links$^2 \times c_1$;
string of offsets of an agent - {3, 1, 4, 5, 2, …}
clear routing table;
clear flow pattern stats;
send out half population of individuals;

processing agents

```
if (case of backward agent)
then    if (agent arrives at the source)
        then    if (agent timeout)
                then    (kill agent);
                else    (put into "back" list);
                end if
        else    if (next hop is down)
                then    (kill agent);
                else    (forward to the link)
                end if
        end if
  else  agent records the trip time info;
        retrieve offset from the next unused gene position;
        if (corresponding link is available and no loop caused)
        then    (send the agent to the link);
        else    (randomly [each available link has equal probability] select an available link [without entering a loop]);
        end if
        if (no such link found)
        then    (convert the agent into a backward agent)
                (set the offset to the new value);
                (send agent to the link);
        end if
end if
```

updating routing table & population (once 4 agents return to the same source, i.e. steady state tournament**)**

update the performance table by aging mechanism:
fitness of agent = original fitness × c2;
trip time to every node of every entry = original trip time / c2;
use the fitness function to evaluate the fitness of backward agents;
select the best two agents as parents;
put/update the fitness of parent agents in the routing table;
delete the entries of the worst two agents in the routing table;
use standard crossover and mutation on the parents to generate two children;
put the children into the population;
delete the worst two agents from the population;
if (current time > last clear time + c3)
then (clear flow statistics)
randomly launch 4 agents from the population to explore the network;

routing data packets

```
if (routing table is empty)
then    (randomly choose a link to forward)
else    (search the routing table for the shortest trip time to the desired destination)
        if (no entry found for the desired destination)
        then    (choose fittest entry);
        end if
end if
if (no route is found)
then    (discard the packet)
end if
```

3.1 DATA STRUCTURES

Every agent consists of a string of next hop offsets, and time stamp records. Every router consists of an incoming buffer, a processing buffer (stores a packet at a time), and an outgoing buffer for each neighboring router. For the GA approach, every router has a population of chromosomes, a routing table, a flow pattern statistics table, and a fitness table. The number of chromosomes per population is in direct proportion to the square of number of neighbors. The routing table, which is updated whenever four chromosomes return, consists of current fittest individuals, c.f (1). The flow pattern estimates the popularity of data packets passing through the node, c.f. (2). The fitness table stores the fitness of every chromosome, currently a member of the routing table.

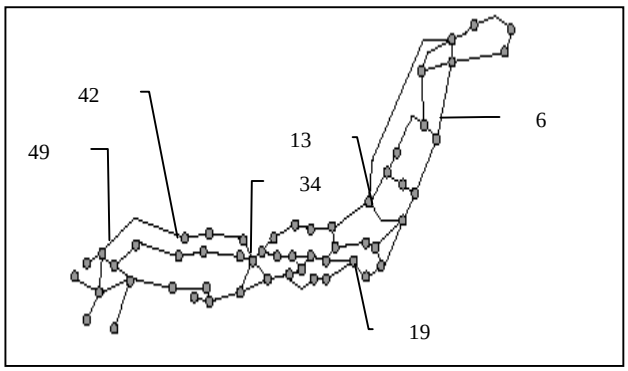


Figure-1: Japanese Backbone - NTTnet (55 nodes)

To simulate and test the GA-agent algorithm, an event driven simulation environment is developed (C++ on UNIX system). Specifically, the Japanese Internet backbone (NTTNET – see figure 1) is modeled, where this represents a narrow long configuration in which the degree of connectivity is low (from 1 to 5), when compared to the US backbone. Hence the Japanese network provides a more demanding configuration for testing routing algorithms, as higher degrees of connectivity lower the possibility of packet loss due to loops, timeouts, i.e., in a narrow long shaped network, once a packet is forwarded in a wrong direction, it might never have the chance to be routed to the desired destination.

4 SIMULATION ENVIRONMENT

The event driven simulation models the network as routers (nodes) and links. Every router has an incoming buffer, a memory space for processing packets, and an outgoing buffer for each link to its neighboring routers. A priority queue is used to store the events. Both AntNet (local routing table information, Table 2) and GA-agent algorithms are simulated under the same environmental conditions. That is, an event generator is used to generate the events, such as new packet time of generation, or routers availability. The following are the parameters used in the simulation,

- Network topology takes the form of the Japanese backbone, figure 1;
- Forward ants are launched every 300ms;
- The AntNet algorithm is given 5 seconds at the beginning of the simulation to converge the initial routing tables, during this period, routing packets (ants or GA-agents) are the only packets traversing the network;
- Data packets are generated by Poisson distribution (mean of 35ms);
- The parameters for the GA based scheme are given as the first 5 rows of tables 4 - 7, where 4 (columns 2 - 5) different GA based agent structures are simulated;
- Any packets, including data packets, are *killed* should they encounter a previously visted node. Given the probabilistic nature of the routing tables this represents a rather harsh constraint, but in doing so is utilized to emphasize the properties of different routing strategies. In addition packets that are routed down links representing a fault condition are distinguished separately as *lost* packets.

The simulation length is 1250s. As a result, 1985536 data packets are generated within 1250s. The queue length is the total number of waiting packets per second, which includes the data packets and the routing packets. In this paper, the routing packets refer to the ants in the AntNet algorithm, and to the GA-agents in the GA approach.

5 RESULTS

On measuring the performance of a routing algorithm, we focus on:

- Network throughput, which is defined as number of data packet bytes successfully received at their destination in a two second window;
- Total time to deliver all the data packets (finish time);
- Number of arrived data packets;
- Number of 'killed' and 'lost' packets;
- Average trip time of arrived data packets.

Two sets of experiments are conducted, in both cases using a series of network scenarios designed to investigate operation under changing network conditions. The first set of experiments investigates parameters associated with the distributed GA. The second of experiments takes one set of these parameters and reduces the degree of exploration/ exploitation (mutation/ crossover respectively).

There are a total of 4 scenarios in each set of experiments, in the first case all routers remain available. The remaining scenarios investigate plasticity of the network by removing different router combinations. First, router 34 is removed at a time step of 500s. From figure 1, it is apparent that router 34 represents a significant node in the topology, although alternative paths certainly exist. In the third scenario, two routers are removed, whereas in scenario four the same two routers are removed but asynchronously.

5.1 PARAMETERIZATION OF DISTRIBUTED GA

In the case of routing using GA-agents, there are six basic parameters,

1. Agents / link^2 – c_1, determines the population of chromosomes per node. The implication being that there are $O(L^2)$ locations in each routing table, where L is the number of neighboring nodes;

2. Aging – c_2, rate by which fitness of individuals currently populating the routing tables decay;
3. Propagate ratio – the number of chromosomes exchanged between populations, expressed as a % of node population size;
4. Propagate freq – rate of exchange of chromosomes between populations;
5. Flow clear freq – c_3, time interval over which data packet destination statistics are collected;
6. Crossover and Mutation – the results in this section utilize maximum crossover and mutation rates in order to encourage continuous investigation of alternative routes. Section 5.2 considers the case of a more classical section of crossover and mutation thresholds.

These are initially selected to enable qualification of the sensitivity to population size, rate of aging etc. and remain the same across all experiments; Tables 4 to 7, columns 2 - 5. Table 8 summarizes the same information for the AntNet algorithm under a 'local' routing table configuration. Thus, Local AntNet utilizes tables of length O(L), significantly less than the GA-agent case.

Table 4 – No Network Failure.

# Agents / link^2	32	32	40	48
Aging	0.8	0.9	0.9	0.9
Propagate ratio	5%	3%	3%	2%
Propagate freq	500ms	500ms	700ms	700ms
Flow clear freq	50s	50s	50s	50s
Finish time (s)	1,252	1,253	1,252	1,267
Arrived Packets (AP) (×1000)	1,619	1,585	1,583	1,560
Avg. trip time for AP (ms)	742	905	678	1,236
# killed packets	366,533	400,351	402,517	385,750
# lost packets	0	0	0	0
# Agents (×10^3)	1,690	1,028	801	475

Table 5 – Router 34 is Down at 500s.

# Agents / link^2	32	32	40	48
Aging	0.8	0.9	0.9	0.9
Propagate ratio	5%	3%	3%	2%
Propagate freq	500ms	500ms	700ms	700ms
Flow clear freq	50s	50s	50s	50s
Finish time (s)	1,417	1,307	1,444	1,494
Arrived Packets (AP) (×1000)	1,346	1,298	1,333	1,373
Avg. trip time for AP (ms)	2,014	2,613	3,156	2,668
# killed packets	617,064	665,188	630,479	590,732
# lost packets	21,922	21,922	21,923	21,918
# Agents (×10^3)	1,801	1,087	966	552

Table 6 – Routers 49 & 13 are Down at 500s.

# Agents / link^2	32	32	40	48
Aging	0.8	0.9	0.9	0.9
Propagate ratio	5%	3%	3%	2%
Propagate freq	500ms	500ms	700ms	700ms
Flow clear freq	50s	50s	50s	50s
Finish time (s)	1,254	1,445	1,258	1,520
Arrived packets (AP) (×1000)	1,317	1,369	1,402	1,504
Avg. trip time for AP (ms)	947	1,301	850	1,759
# killed packets	623,539	571,390	539,747	438,378
# lost packets	44,466	44,882	43,658	43,496
# Agents (×10^3)	1,543	973	754	514

Table 7 – Router 13 is down at 300s, Router 49 is down at 500s, and both are up at 800s.

# Agents / link^2	32	32	40	48
Aging	0.8	0.9	0.9	0.9
Propagate ratio	5%	3%	3%	2%
Propagate freq	500ms	500ms	700ms	700ms
Flow clear freq	50s	50s	50s	50s
Finish time (s)	1,535	1,261	1,496	1,437
Arrived packets (AP) (×10^3)	1,410	1,334	1,441	1,458
Avg. trip time for AP (ms)	2088	1202	470	2018
# killed packets	551,218	627,596	520,989	503,873
# lost packets	23,953	23,426	23,401	23,085
# Agents (×10^3)	1,447	1,043	896	648

Performance is qualified in terms of two basic parameters, time taken for all packets to be received (or lost) and the number of packets successfully received. Naturally, the former should be minimized and the latter maximized. In the case of experiment 1 – no network failures – the time for all packets to be accounted for is essentially the same,

irrespective of parameter or algorithm. An immediate difference is recognized, however, in the number of arrived packets. The AntNet algorithm can only successfully route 55% of those in the GA-agent approach. This observation is repeated across all the remaining scenarios. Moreover, in terms of 'killed' packets this means that less than 50% of the packets in the local version of the AntNet algorithm revisit sites previously encountered.

Table 8 – AntNet with Local Information Only

No network failure	
Finish time (s)	1,267
Arrived packets (AP)	903 ($\times 10^3$)
Avg. trip time of AP (ms)	398
# killed packets	1,082,652
# lost packets	0
# of Ants	218 ($\times 10^3$)
Router 34 down at 500s	
Finish time (s)	1,369
Arrived packets (AP)	813 ($\times 10^3$)
Avg. trip time of AP (ms)	2,899
# killed packets	1,138,860
# lost packets	32,763
# of Ants	219 ($\times 10^3$)
Routers 49 & 13 down at 500s	
Finish time (s)	1,300
Arrived packets (AP)	827 ($\times 10^3$)
Avg. trip time of AP (ms)	1,617
# killed packets	1,114,729
# lost packets	43,682
# of Ants	219 ($\times 10^3$)
Routers 13 down at 300s, Router 49 down at 500s, both up at 800s	
Finish time (s)	1,272
Arrived packets (AP)	863 ($\times 10^3$)
Avg. trip time of AP (ms)	1,254
# killed packets	1,099,283
# lost packets	23,209
# of Ants	219 ($\times 10^3$)

In terms of specific parameter settings, the GA-agent approach appears to consistently route the most packets successfully when the number of agents per link is highest and propagation ratio least. (Investigation of GA-agents without demes, however, performs very badly.) It is also noticed that although a maximum allowable length of 30 genes per individual is permitted, chromosomes never reach this limit. Instead a preference of chromosome lengths of 10 or less genes is found for nodes with a low level of connectivity and 15 to 25 for individuals with a connectivity of 3 or more.

5.2 PARAMETERIZATION OF CROSSOVER AND MUTATION

As a final experiment, one instance of the distributed parameter set is investigated under a classical crossover and mutation rate of 90% crossover and 10% mutation. As identified in section 5.1, lower agent per link counts result in less packets being delivered. Table 9 reports the case of 32 agents/ link, an aging factor of 0.9, a propagation ration of 3% and a frequency of 500ms (column 3 in tables 4 to 7).

On comparison with the same parameterization under 100% crossover and mutation, the number of 'killed' or 'lost' packets decreases by 33% to 8%, and the trip time improves in each scenario other than no network failure. Moreover, the case of 90% crossover and 10% mutation betters all combined 'killed-lost' packet counts of any of the distributed GA parameters investigated in section 5.1. The implication being that more data packets are routed to the destination without either encountering a faulty link or a previously visited node. The principle penalty, however, appears to be an increase in the number of GA-agents introduced. Future work will naturally investigate whether this trend holds for other distributed GA parameterizations (the case of 48 agents per link appears to utilize less GA-agents).

Table 9 – GA-agent with Crossover of 90%, Mutation 10%

No network failure	
Finish time (s)	1,252
Arrived packets (AP)	1,693 ($\times 10^3$)
Avg. trip time of AP (ms)	1,171
# killed packets	292,723
# lost packets	0
# of Agents	961 ($\times 10^3$)
Router 34 down at 500s	
Finish time (s)	1,507
Arrived packets (AP)	1,401 ($\times 10^3$)
Avg. trip time of AP (ms)	356
# killed packets	562,751
# lost packets	21,924
# of Agents	1,170 ($\times 10^3$)

Routers 49 & 13 down at 500s	
Finish time (s)	1,252
Arrived packets (AP)	1,417 ($\times 10^3$)
Avg. trip time of AP (ms)	861
# killed packets	523,673
# lost packets	44,658
# of Agents	1,025 ($\times 10^3$)
Routers 13 down at 300s, Router 49 down at 500s, both up at 800s	
Finish time (s)	1,252
Arrived packets (AP)	1,555 ($\times 10^3$)
Avg. trip time of AP (ms)	1,012
# lost packets	406,840
# killed packets	23,861
# of Ants	1,083 ($\times 10^3$)

6 CONCLUSIONS

As indicated in the introduction, network routing problems force an interesting set of constraints, which present a suitable test-bed for problem solving using co-evolutionary techniques. In this work, we emphasize the case in which routing table features, as well as content, are evolved. Thus, we are not privy to *a priori* knowledge regarding the number of nodes in the network. The AntNet algorithm (Di Caro *et al*., 1998) does not perform efficiently and the GA representation cannot make use of global knowledge of network connectivity, as has been the case in the past (Munetomo *et al*., 1997). Such an environment implies that packets responsible for updating network connectivity requires more autonomy than were previously acknowledged to solve packet switched routing problems. As a first attempt at addressing these problems directly, we utilize a representation that is independent of specific network connectivity patterns and distributed in its operation (multi-population model with chromosomes physically traveling the network). Such a system improves on the AntNet algorithm when constrained to a 'local' table representation, Table 2 (see (Liang *et al*., 2002) for a detailed discussion of AntNet under 'local' and 'global' routing table constraints), or be it whilst utilizing larger routing tables. The principle drawback for the GA-agent is the search efficiency of the ensuing routing table where a search as opposed to an indexing process is now necessary. Future work will expand the interaction between chromosomes to facilitate a more co-evolutionary approach to the development of routing policies and develop a better organization to the routing table structure. Moreover, the relationship between routing table size and performance requires further investigation.

Acknowledgments

A. Nur Zincir-Heywood and Malcolm I. Heywood gratefully acknowledge the support of the Individual Research Grants from the Natural Sciences and Engineering Research Council of Canada.

References

Boyan J.A., Littman M.L., "Packet Routing in Dynamically Routing Networks: A Reinforcement Learning Approach," *Advances in Neural Information Processing Systems*, Volume 6, pp 671-678, 1994.

Corne D.W., Oates M.J., Smith G.D., Telecommunications Optimization: Heuristic and Adaptive Techniques. John Wiley & Sons, isbn 0-471-98855-3, 2000.

Di Caro G., Dorigo M., "AntNet: Distributed Stigmergetic Control for Communications Networks," Journal of Artificial Intelligence Research, 9, pp 317-365, 1998.

Dorigo M., Maniezzo V., Colorni A., "Ant System: Optimization by a Colony of Cooperating Agents," IEEE Transactions on Systems, Man and Cybernetics – B, 26(1) pp 29-41, Feb 1996.

Forouzan B. A., "Data Communications and Networking", Mc-Graw Hill, ISBN 0-07-232204-7, 2001.

Gelenbe E., Xu Z., Seref E., "Cognitive Packet Networks," Proceeding of 11th IEEE International Conference on Tools with Artificial Intelligence, pp 47-54, 1999.

Halabi B., Internet Routing Architectures, Cisco Press, ISBN 1-56205-652-2, 1997.

Heusse M., Snyers D., Guerin S., Kuntz P., "Adaptive Agent-driven Routing and Load Balancing in Communication Networks," Advances in Complex Systems, 1, pp 237-254, 1998.

Liang S., Zincir-Heywood A.N., Heywood M.I., "The Effect of Routing under Local Information using a Social Insect Metaphor," IEEE International Congress on Evolutionary Computation, May 2002.

Maniezzo V., Colorni A., "The Ant System Applied to the Quadratic Assignment Problem," IEEE Transactions on Knowledge and Data Engineering, 11(5), pp 769-778, Sept/ Oct 1999.

Munetomo M., Takai Y., Sato Y., "An Adaptive Network Routing Algorithm Employing Path Genetic Operators," Proceedings of the 7th International Conference on Genetic Algorithms, pp 643-649, 1997.

Tennenhouse D., Smith J., Sincoskie W., Wetherall D., Minden G., "A Survey of Active Network Research," IEEE Communications Magazine, 35(1), pp 80-86, Jan 1997.

Agent Support for Genetic Search in an Immunological Model of Sparse Distributed Memory

Keith E. Mathias and Jason S. Byassee
TRW Systems
16201 E. Centretech Pkwy.
Aurora, CO 80011
keith.mathias, jason.byassee@auc.trw.com

Abstract

This research focuses on agent migration strategies and communication behaviors in a sparse distributed memory implementation based on the human immune system. Evaluation of various agent strategy/behavior combinations is measured in the context of genetic search performance at multiple, independent system nodes. Results indicate that agent behaviors which promote and enhance information exchange between distributed nodes yield the best performance.

1 Introduction

Our research involves a sparse distributed memory (SDM) where the theoretical memory capacity far outweighs the physical memory space (i.e., the ratio of memory cells to items represented is 1 to n, where $n >> 1$). This model is based in part on the human immune system, wherein memory persists in the form of a relatively modest population of antibodies (10^7 to 10^8) with a high affinity to a much greater number (10^{12} to 10^{16}) of possible antigen strains [5]. An effective SDM must develop and maintain a sufficient (with respect to quality) population of memory cells so that associative recall is not only feasible, but efficient.

The memory cell population in this SDM is sparsely distributed in representation space and physically distributed in the execution environment. In this system, mobile software agents circulate a limited number of memory cells between system nodes (Figure 1). Distributed, independent genetic search is leveraged in order to develop a system-wide memory cell population. Emergent behavior at the system level is a result of interactions between simultaneous and independent genetic searches, as well as, local feedback decisions.

Significant work has been performed with respect to agent strategies and enhanced distributed communication performance [3, 4]. This research differs in that we seek to examine the impact of mobile agents with respect to their migration strategies and communication behaviors to improve genetic search performance. Improved genetic search performance in turn, results in a more efficient SDM.

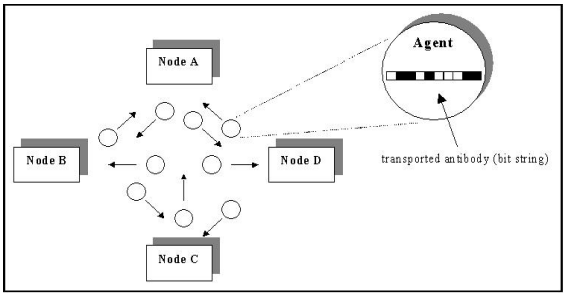


Figure 1: Agent Circulation of Antibodies.

2 Sparse Distributed Memory

This investigation is influenced by previous work that incorporates genetic algorithms in an immune system model to explore pattern recognition [2]. We have modeled the problem space using Hamming space (i.e., bit strings). In training the SDM, the objective is to dynamically develop "immunity" to patterns that are repeatedly re-introduced from a fixed library of patterns. Immunity is achieved by evolving a memory cell population that generalizes to adequately represent a much larger set of random bit string patterns. The random set of bit string patterns that must be matched is known here as the **antigen library**. The system population of memory cells, known as **antibodies**, consists of a small (relative to the size of the antigen library) collection of bit string patterns. Antibody evolution (i.e., system level learning) is a result

of isolated genetic search, local feedback decisions, and the ability of mobile agents to maintain an adequate distribution and diversity of antibodies between system nodes.

2.1 System Operation

This SDM consists of two core operations. The first is genetic search, taking place simultaneously and continuously on each node. The second involves mobile agents that circulate antibodies between the system nodes. Genetic search is used to perform pattern matching at each node, where each node randomly samples from a common antigen library, similar to Forrest, et. al. [2]. When an antigen sample is taken, the resident antibodies in the input queue on each respective node are compared against the sample. If a match is found, the search is complete and the node prepares for a new antigen sample. If a match is not found, the antibody population in the local input queue is used to seed the initial population for genetic search. The sampling of an antigen and searching for a match constitute a **cycle**.

The final population for each independent search includes the solution (i.e., antibody pattern matching the sampled antigen) and antibodies that are similar to the antigen, but not necessarily perfect matches. This provides the opportunity to feed patterns (in the form of antibodies) back into the system that are similar to the current antigen sample. The system antibody population subsequently evolves with representatives that have high affinity for the antigen library.

In this SDM, the mobile agents operate autonomously. From the perspective of each node, agents continuously arrive, deposit antibodies in the input queue, retrieve new antibodies from the output queue and transport them to new nodes (Figure 2). Meanwhile, patterns are continuously sampled from the antigen library as described above. When a sample is taken from the antigen library that must be matched, an initial population of 50 individuals is constructed to start genetic search. To take advantage of the system's learned knowledge, the initial population is comprised of: 1) antibodies taken from the local input queue, 2) copies of antibodies currently waiting in the output queue and 3) mutated copies of antibodies from steps 1 and 2. Copying and mutating antibodies is repeated until the initial population is complete.

In order to bound the size of the antibody population while promoting quality information in the system, we have introduced a survival scoring mechanism based on 1) age and 2) affinity to antigen library samples. This rewards antibodies for survival time (long-term

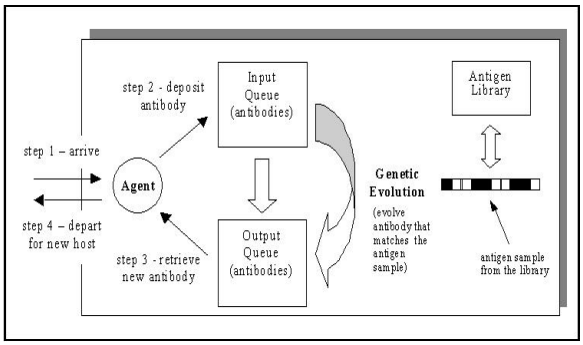


Figure 2: Simultaneous Activities At Each Node.

reward) and for scoring well against the current antigen sample (short-term reward). The survival score is the sum of the *age* and $affinity$ values.

The *age* of an antibody corresponds to the number of nodes that it has visited since creation. A new antibody has an age of zero, and this value is subsequently incremented by one for each new search in which it is used as an initial seed. The $affinity$ is based on the percentage of bits that match antigen samples. This value is initially set to 100, giving newly created antibodies a chance to survive infancy. The value is subsequently decremented at each feedback step. The affinity value is reset to the affinity for the current antigen sample if that score is greater than the current affinity.

At the conclusion of every genetic search on each system node (when a given antigen sample is matched), a competition takes place to determine which antibodies are fed back into the system. At each competition, individuals in the antibody population that were in the input queue prior to genetic search (i.e., seeds) are compared with individuals from the final search population. The highest scoring antibodies are fed back into the system, and the remainder are discarded. A search *feedback threshold* allows individuals from the final search population that are not exact matches to be competitive in the feedback competition.

2.2 Pattern Matching Application

These experiments were designed to examine the impact of agent behavior and agent mobility strategies on the performance of this SDM. Performance in this context is measured with respect to the work necessary to discover the antibody strings that match the antigens sampled at the nodes in the system over the course of time. The antibody population consists of bit strings that are used to seed the population at local nodes for genetic search in order to match the patterns sampled from the antigen library.

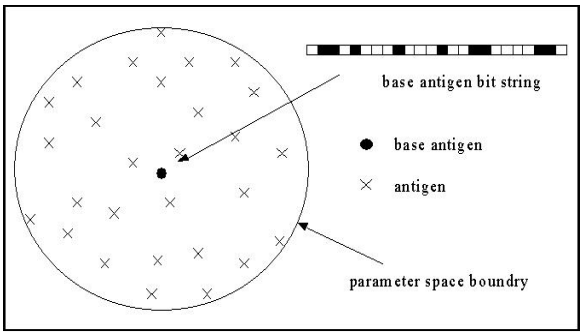


Figure 3: Antigen Library Distribution.

In each of these experiments, a parameter space is defined surrounding a randomly generated bit string, known as the **base antigen string**. An antigen library is generated in a binomial distribution around the base antigen, bounded by a given Hamming distance (Figure 3). The distribution reflects the frequency of occurrence of samples based on their Hamming distance from the base antigen. Thus, bit strings that are closer to the base antigen string are included in the library less frequently than those that are farther from the base string. This distribution is similar to that of B and T cell clones as modeled by Smith, et. al. [6], in simulations of immune system models.

In training the SDM, success is measured by the ability to generate antibodies quickly (i.e., few trials) that match strings sampled from the antigen library. A trivial solution to this problem is to generate one or more antibodies that match each antigen sampled (i.e., specialists). However, in a sparse distributed memory, the address space can be orders of magnitude larger than the instantiated address locations[6]. We used an antigen to antibody ratio in this system of 10:1. This means that, on average, one antibody must represent ten antigens. The hypothesis is that good system-wide performance requires an antibody circulation scheme that promotes an antibody population comprised of "generalist" antibodies (as opposed to specialists).

2.3 Migration Strategies and Communication Behaviors

Within the context of the SDM described, we have devised three agent migration strategies and three agent communication behaviors for investigation. This results in nine agent strategy/behavior combinations. The three agent migration strategies are as follows:

1. **Random Migration** - In this migration strategy, agents circulate antibodies by moving at random between system nodes.
2. **Directed Migration** - This migration strategy is intended to promote maximum diversity by correlating agent movement with the specific antibody that is being transported. Each node maintains a *most recently sent queue* that consists of a single entry representing every other node in the system. Each entry associates a node ID with the last antibody transported by an agent to the respective remote node from the local node. When an agent retrieves an antibody to transport, the Hamming distance between that antibody and every other entry in the queue is measured. The agent selects the destination node based on the entry that is furthest away in Hamming space.
3. **Cyclic Migration** - Cyclic migration agents move in a fixed pattern between system nodes. Each agent generates a random itinerary upon creation that includes a single visit to each node (i.e., Hamiltonian cycle).

The three communication behaviors are as follows:

1. **Always Communicate** - This is a simple behavior that requires no thinking, or decision process, on behalf of the agent. The agent simply deposits the transported antibody into the input queue of the node on which it arrives.
2. **Just In Time (JIT) Communication** - Agents search for a host that has just sampled an antigen from the library and is ready to begin genetic search. Agents continue to move until a host in this state is found, and then they deposit the transported antibody, "just in time" to begin genetic search.
3. **Load Balanced Communication** - Agents have a tendency to move away from other agents when exhibiting a load balancing communication behavior. This behavior forces agents to move away from "busy" nodes, thereby evenly distributing the antibody population among the system nodes.

2.4 CHC Algorithm

There are numerous genetic search approaches that could be used in the context of this SDM. We have chosen to incorporate the CHC adaptive search algorithm for antibody evolution at the local nodes. The CHC adaptive search algorithm [1] is a generational genetic algorithm that has been shown to yield very good performance for optimizing a wide variety of test problems and requires no parameter tuning [7]. Mating in CHC is performed by randomly pairing parents

and applying the HUX crossover operator. HUX exchanges exactly half of the bits that differ between the mates. Crossover is only performed if the differences between mates is greater than a threshold that is dynamically adjusted during the search process. This is known as *incest prevention* and serves to slow genetic convergence appreciably. Selection is performed using the $(\mu + \lambda)$ strategy, preserving the best N individuals from the child and parent populations, where N is the population size. When the population does converge, the search process is restarted via *cataclysmic mutation*. The population is filled with copies of the best individual and then 35% of the bits in all but one individual are complemented and search is restarted.

3 Experimental Conditions

For these experiments, the three migration strategies were paired with all three of the communication behaviors. These strategy/behavior combinations were also compared against the performance of a control strategy wherein no agents were used (i.e., no communication between nodes was possible). Table 1 identifies the system parameter values used for all experiments.[1] To emulate a sparse distributed memory, we maintained a 10:1 antigen to antibody ratio. To adequately sample the antigen pool, each simulation consisted of 5,000 cycles[2] at each of four nodes. This allows examination of a full range of behavior without spurious states (due to early termination) and an even sampling distribution of the patterns from the antigen library.

4 Results

All nine of the agent migration strategy/ communication behavior combinations were simulated for 30 independent runs. Combining all agent communication behaviors with all of the agent migration strategies provides a complete factorial design, supporting analysis of variance (ANOVA) testing. This allowed us to determine if any of the migration strategies or communication behaviors resulted in a statistically significant performance advantage or disadvantage (i.e., significant main effect).

[1]To support a fair comparison between the migration strategy/communication behavior pairs, the parameter values for the *total number of agents*, *antibody mutation rate*, and *search feedback threshold* were established via a separate search using a meta-GA. The purpose of the meta-GA was to find a good, if not optimal, set of parameter values for operation of this system.

[2]Genetic search is used to find an antibody that matches an antigen sample at each cycle except when a perfect match resides in the initial genetic population.

Parameter	Value
System Nodes	4
Antigen Library Size	320
System Antibody Population	32
Antibody String Length (bits)	32
Parameter Radius (bits)	5
Cycles (antigen samples/node)	5000
Time Between Search at each Node (msec)	20
Total Agents	24
Antibody Mutation Rate (bits)	3
Search Feedback Threshold (bits)	5

Table 1: System Operational Parameters.

Table 2 shows the average number of trials (and standard error of the means - SEM) to match antigen samples for each of the strategy/behavior combinations. The random agent migration strategy paired with the always communicate behavior expended the least amount of work (i.e, fewest trials), on average, to discover the antibodies that match the antigens sampled in the simulations. However, this performance advantage is only statistically significantly better than a few of the other cells in Table 2[3] (particularly the directed/JIT combination). The average trials to match the antigens sampled using the cyclic agent migration strategy are significantly worse than any of the other strategy/behavior combinations. ANOVA tests confirm this fact as a significant main effect.

Comm. Behavior	Migration Strategy		
	Random	Directed	Cyclic
Always	283.1 (0.73)	285.1 (1.19)	300.8 (1.69)
JIT	285.5 (0.66)	286.2 (0.72)	301.1 (1.87)
Load Bal	284.3 (1.46)	287.1 (2.61)	306.2 (2.13)

Table 2: Average Trials to Match Antigen Samples. When no agents are present, 298.7 trials (SEM = 1.49) are needed to match the pattern, on average.

The cyclic migration strategy combined with the load balancing communication behavior results in the worst performance, relative to all other strategy/behavior combinations. This performance is 10 standard errors worse than the random migration, always communicate runs and more than 2 standard errors worse than the experimental runs with the cyclic/JIT implementation. In fact, it is even inferior to the performance of simulations where no agents were present in the system. The average trials to match the antigens sampled when no agents are present (i.e., antibodies are not circulated) is 298.7 (SEM = 1.49).

[3]This may be due to the stochastic nature of the simulations contributing more noise than the variance between the strategy/behavior combinations.

It is important to note that the system learns and performs significantly better than genetic search on an equivalent problem when a random initial population is used. For example, on average, CHC solves a 32-bit one-max[4] problem in 504 trials (SEM = 9.12), when beginning with a random initial population.

4.1 An Emergent Behavior

Although the goal of genetic search at each local node is to match antigen library samples, the search efforts combined with the local survival decisions result in a globally emergent behavior. The communication of information via agents consistently resulted in an interesting phenomenon, notably the discovery and propagation of the string pattern used to create the antigen library (i.e., the base antigen string). Table 3 shows the average percentage of the system's final antibody population occupied by copies of strings matching the base antigen for each respective strategy/behavior experiment. This is referred to as the **saturation rate**.

Comm. Behavior	Migration Strategy		
	Random	Directed	Cyclic
Always	100.0% (0.00)	100.0% (0.00)	89.7% (1.04)
JIT	100.0% (0.00)	100.0% (0.00)	89.1% (1.35)
Load Bal	96.1% (0.74)	97.2% (0.68)	82.9% (3.40)

Table 3: Antibody Population Saturation Rate. When no agents are present, the average saturation rate is 91.1% (SEM = 0.50).

ANOVA testing confirms the trends evident by visual inspection as significant. First, the load-balancing communication behavior does not allow the base antigen to saturate the system, regardless of the agent migration strategy employed. The cyclic agent migration strategy also prevents the base antigen string from saturating the antibody population. This seems obvious in hindsight as the cyclic migration strategy is the most restrictive of the migration strategies. Visitation by a given agent is not equally likely at all nodes at each time step for this migration strategy. This restriction is so severe in fact, that the results were comparable to runs where no agents were present in the system. While simulations using no agents did discover this base antigen string, the simulations yielded an average saturation rate of 91.1% (SEM = 0.50).

The discovery of antibodies that match the base antigen string cannot be a result of searches in which the base antigen is sampled from the antigen library. Antibodies fed back to the system must meet or exceed a *feedback threshold* of five bits.

In searching for strings to match samples from the antigen library, each node contributes strings to the system antibody competition that have a large number of bits in common with the base antigen. This may or may not be sufficient for a given antibody to survive the feedback competition and be propagated to other nodes. However, those strings that are close to the base antigen string in Hamming distance will also likely score well against other antigens, if kept in the system antibody population. This causes the system antibody population to accumulate alleles in common with the base antigen string. When the antibody population is viewed as a probability vector that represents the percentage of 0- or 1-bits at each locus over the strings in the antibody population, this vector will approximate the base antigen string more accurately over time.

Eventually, an antibody matching the base antigen is a by-product of a search for another antigen library sample. Antibody copies of the base antigen string will likely perform well in the feedback competitions at each node, and chances of survival in the system will be better than average. After surviving in the antibody population for several cycles, the age weighting guarantees future survival.

The base string is very rarely useful in exactly matching any string in the antigen library (a 1 in 320 chance), yet this string serves as a good seed string for the genetic search. The discovery of the base string may or may not be an optimal system-wide strategy for learning how best to reduce the number of trials required to evolve an antibody that matches an antigen sample. For example, the discovery of four antibodies that divide the antigen library into equally sized attraction basins, based on Hamming distance, might work as well as, or better than, a single generalist. Regardless, the discovery of the base string is an interesting example of local behavior that facilitates emergent global behavior.

The best performances shown in Table 2 generally correspond with complete saturation (Table 3), yet there is not a perfect correlation. For example, the load balancing/random migration implementation performs quite well, but does not exhibit complete saturation. Therefore, saturation of the antibody population with the base antigen must not be the only factor in obtaining good performance.

[4]A one-max problem is equivalent to finding a matching bit-string using an evaluation score that reports the number (or percentage) of bits matching another pattern.

4.2 Propagation of Information

To further explore the correlation between the propagation of quality information and the efficiency of discovering antibody/antigen matches, we measured the average number of cycles required to: 1) discover the base string, 2) propagate the base string to all nodes *after* it has been discovered, and 3) saturate the system after a copy of the base string has been seen at all nodes. Table 4 shows these results.[5]

Comm. Behavior	Migration Strategy		
	Random	Directed	Cyclic
Avg. Cycles to Discover Base String			
Always	138.4 (24.5)	234.4 (51.7)	123.4 (19.9)
JIT	147.2 (23.6)	153.0 (26.6)	182.3 (35.7)
Load Bal	195.4 (37.2)	153.6 (32.4)	123.5 (20.9)
Avg. Additional Cycles to Circulate Base String to All Nodes			
Always	36.4 (12.4)	60.4 (22.1)	818.7 (77.7)
JIT	35.6 (12.4)	58.8 (17.8)	1018.9 (135.1)
Load Bal	19.4 (4.3)	89.4 (31.5)	1278.1 (130.3)
Avg. Additional Cycles to Saturate With Base String			
Always	1083.9 (154.4)	1106.6 (178.0)	*3906 (0)
JIT	865.1 (86.8)	973.2 (107.4)	*3798.5 (831.5)
Load Bal	*2944.5(410.3)	*2943.4 (378.5)	*2013 (0)

Table 4: Average Cycles (SEM) to Discover, Circulate and Saturate the Antibody Population.

ANOVA testing shows that there is no significant main effect in the time taken to *discover* the base antigen by any of the strategy/behavior combinations. Surprisingly, the average cycles for the experimental runs using a cyclic agent migration strategy and the always and load-balancing communication behaviors are better than the other strategies at discovering the base string (but not significantly so in most cases, due to large SEM values). The no agent strategy required, an average of 163 cycles (SEM=19) to discover the base string. This is comparable with most cells in Table 4.

There is a significant main effect seen in the number of cycles required to propagate the base string to all of the nodes after it has been discovered. In fact, it is at this stage of the simulation that those strategy/behavior combinations that incorporate the cyclic agent migration strategy experience a significant disadvantage, as compared to the other strategy/behavior combinations. In fact, the number of cycles needed by the cyclic agent migration strategy to propagate the base string to all other nodes after discovery is comparable with having no agents in the system. On average, the SDM runs where no agents are employed require 1158 cycles (SEM=102) after the initial discovery of the base string, until all nodes have independently discovered the base string.

There is also a weak main effect that indicates that the load-balancing behavior is slower at propagating the base string to all nodes after discovery than either the always or JIT communication behaviors. However, this trend is to be treated carefully, as there is an obvious exception. The random migration strategy that incorporates the load-balancing behavior appears to be considerably faster at propagating the base string among all of the nodes. We performed several repetitions of the complete factorial design and this was the only occurrence of this rapid propagation of information (while all other trends were verified).

The ANOVA tests could not be performed for the average number of cycles between complete circulation and saturation due to the fact that all 30 experimental runs for every strategy/behavior combination did not saturate. However, it can be observed that the random and directed strategies that use the load-balancing behavior do not propagate the base string nearly as well as when the always and JIT communication behaviors were employed.

4.3 Performance at Various Stages of the Simulation

There is an obvious difference in the ability of the strategy/behavior combinations to propagate information (although that information does not always appear to expedite search speed). It seemed prudent to test the hypothesis that the discovery of the base string does in fact affect the number of trials to match an antigen. Table 5 shows the average number of trials (and SEM) required to match an antigen during the stages relative to: 1) discovering the base string, 2) propagating the base string to all nodes after the first discovery, 3) between circulating the base to all nodes and saturation occurring, and 4) after saturation.[5]

ANOVA tests show that there is indeed a significant main effect where the discovery of the base antigen reduces the average number of trials required to match a sampled antigen. This holds true for all strategy/behavior combinations but could not be confirmed for the final two stages of simulation (i.e., after circulation and after saturation), since all runs did not saturate. The cyclic migration strategies performed consistently worse than the random and directed migration strategies, although it is not statistically significant. Therefore, the average cycles for the cyclic agent migration strategy between base string circulation among all nodes and population saturation (Table 4) must account for the significant performance

[5]The * indicates that all 30 runs did not saturate. Average cycles reported, include only those runs that did saturate.

Comm. Behavior	Migration Strategy		
	Random	Directed	Cyclic
Stage 1 - Prior to Base String Discovery			
Always	380.0 (5.46)	371.8 (3.96)	392.5 (9.34)
JIT	375.1 (4.60)	381.8 (4.79)	373.7 (2.67)
Load Bal	371.5 (2.98)	390.0 (9.98)	385.7 (5.63)
Stage 2 - Between Base String Discovery & Circulation			
Always	346.7 (2.71)	344.6 (2.59)	338.8 (2.51)
JIT	350.6 (2.95)	355.9 (3.19)	337.9 (2.60)
Load Bal	346.3 (2.72)	353.4 (3.18)	339.2 (2.19)
Stage 3 - Between Base String Circulation & Saturation			
Always	*291.5 (2.70)	291.0 (2.77)	*278.3 (0.0)
JIT	290.5 (1.92)	292.7 (2.20)	*279.1 (0.7)
Load Bal	*282.4 (2.66)	*284.1 (1.33)	*282.1 (0.0)
Stage 4 - After Saturation			
Always	*278.2 (0.33)	278.3 (0.29)	N/A
JIT	280.8 (0.28)	280.2 (0.30)	N/A
Load Bal	*276.0 (0.47)	*275.3 (0.61)	*274.6 (0.0)

Table 5: Average Trials to Match Antigen Samples During Critical Stages of Simulation.

differences observed in Table 5. This is also consistent with the infrequent saturation rates exhibited by the cyclic migration strategy.

Figure 4 shows the number of trials required to match an antigen for the first 1,500 samples of the 5,000 cycle simulation at one of the four nodes for a single representative run. Trials are shown on the Y-axis while cycles are shown on the X-axis. The open circles indicate the trials required to match an antigen during a particular cycle, and the black line represents the running average (lag = 100). The base string is first discovered at cycle 259. The trials to discover a match for the antigen samples begins to decrease at this point. The running average reaches a low of approximately 250 trials by cycle 410, where the system antibody population saturates with the base string.

4.3.1 The Effects of Seeding Genetic Search in the SDM Simulation

An unusual behavior observed in Figure 4 is the occurrence of searches that expend two to three times the normal number of trials to find an antibody/antigen match. This is indicative of seeding the initial population for the CHC search in a biased manner, risking the incidence where the correct allele is not present in any member of the initial population. Since CHC does not employ mutation, except at divergences, the search will converge to an antibody string that does not match the antigen sample, and hence cataclysmic mutation will be performed to restart the search. Such an event can significantly impact the number of trials necessary to find the matching string.

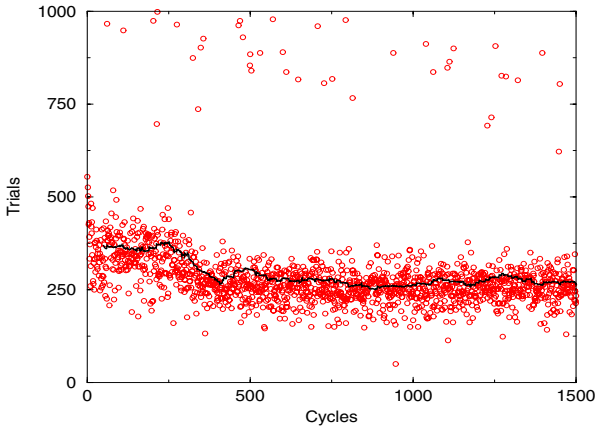


Figure 4: Search Profile At A Single Node For First 1,500 Cycles.

This phenomenon occurs here due to the seeding procedure. Each genetic search is started by seeding the population with three-bit mutations of the antibodies available in the input queue, where each antibody in the input queue seeds an equal fraction of the genetic population. The expected difference between the base string and an antigen sample is five bits. Hence, the expected difference between antibodies that are fed back from the genetic search (i.e., at least Hamming distance five from the antigen just matched) and a another antigen sample is ten bits. Therefore, the occurrence of restarts is not unexpected. It is important then, that genetic meta-search was used to discover the seeded mutation value (i.e., three bits) as opposed to arbitrary determination.

5 Conclusions

Evidence from this study illustrates that seeding the initial population with a single "generalist" pattern can expedite genetic search for other related patterns. In this context, a generalist pattern can form an effective sparse representation for a library of patterns. This SDM learns that a good strategy for reducing the work necessary to match antigen library samples is to evolve and propagate antibodies matching the base antigen string. Performance analysis reveals that providing feedback from the final population used in genetic search is sufficient to discover such a generalist, even when the genetic material does not contain precise matches for the antigen samples.

It is clear that the use of mobile agents to circulate genetic material between nodes expedites the discovery and propagation of the base antigen string. Without agents to circulate the information, each node must

discover the base string independently. The number of trials required to match an antigen is significantly reduced when the base antigen is discovered and its representation (antibody copies) is shared in the system via mobile agents. This is evident from comparing the performances of the simulations using random and directed migration strategies with the performances of simulations implementing cyclic migration, where communication is hindered, or those containing no agents, where communication is non-existent. Simulations using the cyclic agent migration strategy are unable to propagate the base string throughout the system and do not consistently saturate the antibody population.

Additional analysis of system behavior provides insight into the operational dynamics of each agent communication behavior. Although not impacting total trials, the load balancing behavior did not saturate the population in all instances. The relatively large number of cycles required for this communication behavior to saturate the population with copies of the antibody matching the base antigen (Table 4) gives rise to the hypothesis that agents implementing this behavior may be hiding quality material while "looking" for non-busy nodes.

On average, the JIT communication behavior provides more antibodies to begin each search. This behavior did in fact expedite the antibody population saturation, as evident from the number of cycles to saturate the population with the base string (Table 4), although it did not seem to significantly impact the performance metric used in the experiments. Examining metrics beyond total average trials suggests that additional experiments run with different performance criteria (such as reducing the number of cycles per node) could very well serve as a significant discriminator among communication behaviors.

The directed agent migration strategy was designed to promote maximum antibody diversity in the system. This objective was not realized with respect to improved genetic search performance (Table 2). In fact, directed migration did not perform quite as well as random migration during several simulation stages (Tables 4 and 5). We surmise that *near* real-time knowledge (as opposed to real-time) contained in the *most recently sent queue* mitigates the anticipated advantage of antibody diversity promotion. This is a result of the decentralized implementation, where an instantaneous global snapshot of node state is not available.

Thorough study of agent behaviors and migration strategies is a valuable performance analysis exercise. This investigation illustrates that various communication implementations can yield surprising results. A restrictive agent strategy (cyclic migration), conducive to uneven visitation, performed worse than simulations using no agents. Agent implementations employing more complex strategies and behaviors (such as those that are based on current system state or require coordination) are not always performance leaders. Our results indicate that a greater degree of agent autonomy, where agents make simple, independent decisions, facilitates expedited genetic search that improves sparse distributed memory performance.

References

[1] Larry Eshelman. The CHC Adaptive Search Algorithm. How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 265–283. Morgan Kaufmann, 1991.

[2] Stephanie Forrest, Robert Smith, Brenda Javornik, and Alan Perelson. Using genetic algorithms to explore pattern recognition in the immune system. *Journal of Evolutionary Computation*, 1(3):191–211, 1993.

[3] Ashraf Iqbal, Joachim Baumann, and Markus Straßer. Efficient algorithms to find optimal agent migration strategies. Technical Report 1985/05, Universitat Stuttgart Fakultat Informatik, Bericht Nr., 1998.

[4] Frederick Knabe. Performance oriented implementation strategies for a mobile agent language. In *Lecture Notes in Computer Science Series*, pages 229–243. Springer-Verlag, 1997.

[5] Derek Smith, Stephanie Forrest, and Alan Perelson. Immunological Memory is Associative. In Dipankar Dasgupta, editor, *Artificial Immune Systems and Their Applications*, pages 105–112. Springer, 1998.

[6] Derek Smith, Stephanie Forrest, Alan Perelson, and David Ackley. Using lazy evaluation to simulate realistic-size repertoires in models of the immune system. *Bulletin of Mathematical Biology*, 60:647–658, 1997.

[7] Darrell Whitley, Keith Mathias, Soraya Rana, and John Dzubera. Building Better Test Functions. In L. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1995.

Studies on the Dynamics of Ant Colony Optimization Algorithms

Daniel Merkle
Institute AIFB
University of Karlsruhe
D-76128 Karlsruhe, Germany
merkle@aifb.uni-karlsruhe.de

Martin Middendorf
Computer Science Group
Catholic University of Eichstätt-Ingolstadt
D-85072 Eichstätt, Germany
martin.middendorf@ku-eichstaett.de

Abstract

A deterministic model for Ant Colony Optimization (ACO) algorithms is proposed and used to study the dynamics of ACO. The model is based on the average expected behaviour of ants. The behaviour of ACO algorithms and the model are analysed for certain types of permutation problems. It is shown numerically that decisions of the ants are influenced in an intriguing way by the properties of the pheromone matrix. This explains why ACO algorithms show a complex dynamic behaviour. Simulations are done to compare the behaviour of the ACO model with the ACO algorithm. The results show that the model describes essential features of the dynamics of ACO algorithms.

1 INTRODUCTION

Ant Colony Optimization (ACO) has been applied successfully to several optimization problems (ACO was proposed in [1, 2]). Since ACO algorithms are based on sequences of random decisions of artificial ants which are usually not independent it is difficult to analyze the behaviour of ACO algorithms theoretically. Except from convergence proofs for types of ACO algorithms with a strong elite principle [3, 4, 12] not much theoretical work has been done. Usually ant algorithms have been tested on benchmark problems or real world problems. In this paper we propose and analyze a deterministic model for ACO algorithms and use it to derive exact results on optimization problems with a simple structure. The analytical results are complemented with empirical tests to compare computations done with the ACO model with test runs of the ACO algorithm.

Modelling has been done in the field of genetic algorithms (GAs) by several authors in order to better understand GAs behaviour. One line of modelling is to use an infinite population which is often easier to handle than a finite population since many properties of an infinite population do not fluctuate due to few random events [13, 14]. Another method is to characterize the population by few parameters (e.g., mean and variance of the fitness distribution of the population) which capture important aspects of the population instead of dealing with a concrete population (e.g., [9–11]). Mostly in these studies, GAs are modelled on problems which are simple but have some characteristic features of more complicated and real-world problems (e.g., "Royal Road" functions [8]).

The approach used in this paper is to define a deterministic model for ACO that is based on the expected decisions of the ants. In the model the pheromone update in every iteration is done by adding for each pheromone value the expected update of a random generation of ants.

In Section 2, we describe the permutation problems that are used in this paper. The ACO algorithm is described in Section 3 and the ACO model is defined in Section 4. In Section 5, we discuss how to apply the model to permutation problems. A fixed point analysis of pheromone matrices is done in Section 6. In Section 7, we analyze the dynamic behaviour of the ACO model. Simulation results are described in Section 8 and conclusions are given in Section 9.

2 PERMUTATION PROBLEMS

Although the general approach of our ACO model does not depend on a specific type of optimzation problems we give a more elaborated description only for permutation problems. They are also used as test problems. In particular, we use the following type of permutation

problems. Given are n items $1, 2, \ldots, n$ and an $n \times n$ cost matrix $C = [c(ij)]$ with integer costs $c(i,j) \geq 0$. Let $\mathcal{P}_n$ be the set of permutations of $(1, 2, \ldots, n)$. For a permutation $\pi \in \mathcal{P}_n$ let $c(\pi) = \sum_{i=1}^{n} c(i, \pi(i))$ be the cost of the permutation. Let $\mathcal{C} := \{c(\pi) \mid \pi \in \mathcal{P}_n\}$ be the set of possible values of the cost function. The problem is to find a permutation $\pi \in \mathcal{P}_n$ of the n items that has minimal costs, i.e., a permutation with $c(\pi) = \min\{c(\pi') \mid \pi' \in \mathcal{P}_n\}$.

3 ACO ALGORITHM

An ACO algorithm consists of several iterations where in every iteration each of m ants constructs a solution for the optimization problem. It has to be mentioned that we can not consider here all the variants and improvements that have been proposed in recent years for ACO.

For the construction of a solution (here a permutation) every ant selects the items in the permutation one after the other. For the selection of an item the ant uses pheromone information which stems from former ants that have found good solutions. The pheromone information, denoted by τ_{ij}, is an indicator of how good it seems to have item j at place i of the permutation. The matrix $(\tau_{ij})_{i,j \in [1:n]}$ of pheromone values is called the pheromone matrix. In addition an ant may also use problem specific heuristic information. But since we want to study ACO algorithms in general and not for some specific problem we do not consider heuristics in this paper.

The next item is chosen by an ant from the set $\mathcal{S}$ of items, that have not been placed so far, according to the following probability distribution (e.g. [2]) that depends on the pheromone values in row i of the pheromone matrix: $p_{ij} = \tau_{ij} / \sum_{h \in \mathcal{S}} \tau_{ih}$, $j \in \mathcal{S}$.

Note that alternative methods where the ants do not consider only the local pheromone values have also been proposed [5, 7]. Before the pheromone update is done a certain percentage of the old pheromone evaporates according to the formula $\tau_{ij} = (1-\rho) \cdot \tau_{ij}$. Parameter ρ allows to determine how strongly old pheromone influences future decisions. Then, for every item j of the best permutation found so far some amount Δ of pheromone is added to element τ_{ij} of the pheromone matrix (i is the place of item j). The algorithm stops when some stopping criterion is met, e.g. a certain number of generations has been done. For ease of description we assume that the sum of the pheromone values in every row and every column of the matrix is always one, i.e., $\sum_{i=1}^{n} \tau_{ij} = 1$ for $j \in [1:n]$ and $\sum_{j=1}^{n} \tau_{ij} = 1$ for $i \in [1:n]$ and $\Delta = \rho$.

4 ACO MODEL

In the proposed ACO model the pheromone update of a generation of ants is done by adding to each pheromone value the expected update value. This means that the effect of an individual ant in a run is averaged out. Since the update values in the ACO algorithm are always only zero or $\Delta = \rho$ the ACO model can only approximate the average behaviour of an ACO algorithm over more than one generation.

In order to determine the expected update for a random generation of ants the probabilities for the various decisions of the ants have to be determined. Let $M = (\tau_{ij})$ be a pheromone matrix and let σ_{ij} be the probability that a random ant selects item j for place i. Clearly, this selection probability can be computed as described in the following. Let $\mathcal{P}_n$ be the set of possible solutions, i.e. the set of permutations of $(1, 2, \ldots, n)$. The probability to select a solution $\pi \in \mathcal{P}_n$ is

$$\sigma_\pi = \prod_{i=1}^{n} \frac{\tau_{i,\pi(i)}}{\sum_{j=i}^{n} \tau_{i,\pi(j)}} \tag{1}$$

The probability that item i is put on place j is

$$\sigma_{ij} = \sum_{\pi \in \mathcal{P}_n} \sigma_\pi \cdot g(\pi, i, j)$$

where $g(\pi, i, j) = 1$ if $\pi(i) = j$ (otherwise $g(\pi, i, j) = 0$).

Given a permutation problem P with corresponding cost matrix and pheromone matrix let $\sigma_{ij}^{(m)}$ be the probability that the best of m ants in a generation selects item j for place i. Let $\mathcal{P}_{min}(P, \pi_1, \ldots, \pi_m)$ be the subset of permutations of $\{\pi_1, \ldots, \pi_m\}$ with minimal costs, i.e., $\mathcal{P}_{min}(P, \pi_1, \ldots, \pi_m) = \{\pi_i, i \in [1:m] \mid c(\pi_i) = \min\{c(\pi_j) \mid j \in [1:m]\}\}$. Probability $\sigma_{ij}^{(m)}$ can be computed by

$$\sigma_{ij}^{(m)} = \sum_{(\pi_1, \ldots, \pi_m), \pi_i \in \mathcal{P}_n} \left(\prod_{k=1}^{m} \sigma_{\pi_k}\right) \cdot g(\pi_1, \ldots, \pi_m, i, j) \tag{2}$$

where $g(\pi_1, \ldots, \pi_m, i, j)$ equals

$$\frac{|\{\pi \in \mathcal{P}_{min}(P, \pi_1, \ldots, \pi_m) \mid \pi(i) = j\}|}{|\mathcal{P}_{min}(P, \pi_1, \ldots, \pi_m)|} \tag{3}$$

At the end of a generation the pheromone update is done in the ACO model by $\tau_{ij} = (1-\rho) \cdot \tau_{ij} + \rho \cdot \sigma_{ij}^{(m)}$.

In the following an alternative way to compute the selection probabilities for the best of m ants is described. Let $\mathcal{C}$ be the set of possible cost values for a permutation or in other words the set of possible solution qualities. Let $\xi^{(m,x)}$ be the probability that the best of m ants in a generation finds a solution with quality $x \in \mathcal{C}$. Let $\omega_{ij}^{(x)}$ be the probability that an ant which found a solution with quality $x \in \mathcal{C}$ has selected item i for place j. Then

$$\sigma_{ij}^{(m)} = \sum_{x \in \mathcal{C}} \xi^{(m,x)} \cdot \omega_{ij}^{(x)} \tag{4}$$

An interesting aspect of this formula is that the pheromone update that is performed at the end of an iteration is obtained as a weighted sum over the possible solution qualities. For each (possible) solution quality the update value is determined by the probabilities for the decisions of a single ant when it chooses between all possible solutions with that same quality. The effect of the number m of ants is only that the weight of the different qualities in this sum changes. The more ants per iteration, the higher becomes the weight of the optimal quality.

We now consider a variant of the above situation that is needed in the next section. We introduce the concept of malus values. It is assumed that some ants in an iteration receive a malus value. This value is added to the cost of the permutation they found. An ant with a malus is allowed to update only when the cost of its solution plus the malus is better than the solution of every other ant plus its malus (in case it has a malus). Formally, for $i \in [1:m]$ let $d_i \geq 0$ be the malus of ant i. We always assume that ant 1 has no malus, i.e, $d_1 = 0$. Let $\xi^{m;x;d_2,\ldots,d_m}$ be the probability that the best of m ants where ant $i \in [1:m]$ has a malus d_i has found a solution of quality $x \in \mathcal{C}$. Then

$$\sigma_{ij}^{(m;d_2,\ldots,d_m)} = \sum_{x \in \mathcal{C}} \xi^{(m;x;d_2,\ldots,d_m)} \cdot \omega_{ij}^{(x)} \tag{5}$$

5 ACO MODEL FOR RESTRICTED PERMUTATION PROBLEMS

Many real world problems consist of subproblems that are more or less independent from each other. In order to study the behaviour of ACO algorithms on such problems we model this in an idealized way. We consider restricted permutation problems which consist of several small independent problems. Define for a permutation problem P of size n_0 a restricted permutation problem P^q that consists of q independent instances of P. Formally, let $C_1, C_2, \ldots, C_q$ be the cost matrices of q instances of P and denote by $c_{ij}^{(l)}$ element (i,j) of matrix C_l, $l \in [1:q]$. Then for problem P^q the item $(l-1)\cdot n_0 + j$, $l \in [1:q]$, $j \in [1:n_0]$ can be placed only at places $(l-1)\cdot n_0 + 1$, $(l-1)\cdot n_0 + 2$, $\ldots$, $(l-1)\cdot n_0 + n_0$. The cost to place item $(l-1)\cdot n_0 + j$ at place $(l-1)\cdot n_0 + h$ is c_{hj}^{l}. Let C be the corresponding cost matrix of the instance of problem P^q where $c_{ij} = \infty$ when j is not of the form $(l-1)\cdot n_0 + h$. Note, that our definition of restricted permutation problems does not allow an ant to make a decision with cost ∞. We call P the elementary subproblem of P^q and the q instances of P that form an instance of P^q the elementary subinstances of P^q. We consider here only the case that all cost matrices $C_1, C_2, \ldots, C_q$ are equal, i.e. $C = C_1 = C_2 = \ldots = C_q$ for some cost matrix C. Then P^q is called homogeneous restricted permutation problem and the cost matrix of P^q is denoted by $C^{(q)}$.

In the following we show how the behaviour of the ACO algorithm for a (possibly inhomogeneous) restricted permutation problem P^q can be approximated using the behaviour of the ACO model for the elementary subproblem P. Consider an arbitrary of the q elementary subinstances of P^q — say the rth subinstance — and the quality of the solutions that m ants in an iteration have found on the other elementary subinstances (which form an instance of problem P^{q-1}). Without loss of generality assume that the quality of the solution found by ant i is at least as good as the solution found by ant $i+1$, $i \in [1:m-1]$. Let d_{max} be the maximum difference between two values in cost matrix C_r of the rth subproblem, i.e. $d_{max} := \max\{c_{ij}^{(r)} \mid i,j \in [1:n_0]\} - \min\{c_{ij}^{(r)} \mid i,j \in [1:n_0]\}$. Let d_i, $i \in [2:m]$ be the minimum of $d_{max}+1$ and the difference of the cost of the permutation found by ant i on P^{q-1} minus the cost of the permutation found by ant 1 on P^{q-1}. Our assumption implies $0 \leq d_2 \leq \ldots \leq d_m$. Define $\phi^{m;d_2,\ldots,d_m}$, $0 \leq d_i \leq d_{max}+1$, $i \in [2:m]$ as the probability that for m ants on problem P^{q-1} the difference of the costs of the solutions found by the ith best ant and the best ant is d_i when $d_i \leq d_{max}$ and when $d_i = d_{max}+1$ it is the probability that this difference is $> d_{max}$, $i \in [2:m]$. Let $\mathcal{D}$ be the set of all vectors $(d_2,\ldots,d_m)$ with integers $d_2 \leq \ldots \leq d_m$, $0 \leq d_i \leq d_{max}+1$, $i \in [2:m]$. Then for the rth elementary subproblem of P^q we obtain $\sigma_{ij}^{(m)}$ equals

$$\sum_{(d_2,\ldots,d_m) \in \mathcal{D}} \phi^{(m;d_2,\ldots,d_m)} \cdot \sigma_{ij}^{(m;d_2,\ldots,d_m)} = \sum_{x \in \mathcal{C}} w_x \cdot \omega_{ij}^{(x)}$$

with $w_x = \sum_{(d_2,\ldots,d_m)\in\mathcal{D}} \phi^{(m;d_2,\ldots,d_m)} \cdot \xi^{(m;x;d_2,\ldots,d_m)}$. This shows that the effect of the subproblem P^{q-1} on the remaining subinstance P_r of P^q is to change the weights between the influence of the different solution quality levels when compared to formula 4 for solving only the subproblem $P = P_r$.

We study the case of $m = 2$ ants in more detail. For problem P^q let $p_{d>y}$ and $p_{d=y}$ be the probabilities that the absolute value of the difference of the solution quality of two ants on the smaller problem P^{q-1} is $> y$ respectively $= y$. Then the probability $\sigma^{(2)}_{(l-1)n_0+h,(l-1)n_0+j}$ to select item j of the lth elementary subproblem equals

$$p_{d>d_{max}} \cdot \sigma'_{hj} + \sum_{y=1}^{d_{max}} p_{d=y} \cdot \sigma'^{(2;y)}_{hj} + p_{d=0} \cdot \sigma'^{(2)}_{hj}$$

where σ' refers to probabilities for the elementary subinstance P. This equation shows the interesting fact that part of the probability to select the item is just the probability σ'_{hj} of a single ant to select item j at place h for the elementary subproblem P. This is the case when the quality of the solutions of both ants differ by more than d_{max}. When the qualities of both solutions are the same the probability $\sigma'^{(2)}_{hj}$ to select item j at place h equals the probability that the better of two ants on problem P selects item j at place h. All other cases correspond to the situation that one of two ants on problem P has a malus.

The larger the number q of subproblems is the larger becomes the probability $p_{d>d_{max}}$. An important consequence is that the (positive) effect of competition between the two ants for finding good solutions becomes weaker and a possible bias in the decisions of a single ant has more influence.

Let $q_{d=y}$ ($q^{(q-1)}_{d=y}$) be the probability that the difference of the solution quality found by the first ant minus the solution quality found by the second ant on subproblem P (respectively P^{q-1}) is y (here we do not assume that the first ant finds the better solution). The value of this difference on subproblem P^{q-1} can be described as the result of a generalized one-dimensional random walk of length $q-1$. Define I_y as the set of tuples $(k_{-d_{max}}, k_{-d_{max}+1}, \ldots, k_{d_{max}-1}, k_{d_{max}})$ with $q-1 = \sum_{i=-d_{max}}^{d_{max}} k_i$, $y = \sum_{i=-d_{max}}^{d_{max}} k_i \cdot d_i$ where k_i is the number of elementary subinstances of P^{q-1} where the difference between the first and the second ant is $i \in [-d_{max} : d_{max}]$. Then $q^{(q-1)}_{d=y}$ can be computed as follows

$$\sum \frac{(q-1)!}{k_{-d_{max}}! \cdot \ldots \cdot k_{d_{max}}!} \cdot q_{d=-d_{max}}^{k_{-d_{max}}} \cdot \ldots \cdot q_{d=d_{max}}^{k_{d_{max}}}$$

where the sum is over $(k_{-d_{max}}, \ldots, k_{d_{max}}) \in I_y$. Clearly, $p_{d=0} = q^{(q-1)}_{d=0}$ and due to symmetry, for $y \neq 0$ $p_{d=y} = 2 \cdot q^{(q-1)}_{d=y}$. The remarks on analysing the ACO model for $m = 2$ ants can be extended to $m \geq 3$.

As an example consider the following problem P_1 with cost matrix

$$C_1 = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix} \tag{6}$$

The possible solution qualities for problem P_1 are 0, 2, and 4 and the optimal solution is to put item i on place i for $i \in [1:3]$. Hence $\sigma^{(2)}_{3i+j,3i+h} = p_{d>4} \cdot \sigma'_{hj} + \sum_{y=2,4} p_{d=y} \cdot \sigma'^{(2;y)}_{hj} + p_{d=0} \cdot \sigma'^{(2)}_{hj}$. Consider the following pheromone matrix for P_1

$$\begin{pmatrix} \tau_{11} & \tau_{12} & \tau_{13} \\ \tau_{21} & \tau_{22} & \tau_{23} \\ \tau_{31} & \tau_{32} & \tau_{33} \end{pmatrix} = \begin{pmatrix} 0.1 & 0.3 & 0.6 \\ 0.6 & 0.1 & 0.3 \\ 0.3 & 0.6 & 0.1 \end{pmatrix} \tag{7}$$

Then the probability for an ant to put, e.g., item 2 on place 2 can be computed as $\sigma_{22} = 0.1 \cdot 0.1/(0.1+0.3) + 0.6 \cdot 0.1/(0.1+0.6) \approx 0.111$. The matrix of selection probabilities for one ant on problem P_1 is

$$\begin{pmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{pmatrix} \approx \begin{pmatrix} 0.1 & 0.3 & 0.6 \\ 0.714 & 0.111 & 0.175 \\ 0.186 & 0.589 & 0.225 \end{pmatrix}$$

Since the optimal solution is to place item i on place i for $i \in [1:3]$ it is seems likely that the corresponding selection probabilities are larger with two ants per iteration compared to the case of a single ant in an iteration. But our example shows that this is not necessary. The probability to place item 2 on place 2 is $\sigma^{(2)}_{22} = 0.109$ and slightly smaller than $\sigma_{22} = 0.111$. When one of two ants has a malus the selection probabilities are mostly in between the case of two ants per iteration and a single ant per iteration. But again, the probability to place item 2 on place is a counterexample: $\sigma^{(2)}_{22} < \sigma_{22} < \sigma^{(2;2)}_{22}$. Although not true in every case, it can be observed that the selection probabilities for the better ant become more similar to the matrix of the selection probabilities for a single ant the higher the malus is.

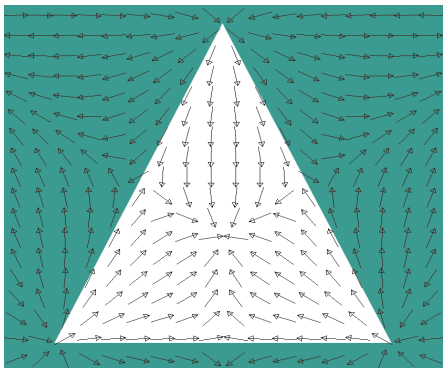


Figure 1: Direction of the vector field for changes of pheromone vectors $(\tau_{21}, \tau_{22}, \tau_{23})$ for the ACO model with a single ant and $\tau_{11} = \tau_{12} = \tau_{13} = 1/3$; possible pheromone vectors lay in the white area; distance from right (bottom,left) line: τ_{21} (τ_{22},τ_{23})

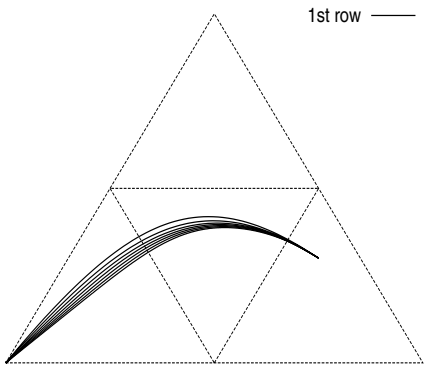


Figure 2: ACO model for P_1^q, $q = 2, 4, 8, 16, 32, 64, 128$, $m = 2$; change of pheromone values $\tau_{11}, \tau_{12}, \tau_{13}$ (first row of corresponding fixed point matrix is identical): starting point at $(\tau_{11}, \tau_{12}, \tau_{13}) = (0.1, 0.3, 0.6)$

6 FIXED POINTS

Since the pheromone values of an ACO algorithm reflect the frequencies of decisions that resulted in good solutions it is desirable that the selection probabilities used by an ant are equal to their corresponding pheromone values. As observed in [6] this might often not be the case since the decisions of an ant are not independent. We say that there is a selection bias when the probability of a random ant to choose an item is different from the corresponding pheromone value. A pheromone matrix where the probability of a random ant to choose an item is the same as the corresponding pheromone value, i.e., $\tau_{ij} = \sigma_{ij}$, for all i, j, is called fixed point matrix. A fixed point matrix can change in the ACO model only for $m \geq 2$ ants. The question arises which matrices are fixed point matrices for a permutation problem.

As an example consider a permutation problem of size $n = 3$ where the pheromone matrix $(\tau_{ij})_{i,j\in[1:3]}$ is defined by the values $\tau_{11}, \tau_{12}, \tau_{21}, \tau_{22}$. Clearly, the selection probabilities for the items in the first row are always equal to their corresponding pheromone values. It remains to determine the probability of choosing the first and the second item in the second row. The selection probabilities for these items are

$$\sigma_{21} := \frac{\tau_{12}\,\tau_{21}}{1-\tau_{22}} + \frac{(1-\tau_{11}-\tau_{12})\,\tau_{21}}{\tau_{21}+\tau_{22}}$$

$$\sigma_{22} := \frac{\tau_{11}\,\tau_{22}}{1-\tau_{21}} + \frac{(1-\tau_{11}-\tau_{12})\,\tau_{22}}{\tau_{21}+\tau_{22}}$$

The solutions of $\sigma_{21} - \tau_{21} = 0$ and $\sigma_{22} - \tau_{22} = 0$ show that the fixed points depend only on the pheromone values in the first row of the pheromone matrix:

1. $\tau_{21} = 0$, $\tau_{22} = (\tau_{11} + \tau_{12} - 1)/(\tau_{11} - 1)$
2. $\tau_{21} = (\tau_{11} + \tau_{12} - 1)/(\tau_{12} - 1)$, $\tau_{22} = 0$
3. $\tau_{21} = (\tau_{11})/(\tau_{11} + \tau_{12})$, $\tau_{22} = (\tau_{12})/(\tau_{11} + \tau_{12})$
4. $\tau_{21} = 1 - 2 \cdot \tau_{11}$, $\tau_{22} = 1 - 2 \cdot \tau_{12}$

Analysing the eigenvalues of the Jacobian matrix of $[\sigma_{21} - \tau_{21}, \sigma_{22} - \tau_{22}]$ the stability of the fixed points was determined. For every pair of possible values τ_{11} and τ_{12} exactly one of the fixed points is stable and attracting in the range of possible pheromone values. The cases (1), (2), and (3) are symmetric: for $\tau_{11} > 0.5$ the fixed point (1) is stable, for $\tau_{12} > 0.5$ the fixed point (2) is stable, and for $1 - \tau_{11} - \tau_{12} > 0.5$ the fixed point (3) is stable. In every other case the fixed point (4) is stable. Thus, there exists always exactly one stable fixed point matrix. Some of the three unstable fixed points might lay outside of the allowed parameter range $\tau_{21}, \tau_{22}, \tau_{23} \in (0, 1)$, $\tau_{21} + \tau_{22} + \tau_{23} = 1$.

The directions of the vector field for changes of pheromone vector $(\tau_{21}, \tau_{22}, \tau_{23})$ when $(\tau_{11}, \tau_{12}, \tau_{13}) = (1/3, 1/3, 1/3)$ are shown in Figure 1. In this case the vector field is symmetric with respect to rotations of 60 degree around the fixed point $(\tau_{21}, \tau_{22}, \tau_{23}) = (1/3, 1/3, 1/3)$. It is interesting to observe that in some areas of the vector field there are points $(\tau_{21}, \tau_{22}, \tau_{23})$ with a value $\tau_{2i} > 1/3$ for $i \in [1 : 3]$ that becomes even larger. This shows that the effects of the selection bias can be complex even for small problems.

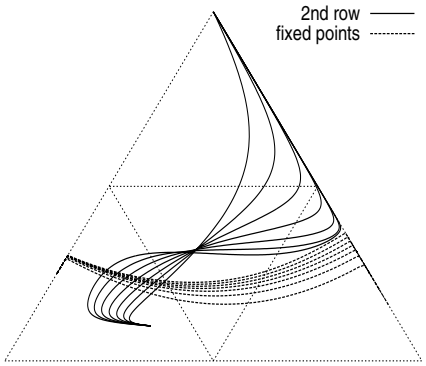


Figure 3: ACO model for P_1^q, $q = 2, 4, 8, 16, 32, 64, 128$, $m = 2$; change of pheromone values $\tau_{21}, \tau_{22}, \tau_{23}$ and fix point matrix: starting at $(\tau_{21}, \tau_{22}, \tau_{23}) = (0.6, 0.1, 0.3)$

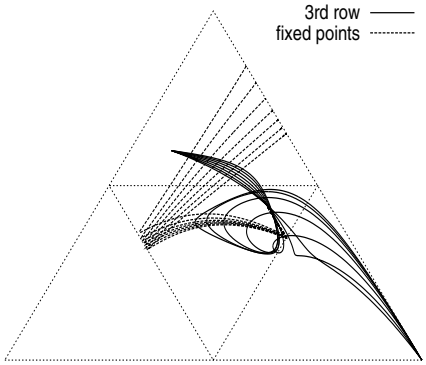


Figure 4: ACO model for P_1^q, $q = 2, 4, 8, 16, 32, 64, 128$, $m = 2$; change of values $\tau_{31}, \tau_{32}, \tau_{33}$ and fix point matrix: starting at $(\tau_{31}, \tau_{32}, \tau_{33}) = (0.3, 0.6, 0.1)$

7 DYNAMIC BEHAVIOUR

To study the dynamic behaviour of the ACO model consider problem P_1 with cost matrix given in (6) in Section 4. Figures 2-4 show the dynamic behaviour of the ACO model for P_1^q with different values of q and $m = 2$ ants. The pheromone evaporation ρ is 0.1 and the initial pheromone matrix is the same as in (7).

In contrast to the situation of one ant where the model converges to its stable selection fixed point the model converges for this example to the optimal solution. During convergence the actual position of the selection fixed point has a strong influence on the system. Note that for the first row of the matrix the pheromone values always equal the corresponding selection probabilities. Hence all dynamic in the first row is only due to competition (and not due to selection bias). Therefore, the pheromone values in Figure 2 approach the optimal values on an almost straight path. This is different for the pheromone vectors of row 2 and 3 (see figures 3, 4) where the stable selection fixed point has a large influence and the system moves often more in direction of the stable selection fixed point than in direction to the optimal solution. In Figure 4 paths with $q \geq 8$ contain a loop that is clearly influenced by the turn of the selection fixed point. The larger q the stronger is the deviation from a straight line because a high number of elementary subproblems leads to a small influence of competition (see Section 5).

In order to investigate the relative influence of selection, pure competition, and weak competition (where one ant has a malus) we computed the probabilities for the possible differences in solution quality between the two ants on the smaller problem P_1^{q-1}. Recall that

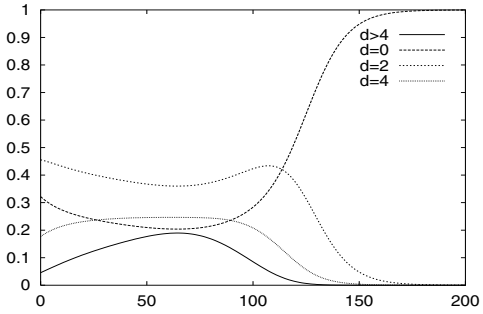


Figure 5: ACO model for P_1^q, $m = 2$, $q = 4$; change of $p_{d=y}$ on P_1^{q-1}: $p_{d=0}$, $p_{d=2}$, $p_{d=4}$, $p_{d>4}$

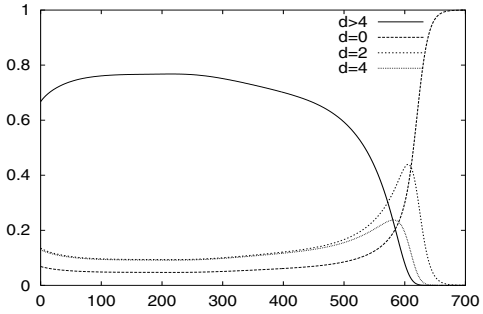


Figure 6: ACO model for P_1^q, $m = 2$, $q = 64$; change of $p_{d=y}$ on P_1^{q-1}: $p_{d=0}$, $p_{d=2}$, $p_{d=4}$, $p_{d>4}$

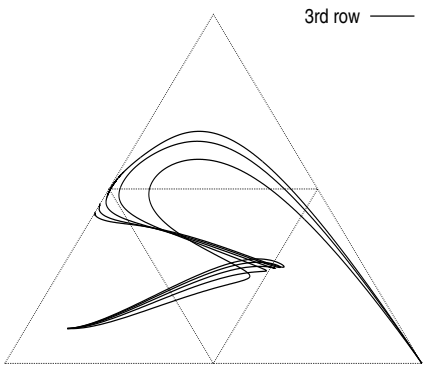


Figure 7: ACO model for problem P_2^q, $m = 2$, $q = 6, 10, 14, 18, 22$, initial matrix defined by ($\tau_{11} = 0.1$, $\tau_{12} = 0.2$, $\tau_{21} = 0.1$, $\tau_{22} = 0.7$; change of pheromone values $\tau_{31}, \tau_{32}, \tau_{33}$

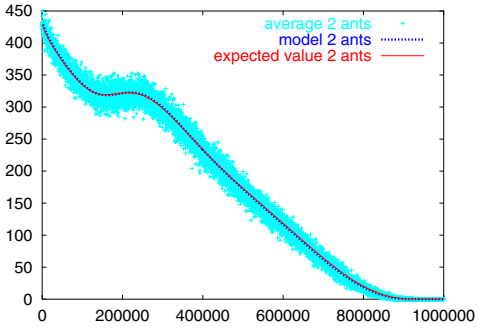


Figure 8: Solution quality on problem P_1^{128} with $m = 2$ over 1000000 (respectively 1000) iterations: average m ants: average quality found by m ants in the iteration of ACO algorithm; expected value m ants: expected quality found by a random ant on the pheromone matrix in the generation of ACO algorithm; model m ants: average quality for ACO model

the solution quality for the elementary subproblem P_1 can be 0, 2, or 4. Figures 5, 6 show the probabilities $p_{d=0}$, $p_{d=2}$, $p_{d=4}$, $p_{d>4}$ for a small and a large number of subproblems. For the large number the cases $p_{d>4}$, respectively $p_{d_1>4,d_2>4}$, which correspond to selection by a single ant on the elementary subproblem have a probability of more than 50% over most parts of the run. Only when the ACO model starts to converge the model is driven more by competition (as suggested by the analysis of the dynamics of the pheromone values).

To show that the selection bias can be so strong that the ACO model is not able to find the optimal solution consider the following small problem P_2 with cost matrix (c_{ij}) where $c_{ii} = 0$, $c_{13} = 100$ and all the other values are $c_{ij} = 1, i, j \in [1:3]$.

Figure 7 shows the behaviour of the ACO model for P_2^q with initial pheromone values $\tau_{11} = 0.1$, $\tau_{12} = 0.2$, $\tau_{21} = 0.1$, $\tau_{22} = 0.7$ for different values of q. For $q = 6, 10, 14$ subproblems the system converges to the optimal solution. But for larger numbers $q = 18, 22$ the influence of the selection bias is so high that the system converges to a non-optimal solution with $(\tau_{11}, \tau_{12}, \tau_{13}) = (x, 1-x, 1)$, $(\tau_{21}, \tau_{22}, \tau_{23}) = (0, 0, 1)$ and $(\tau_{31}, \tau_{32}, \tau_{33}) = (1-x, x, 0)$. Even for small numbers the system is driven by a selection bias but competition becomes stronger early enough to change the direction of convergence to an non-optimal solution. We tested the system also for all 666 matrices with a feasible combination of pheromone values $\tau_{ij} \in 0.1, 0, 2, \ldots, 0.9$, for $i, j \in [1:3]$. Even for the small problem P_2^2 the optimal solution can not be found for 83 of the 666 different initial matrices. This number increases up to 296 for P_2^{60}.

8 SIMULATION RESULTS

Since we have to consider single runs of the ACO algorithm in our simulation a very small value of $\rho = 0.0001$ was chosen for the algorithm. For the ACO model $\rho = 0.1$ was used. We compare then iteration t of the model with iteration $1000t$ of the algorithm. Note that this establishes an additional difference between the model and the algorithm.

Figure 8 shows the behavior of the algorithm and model on problem P_1^q with $q = 128$. The solution quality found by a random ant of the ACO model is nearly the same as the expected behaviour of an ant in the ACO algorithm (in the figure the corresponding curves are nearly identical). The observed average solution quality of the ACO algorithm found by $m = 2$ ants fluctuates around the solution quality that can be expected from the pheromone matrix in that generation. It is interesting that the expected solution quality of the ACO model and algorithm can become worse during the run (This effect is not the result of disadvantageous random decisions but is predicted by the model).

Figure 9 shows the results of the ACO algorithm on problems P_1^q with different values for q. Since every curve stems from one run of the algorithm only (it is not clear how to average in a reasonable way) the curves are not very smooth. Nevertheless the figure shows when compared to figures 3,4 that the ACO model predicts very well the development of the pheromone values of the ACO algorithm.

Of course not all aspects of ACO algorithms behaviour

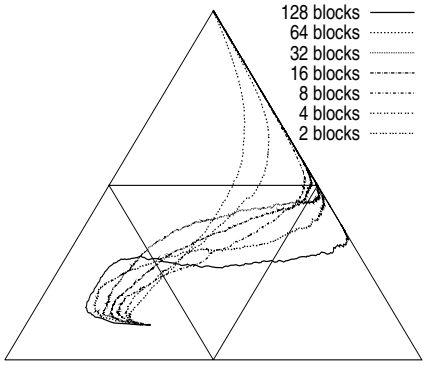


Figure 9: ACO algorithm for P_1^q, $q = 2, 4, 8, 16, 32, 64, 128$, $m = 2$: change of pheromone values τ_{21}, τ_{22}, τ_{23} starting at $(\tau_{21}, \tau_{22}, \tau_{23}) = (0.6, 0.1, 0.3)$

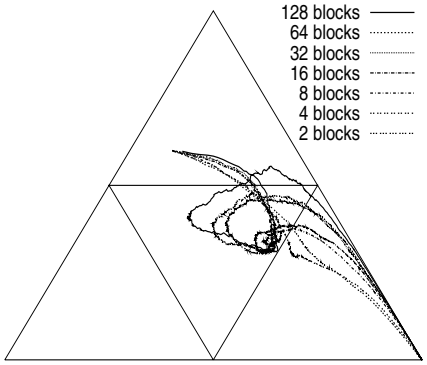


Figure 10: ACO algorithm for P_1^q, $q = 2, 4, 8, 16, 32, 64, 128$, $m = 2$: change of pheromone values τ_{31}, τ_{32}, τ_{33} starting at $(\tau_{31}, \tau_{32}, \tau_{33}) = (0.3, 0.6, 0.1)$

can be observed in the ACO model. As an example consider the restricted homogeneous permutation problem P^q where the ACO model shows the same behaviour on each of the elementary subproblems. The ACO algorithm in contrast behaves differently on the elementary subproblems due to random effects.

9 CONCLUSION

A deterministic model for ACO algorithms was proposed that uses a pheromone update mechanism based on the expected decisions of the ants. An interesting feature of the model is that it describes the behaviour of ACO algorithms through a combination of situations with different strength of competition between the ants. A fixed point analysis of the models pheromone matrices has given insights into the occurrence of biased decisions by the ants. It was shown analytically that the fixed points in the state space of the system have a strong influence on its optimization behaviour. Simulations have shown that the model accurately describes essential features of the dynamic behaviour of ACO algorithms.

References

[1] Dorigo, M. (1992). Optimization, Learning and Natural Algorithms *(in Italian)*. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy.

[2] Dorigo, M., Maniezzo, V., and Colorni, A. (1996). The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Trans. Systems, Man, and Cybernetics – Part B*, 26:29–41.

[3] Gutjahr, W. (2000). A graph-based Ant System and its convergence, *Future Generation Computer Systems*, 16:873–888.

[4] Gutjahr, W. (2002). ACO algorithms with guaranteed convergence to the optimal solution. *Information Processing Letters*, 82:145–153.

[5] Merkle, D., and Middendorf, M. (2000). An Ant Algorithm with a new Pheromone Evaluation Rule for Total Tardiness Problems. In Cagnoni, S., et al. (Eds.) *Real-World Applications of Evolutionary Computing*, LNCS 1803, 287–296, Springer.

[6] Merkle, D., and Middendorf, M. (2001). A New Approach to Solve Permutation Scheduling Problems with Ant Colony Optimization. In Boers, E. J. W., et al. (Eds.) *Applications of Evolutionary Computing*, LNCS 2037, 213–222, Springer.

[7] Merkle, D., and Middendorf, M. (2001). On the Behaviour of Ant Algorithms: Studies on Simple Problems. In *Proceedings of the 4th Metaheuristics International Conference (MIC'2001)*, Porto, 573-577.

[8] Mitchell, M., and Forrest, S. (1997). Royal Road Functions. In Bäck, T., Fogel, D.B. and Michalewicz, Z. (Eds.), *Handbook of Evolutionary Computation*. B2.7:20–25, Oxford University Press, Oxford.

[9] van Nimwegen, E., Crutchfield, J.P., Mitchell, M. (1999). Statistical dynamics of the royal road genetic algorithm. *Theor. Comp. Sci.*, 229:41–102.

[10] Prügel-Bennett, A., and Rogers, A. (2001). Modelling GA Dynamics. In Kallel et al. (Eds.), *Theoretical Aspects of Evolutionary Computing*, Springer, 59-86.

[11] Prügel-Bennett, A, and J.L. Shapiro (1997). An Analysis of Genetic Algorithms Using Statistical Mechanics. *Physica D*, 104:75-114.

[12] Stützle, T. and Dorigo, M. (2000). A short convergence proof for a class of ACO algorithms. TR 2000-35, IRIDIA, Universitè Libre de Bruxelles, Belgium.

[13] Vose, M.D., *The Simple Genetic Algorithm: Foundations and Theory* (1999). MIT Press, Cambridge, MA.

[14] Vose, M.D., Liepins, G.E. (1991). Punctuated equilibria in genetic search. Complex Systems, 5:31–44.

Continual Coevolution through Complexification

Kenneth O. Stanley
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712
kstanley@cs.utexas.edu

Risto Miikkulainen
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712
risto@cs.utexas.edu

Abstract

In competitive coevolution, the goal is to establish an "arms race" that will lead to increasingly sophisticated strategies. However, in practice, the process often leads to idiosyncrasies rather than continual improvement. Applying the NEAT method for evolving neural networks to a competitive simulated robot duel domain, we will demonstrate that (1) as evolution progresses the networks become more complex, (2) complexification elaborates on existing strategies, and (3) if NEAT is allowed to complexify, it finds dramatically more sophisticated strategies than when it is limited to fixed-topology networks. The results suggest that in order to realize the full potential of competitive coevolution, genomes must be allowed to complexify as well as optimize over the course of evolution.

1 INTRODUCTION

In competitive coevolution, two or more populations of individuals evolve simultaneously in an environment where an increased fitness in one population leads to a decreased fitness for another. Ideally, competing populations will continually outdo one another, leading to an "arms race" of increasing sophistication (Dawkins and Krebs 1979; Van Valin 1973). In practice, evolution tends to find the simplest solutions that can win, meaning that strategies can switch back and forth between different idiosyncratic yet uninteresting variations (Darwen 1996; Floreano and Nolfi 1997; Rosin and Belew 1997). Several methods have been developed to encourage the arms race (Angeline and Pollack 1994; Rosin and Belew 1997). For example, a "hall of fame" can be used to ensure that current strategies remain competitive against strategies from the past. Although such techniques improve the performance of competitive coevolution, they do not directly encourage continual coevolution, i.e. creating new solutions that maintain existing capabilities. Much time is wasted evaluating solutions that are deficient in this way.

The problem is that in general genomes have a fixed set of genes mapping to a fixed phenotypic structure. Once a good strategy is found, the entire representational space of the genome is used to encode it. Thus, the only way to improve it is to *alter* the strategy, thereby sacrificing some of the functionality learned over previous generations.

In this paper, we propose a novel solution to this problem. The idea is to *complexify* or add structure to the dominant strategy, so that it does not merely become different, but rather *more elaborate*. This idea is implemented in a method for evolving increasingly complex neural networks, called NeuroEvolution of Augmenting Topologies (NEAT; Stanley and Miikkulainen 2001, 2002b,c). NEAT begins by evolving networks without any hidden nodes. Over many generations, new hidden nodes and connections are added, resulting in the complexification of the solution space. This way, more complex strategies elaborate on simpler strategies, focusing search on solutions that are likely to maintain existing capabilities.

NEAT was tested in a competitive robot control domain with and without complexification. The main results were that (1) evolution did complexify when possible, (2) complexification led to elaboration, and (3) significantly more sophisticated and successful strategies were evolved with complexification than without. These results imply that complexification allows coevolution to continually elaborate on successful strategies, resulting in an arms race that achieves a significantly higher level of sophistication than is otherwise possible.

We begin by describing the NEAT neuroevolution method, followed by a description of the robot duel domain and a discussion of the results.

2 NEUROEVOLUTION OF AUGMENTING TOPOLOGIES (NEAT)

The NEAT method of evolving artificial neural networks combines the usual search for appropriate network weights with complexification of the network structure. This approach is highly effective: NEAT outperforms other neuroevolution (NE) methods, e.g. on the benchmark double pole balancing task by a factor of five (Stanley and Miikkulainen 2001, 2002b,c). The NEAT method consists of solutions to three fundamental challenges in evolving neural network topology: (1) What kind of genetic representation would allow disparate topologies to crossover in a meaningful way? (2) How can topological innovation that needs a few generations to optimize be protected so that it does not disappear from the population prematurely? (3) How can topologies be minimized *throughout evolution* so the most efficient solutions will be discovered? In this section, we explain how NEAT addresses each challenge.[1]

2.1 GENETIC ENCODING

Evolving structure requires a flexible genetic encoding. In order to allow structures to complexify, their representations must be dynamic and expandable. Each genome in NEAT includes a list of *connection genes*, each of which refers to two *node genes* being connected. Each connection gene specifies the in-node, the out-node, the weight of the connection, whether or not the connection gene is expressed (an enable bit), and an *innovation number*, which allows finding corresponding genes during crossover.

Mutation in NEAT can change both connection weights and network structures. Connection weights mutate as in any NE system, with each connection either perturbed or not. Structural mutations, which form the basis of complexification, occur in two ways (figure 1). In the *add connection* mutation, a single new connection gene is added connecting two previously unconnected nodes. In the *add node* mutation an existing connection is split and the new node placed where the old connection used to be. The old connection is disabled and two new connections are added to the genome. This method of adding nodes was chosen in order to integrate new nodes immediately into the network. Through mutation, genomes of varying sizes are created, sometimes with completely different connections specified at the same positions.

In order to perform crossover, the system must be able to tell which genes match up between *any* individuals in the population. The key observation is that two genes that have the same historical origin represent the same structure (although possibly with different weights), since they were both derived from the same ancestral gene from some point in the past. Thus, all a system needs to do to know which genes line up with which is to keep track of the historical origin of every gene in the system.

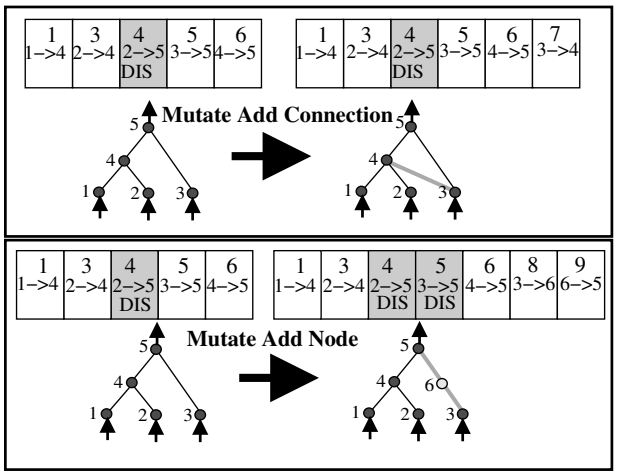


Figure 1: **The two types of structural mutation in NEAT.** Both types, adding a connection and adding a node, are illustrated with the genes above their phenotypes. The top number in each genome is the *innovation number* of that gene. The bottom two numbers denote the two nodes connected by that gene. The weight of the connection, also encoded in the gene, is not shown. The symbol *DIS* means that the gene is disabled, and therefore not expressed in the network. The figure shows how connection genes are appended to the genome when a new connection is added to the network and when a new node is added. Assuming the depicted mutations occurred one after the other, the genes would be assigned increasing innovation numbers as the figure illustrates, thereby allowing NEAT to keep an implicit history of the origin of every gene in the population.

Tracking the historical origins requires very little computation. Whenever a new gene appears (through structural mutation), a *global innovation number* is incremented and assigned to that gene. The innovation numbers thus represent a chronology of every gene in the system. As an example, let us say the two mutations in figure 1 occurred one after another in the system. The new connection gene created in the first mutation is assigned the number 7, and the two new connection genes added during the new node mutation are assigned the numbers 8 and 9. In the future, whenever these genomes crossover, the offspring will inherit the same innovation numbers on each gene; innovation numbers are never changed. Thus, the historical origin of every gene in the system is known throughout evolution.

Through innovation numbers, the system now knows exactly which genes match up with which. Genes that do not match are either *disjoint* or *excess*, depending on whether they occur within or outside the range of the other parent's innovation numbers. When crossing over, the genes in both

[1] A more comprehensive description of the NEAT method is given in Stanley and Miikkulainen (2001, 2002c).

genomes with the same innovation numbers are lined up. Genes that do not match are inherited from the more fit parent, or if they are equally fit, from both parents randomly.

Historical markings allow NEAT to perform crossover without the need for expensive topological analysis. Genomes of different organizations and sizes stay compatible throughout evolution, and the problem of competing conventions (Radcliffe 1993) is essentially avoided. Such compatibility is essential in order to complexify structure.

2.2 PROTECTING INNOVATION THROUGH SPECIATION

Adding new structure to a network usually initially reduces fitness. However, NEAT speciates the population, so that individuals compete primarily within their own niches instead of with the population at large. This way, topological innovations are protected and have time to optimize their structure before they have to compete with other niches in the population.

Historical markings make it possible for the system to divide the population into species based on topological similarity. We can measure the distance δ between two network encodings as a simple linear combination of the number of excess (E) and disjoint (D) genes, as well as the average weight differences of matching genes ($\overline{W}$):

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \overline{W}. \quad (1)$$

The coefficients c_1, c_2, and c_3 adjust the importance of the three factors, and the factor N, the number of genes in the larger genome, normalizes for genome size. Genomes are tested one at a time; if a genome's distance to a randomly chosen member of the species is less than δ_t, a compatibility threshold, it is placed into this species. Each genome is placed into the first species where this condition is satisfied, so that no genome is in more than one species.

As the reproduction mechanism for NEAT, we use *explicit fitness sharing* (Goldberg and Richardson 1987), where organisms in the same species must share the fitness of their niche, preventing any one species from taking over the population.

2.3 MINIMIZING DIMENSIONALITY THROUGH COMPLEXIFICATION

Unlike other systems that evolve network topologies and weights (Angeline et al. 1993; Gruau et al. 1996; Yao 1999; Zhang and Muhlenbein 1993), NEAT begins with a uniform population of simple networks with no hidden nodes. Speciation protects new innovations, allowing topological diversity to be gradually introduced over evolution.

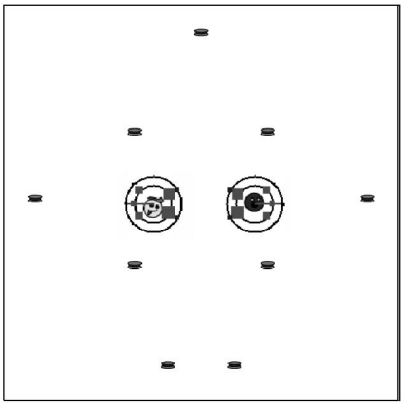


Figure 2: **The Robot Duel Domain.** The robots begin on opposite sides of the board facing away from each other as shown by the lines pointing away from their centers. The concentric circles around each robot represent the separate rings of opponent sensors and food sensors available to each robot. Each ring contains five sensors, which appear larger or smaller depending on their activations. From this initial position, neither robot has a positional advantage. The robots lose energy when they move around, yet they can gain energy by consuming food (shown as black dots). The food is placed in a horizontally symmetrical pattern around the middle of the board. The objective is to attain a higher level of energy than the opponent, and then collide with it. Because of the complex interaction between foraging, pursuit, and evasion behaviors, the domain allows for a broad range of strategies of varying sophistication. Animated demos of the robot duel domain are available at **www.cs.utexas.edu/users/nn/pages/research/neatdemo.html**.

New structure is introduced incrementally as structural mutations occur, and only those structures survive that are found to be useful through fitness evaluations. This way, NEAT searches through a minimal number of weight dimensions, significantly reducing the number of generations necessary to find a solution, and ensuring that networks become no more complex than necessary. In other words, NEAT searches for the optimal topology by *complexifying* when necessary.

3 THE ROBOT DUEL DOMAIN

To demonstrate the effect of complexification on competitive coevolution, a domain is needed where it is possible to develop increasingly sophisticated strategies and where the sophistication can be readily measured. A balance between the potential complexity of evolved strategies and their analyzability is difficult to strike. Pursuit and evasion tasks have been utilized for this purpose in the past (Gomez and Miikkulainen 1997; Jim and Giles 2000; Miller and Cliff 1994; Reggia et al. 2001), and can serve as a benchmark domain for competitive coevolution as well. While past experiments evolved either a predator or a prey, an interesting coevolution task can be established if the agents are instead equal and engaged in a duel. To win, an agent must de-

velop a strategy that outwits that of its opponent, utilizing structure in the environment.

In the robot duel domain, two simulated robots try to overpower each other (figure 2). The two robots begin on opposite sides of a rectangular room facing away from each other. As the robots move, they lose energy in proportion to the amount of force they apply to their wheels. Although the robots never run out of energy (they are given enough to survive the entire competition), the robot with higher energy can win by colliding with its competitor. In addition, each robot has a sensor indicating the difference in energy between itself and the other robot. To keep their energies high, the robots can consume food items, arranged in a symmetrical pattern in the room.

The robot duel task supports a broad range of sophisticated strategies that are easy to observe and interpret without expert knowledge. The competitors must become proficient at foraging, prey capture, and escaping predators. In addition, they must be able to quickly switch from one behavior to another. The task is well-suited to competitive coevolution because naive strategies such as forage-then-attack can be complexified into more sophisticated strategies such as luring the opponent to waste its energy before attacking.

The simulated robots are similar to Kheperas (Mondada et al. 1993). Each has two wheels controlled by separate motors. Five rangefinder sensors can sense food and another five can sense the other robot. Finally, each robot has an energy-difference sensor, and a single wall sensor.

The robots are controlled with neural networks evolved with NEAT. The networks receive all of the robot sensors as inputs, as well as a constant bias that NEAT can use to change the activation thresholds of neurons. They produce three motor outputs: Two to encode rotation either right or left, and a third to indicate forward motion power.

This complex robot-control domain allows competitive coevolution to evolve increasingly sophisticated and complex strategies, and can be used to benchmark coevolution methods.

4 EXPERIMENTS

In order to demonstrate how complexification contributes to continual coevolution, we ran four evolution trials with full NEAT and three trials with complexification turned off. The methodology is described below.

4.1 COMPETITIVE COEVOLUTION SETUP

In each evolution trial, 2 populations, each containing 256 genomes, were evolved simultaneously. In each generation, each population is evaluated against an intelligently chosen sample of networks from the other population. The population currently being evaluated is called the *host* population, and the population from which opponents are chosen is called the *parasite* population (Rosin and Belew 1997). The parasites are chosen for their quality and diversity, making host/parasite evolution more efficient and more reliable than random or round robin tournament.

A single fitness evaluation included two competitions, one for the east and one for the west starting position. That way, networks needed to implement general strategies for winning, independent of their starting positions. Host networks received a single fitness point for each win, and no points for losing. If a competition lasted 750 time steps with no winner, the host received 0 points.

In selecting the parasites for fitness evaluation, good use can be made of the speciation and fitness sharing that already occur in NEAT. Each host was evaluated against the champions of four species with the highest fitness. They are good opponents because they are the best of the best species, and they are guaranteed to be diverse because their compatibility must be outside the threshold δ_t (section 2.2). Another eight opponents were chosen randomly from a Hall of Fame (Rosin and Belew 1997) that contained population champions from all generations. Together, speciation, fitness sharing, and Hall of Fame comprise a state of the art competitive coevolution methodology. However, as our experimental results will show, complexification is the most important ingredient in establishing continual coevolution.

4.2 MONITORING PROGRESS IN COMPETITIVE COEVOLUTION

In order to track progress in coevolution, we need to be able to tell whether one strategy is better than another. Because the board configurations can vary during the game, networks were compared on 144 different food configurations from each side of the board, giving 288 total comparisons. The food configurations included the same 9 symmetrical food positions used during training, plus an additional 2 food items, which were placed in one of 12 different positions on the east and west halves of the board. Some starting food positions give an initial advantage to one robot or another, depending on how close they are to the robots' starting positions. We say that network a *is superior* to network b if a wins more comparisons than b out of the 288 total comparisons.

Given this definition of superiority, progress can be tracked. The obvious way to do it is to compare each network to others throughout evolution, finding out whether later strategies can beat more opponents than earlier strategies. For example, Floreano and Nolfi (1997) used a measure called

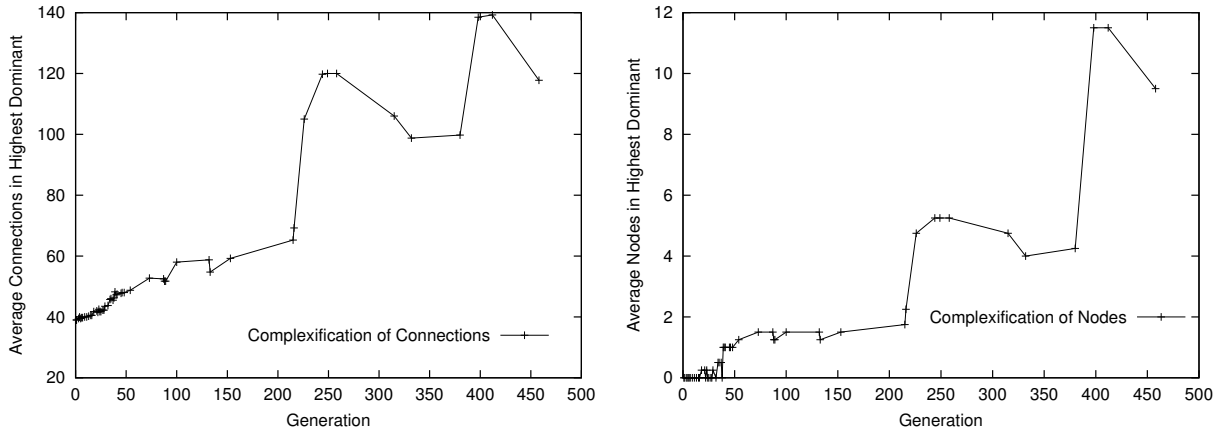


Figure 3: **Complexification of connections and nodes over generations.** The graphs depict the average number of connections and the average number of hidden nodes in the highest dominant network in each generation. Averages are taken over four complexifying runs. A hash mark appears every generation in which a new dominant strategy emerged in at least one of the four runs. The graphs show that as dominance increases, so does complexity level on average. The differences in complexity between the average final dominant and first dominant strategies are statistically significant for both connections and nodes ($p < 0.05$).

master tournament, in which the champion of each generation is compared to all other generation champions. Unfortunately, such methods are impractical in a time-intensive domain such as the robot duel competition. Moreover, the master tournament only shows how often each champion wins against other champions. In order to track strategic innovation, we need to identify *dominant strategies*, i.e. those that defeat *all previous* dominant strategies. This way, we can make sure that evolution proceeds by developing a progression of strictly more powerful strategies, instead of e.g. switching between alternative ones.

To meet this goal, we developed the *dominance tournament* method of tracking progress in competitive coevolution (Stanley and Miikkulainen 2002a). Let a *generation champion* be the winner of a 288 game comparison between the two population champions of a single generation. Let d_j be the jth dominant strategy to appear in the evolution. Then dominance is defined recursively:

- The first dominant strategy d_1 is the generation champion of the first generation;
- dominant strategy d_j, where $j > 1$, is a generation champion such that for all $i < j$, d_j is superior to (wins the 288 game comparison with) d_i.

This strict definition of dominance prohibits circularities. For example, d_4 must be superior to strategies d_1 through d_3, d_3 superior to both d_1 and d_2, and d_2 superior to d_1. The entire process of deriving a dominance hierarchy from a population is a *dominance tournament*, where competitors play all previous dominant strategies until they either lose a 288 game comparison, or win every comparison to previous dominant strategies, thereby becoming a new dominant strategy. Dominance tournaments require significantly fewer comparisons than the master tournament.

The question tested in the experiments is: Does the complexification of networks help attain high levels of dominance?

5 RESULTS

Each of the seven evolution trials lasted 500 generations, and took between 5 and 10 days on a 1GHz PIII processor, depending on the progress of evolution and sizes of the networks involved. The NEAT algorithm itself used less than 1% of this computation: the rest of the time was spent in evaluating networks in the robot duel task. Evolution of fully-connected topologies took about 90% longer than structure-growing NEAT because larger networks take longer to evaluate.

We define *complexity* as the number of nodes and connections in a network: The more nodes and connections there are in the network, the more complex behavior it can potentially implement. The results were analyzed to answer three questions: (1) As evolution progresses does it also continually complexify? (2) How is complexification utilized to create more sophisticated strategies? (3) Does complexification allow better strategies to be discovered than does evolving fixed-topology networks?

5.1 EVOLUTION OF COMPLEXITY

NEAT was run four times, each time from a different seed, to verify consistency of results. The highest levels of dominance achieved were 17, 14, 17, and 16, averaging at 16.

At each generation where the dominance level increased in at least one of the four runs, we averaged the number of connections and number of nodes in the current dominant strategy across all runs (figure 3). Thus, the graphs represent a total of 64 dominance transitions spread over 500 generations. The rise in complexity is dramatic, with the average number of connections tripling and the average number of hidden nodes rising from 0 to almost 10. In a smooth trend over the first 200 generations, the number of connections in the dominant strategy nearly doubles. During this early period, dominance transitions occur frequently (fewer prior strategies need to be beaten to achieve dominance). Over the next 300 generations, dominance transitions become more sparse, although they continue to occur.

Between the 200th and 500th generations a staircase pattern emerges, where complexity first rises dramatically, then settles, then abruptly increases again. The reason for this pattern is speciation. While one species is adding a large amount of structure, other species are optimizing the weights of less complex networks. While it is initially faster to grow structure until something works, such ad hoc constructions are eventually supplanted by older species that have been steadily optimizing for a long period of time. Thus, spikes in complexity occur when structural elaboration leads to a better strategy, and complexity slowly settles when older structures optimize their weights and overtake more recent structural innovations.

The results show more than just that the champions of each generation tend to become complex. The dominant strategies, i.e. the networks that have a strictly superior strategy to every previous dominant strategy, tend to be more complex the higher the dominance level. Thus, the results verify that continual progress in evolution is paired with increase in complexity.

5.2 SOPHISTICATION THROUGH COMPLEXIFICATION

To see how complexification contributes to evolution, let us observe the development of a sample dominant strategy, i.e. the evolution of the species that produced the winning network d_{17}, in the third run. Let us use S_k for the best network in S at generation k, and h_l for the lth hidden node to arise from a structural mutation over the course of evolution. We will track both strategic and structural innovations in order to see how they correlate. Let us begin with S_{100} (figure 4, left), when S had a mature zero-hidden-node strategy:

- S_{100}'s main strategy was to follow the opponent, putting it in a position where it might by chance collide with its opponent when its energy is up. However, S_{100} followed the opponent even when the opponent had more energy, leaving S_{100} vulnerable to attack. S_{100} did not clearly switch roles between foraging and chasing the enemy, causing it to miss opportunities to gather food.

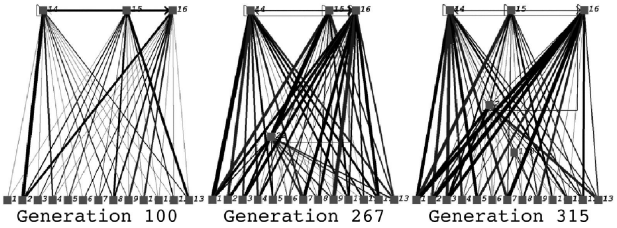


Figure 4: **Complexification of a Winning Species.** The best networks in the same species are depicted at landmark generations. Over generations, the networks in the species complexified and gained skills.

- S_{200}. During the next 100 generations, S evolved a *resting* strategy, which it used when it had significantly lower energy than the enemy. In such a situation, the robot stopped moving, while the other robot wasted energy running around. By the time the opponent gets close, its energy was often low enough to be attacked. The resting strategy is an example of improvement that can take place without complexification: it involved increasing the inhibition from the enemy difference sensor, thereby slightly modifying intensity of an existing behavior only.

- In S_{267} (figure 4, middle), a new hidden node, h_{22}, appeared. Node h_{22} arrived through an interspecies mating, and had been optimized for several generations already, Node h_{22} gave the robot the ability to change its behavior at once into a consistent all-out attack. Because of this new skill, S_{267} no longer needed to follow the enemy closely at all times, leaving it to focus on collecting food. By implementing this new strategy through a new node, it was possible not to interfere with the already existing resting strategy, so that S now switched roles between resting when in danger to attacking when high on energy. This way, the new structure resulted in strategic elaboration.

- In S_{315} (figure 4, right), h_{172} split a link between an input sensor and h_{22}. Replacing a direct connection with a sigmoid function greatly improved S_{315}'s ability to attack at appropriate times, leading to very accurate role switching between attacking and foraging. S_{315} would try to follow the opponent from afar focusing on resting and foraging, and only zoom in for attack when victory was certain. This final structural addition shows how new structure can greatly improve the accuracy and timing of existing behaviors.

The analysis above shows that in some cases, weight optimization alone can produce improved strategies. However, when those strategies need to be extended, adding new structure allows the new behaviors to coexist with old strategies. Also, in some cases it is necessary to add complexity to make the timing or execution of the behavior more accurate. These results show how complexification can be utilized to produce sophistication in competitive coevolution.

5.3 COMPLEXIFICATION VS. FIXED-STRUCTURE EVOLUTION

To see whether complexifying coevolution is more powerful than standard non-complexifying coevolution, we ran three trials with fixed, fully-connected topologies. To make the comparison fair, the fixed-topology networks in the first two trials had 10 hidden nodes, as did the winning networks of complexifying runs on average. In the third trial, fixed-topology networks had only five hidden nodes, which gives them the same number of connections as the complexifying trials. In the first trial, the hidden nodes were fully connected to the outputs. In the other two trials, the *inputs* were also fully connected to outputs. In all standard runs, the hidden layer was fully recurrent, because complexifying runs were found to evolve recurrent connections. Although topologies were fixed, evolution continued to speciate using weight differences.

Fixed-topology Run	Highest Dom.	Equivalent Dom. Level (out of 16)	Equivalent Generation (out of 500)
1: 10 Hidden Node	12	5.5	17.75
2: 10 Hidden Node, Direct Connections	14	9.25	39
3: 5 Hidden Nodes, Direct Connections	10	10.75	65.5

Table 1: **Comparing the dominant strategies in the fixed-topology (i.e. standard) coevolution with those of complexifying coevolution.** The second column shows how many levels of dominance were achieved in the standard coevolution. The third column gives the highest dominance level in complexifying runs that the dominant from the standard run can defeat and the fourth column shows its average generation. The main result is that the level of sophistication reached by standard coevolution is significantly lower than that reached by complexifying coevolution.

In Table 1, the relative sophistication of the strategies developed are compared to those in complexifying coevolution. We compared the highest dominant network from each of the standard runs with the entire dominance hierarchies of all the complexifying runs. The table reports the highest dominance level *within the complexifying runs* that the best fixed-topology network can defeat on average. In all cases, the standard strategy reaches only the middle levels of the hierarchy, i.e. 5.5, 9.25, and 10.75 out of possible 16. Complexifying coevolution on average found 7 levels of dominance *above* the most sophisticated strategies of standard coevolution. Considering that high levels of dominance are much more difficult to attain than low levels, it is clear that complexifying coevolution develops a dramatically higher level of sophistication.

Another significant result is that NEAT developed equivalent strategies very early in evolution. For example, the second standard run stopped producing new dominant strategies after the 169th generation, followed by 331 consecutive generations without any additional dominant strategies. This network can defeat about the 9th dominant from complexifying coevolution, which was found on average in the 39th generation. In other words, standard coevolution is considerably slower in finding even the first few steps in the dominance hierarchy.

In summary, complexifying coevolution progresses faster and discovers significantly more sophisticated solutions than standard coevolution.

6 DISCUSSION AND FUTURE WORK

Evolution in nature acts as both an optimizer *and* a complexifier. Not only do existing genes express different alleles, but *new* genes are added occasionally through a process called gene amplification (Darnell and Doolittle 1986). Therefore, we should expect to find that complexification can also play a role in models of open-ended evolution, such as competitive coevolution, thus strenghening the analogy of evolutionary computation with nature.

Indeed, as the results confirm, complexification does enhance the capability of competitive coevolution to find sophisticated strategies. Complexification encourages continual *elaboration*, whereas evolution of fixed-structures proceeds primarily by *alteration*. When a fixed genome is used to represent a strategy, that strategy can be optimized, but it is not possible to complexify without sacrificing some of the knowledge that is already present. In contrast, if new genetic material can be added, then sophisticated elaborations can be layered above existing structure.

Complexification can find solutions that are difficult to find by evolving fixed structure. In fixed evolution, the complexity must be guessed just right: too little structure will make it impossible to solve the problem and too much will make the search space too large to search efficiently. A complexifying system saves the user from such concerns.

Complexification is a new and still largely unexplored research area. How complexifying systems work in general, and what the best ways are to describe such systems are open questions at this point. Although evolution is the best known complexifier, that does not mean it is the only one.

Organizations (such as corporations and governments) are also complexifying systems, with new positions being created that only have meaning relative to positions that previously existed. We need to develop an abstract description of complexification, from which we can derive theories and rules for understanding and utilizing complexification in different domains.

7 CONCLUSION

We hypothesized that complexification of genomes can lead to continual coevolution of increasingly sophisticated strategies. Experimental results showed three trends: (1) as evolution progresses, complexity of solutions increases, (2) evolution uses complexification to elaborate on existing strategies, and (3) complexifying coevolution is significantly more successful in finding highly sophisticated strategies than evolution of fixed structures. These results suggest that complexification is a crucial component of continual coevolution.

Acknowledgments

This research was supported in part by the National Science Foundation under grant IIS-0083776 and by the Texas Higher Education Coordinating Board under grant ARP-003658-476-2001.

References

Angeline, P. J., and Pollack, J. B. (1994). Competitive environments evolve better solutions for complex tasks. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, 264–270.

Angeline, P. J., Saunders, G. M., and Pollack, J. B. (1993). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5:54–65.

Darnell, J. E., and Doolittle, W. F. (1986). Speculations on the early course of evolution. *Proceedings of the National Academy of Sciences, USA*, 83:1271–1275.

Darwen, P. J. (1996). *Co-Evolutionary Learning by Automatic Modularisation with Speciation*. PhD thesis, School of Computer Science, University College, University of New South Wales.

Dawkins, R., and Krebs, J. R. (1979). Arms races between and within species. *Proceedings of the Royal Society of London Series B*, 205:489–511.

Floreano, D., and Nolfi, S. (1997). God save the red queen! Competition in co-evolutionary robotics. *Evolutionary Computation*, 5.

Goldberg, D. E., and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In Grefenstette, J. J., editor, *Proceedings of the Second International Conference on Genetic Algorithms*, 148–154. San Francisco, CA: Morgan Kaufmann.

Gomez, F., and Miikkulainen, R. (1997). Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342.

Gruau, F., Whitley, D., and Pyeatt, L. (1996). A comparison between cellular encoding and direct encoding for genetic neural networks. In Koza, J. R., Goldberg, D. E., Fogel, D. B., and Riolo, R. L., editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, 81–89. Cambridge, MA: MIT Press.

Jim, K.-C., and Giles, C. L. (2000). Talking helps: Evolving communicating agents for the predator-prey pursuit problem. *Artificial Life*, 6(3):237–254.

Miller, G., and Cliff, D. (1994). Co-evolution of pursuit and evasion i: Biological and game-theoretic foundations. Technical Report CSRP311, School of Cognitive and Computing Sciences, University of Sussex, Brighton, UK.

Mondada, F., Franzi, E., and Ienne, P. (1993). Mobile robot miniaturization: A tool for investigation in control algorithms. In *Proceedings of the Third International Symposium on Experimental Robotics*, 501–513.

Radcliffe, N. J. (1993). Genetic set recombination and its application to neural network topology optimization. *Neural computing and applications*, 1(1):67–90.

Reggia, J. A., Schulz, R., Wilkinson, G. S., and Uriagereka, J. (2001). Conditions enabling the evolution of inter-agent signaling in an artificial world. *Artificial Life*, 7:3–32.

Rosin, C. D., and Belew, R. K. (1997). New methods for competitive evolution. *Evolutionary Computation*, 5.

Stanley, K. O., and Miikkulainen, R. (2001). Evolving neural networks through augmenting topologies. Technical Report AI2001-290, Department of Computer Sciences, The University of Texas at Austin.

Stanley, K. O., and Miikkulainen, R. (2002a). The dominance tournament method of monitoring progress in coevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002) Workshop Program*. San Francisco, CA: Morgan Kaufmann.

Stanley, K. O., and Miikkulainen, R. (2002b). Efficient evolution of neural network topologies. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02)*. IEEE.

Stanley, K. O., and Miikkulainen, R. (2002c). Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*. San Francisco, CA: Morgan Kaufmann.

Van Valin, L. (1973). A new evolutionary law. *Evolution Theory*, 1:1–30.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447.

Zhang, B.-T., and Muhlenbein, H. (1993). Evolving optimal neural networks using genetic algorithms with Occam's razor. *Complex Systems*, 7:199–220.

Cross-Validation in Multiagent-based Simulation: Analyzing Evolutionary Bargaining Agents

Keiki Takadama
ATR Human Information Science Labs.
2-2-2 Hikaridai
Seika-cho, Soraku-gun
Kyoto 619–0288 Japan
keiki@atr.co.jp

Yutaka L. Suematsu
National University of Engineering
Av. Tupac Amaru 210 - Rimac
Apartado 1301 - Lima 100
Lima, PERU
suematsu@terra.com.pe

Norberto E. Nawa
ATR Human Information Science Labs.
2-2-2 Hikaridai
Seika-cho, Soraku-gun
Kyoto 619–0288 Japan
eiji@atr.co.jp

Katsunori Shimohara
ATR Human Information Science Labs.
2-2-2 Hikaridai
Seika-cho, Soraku-gun
Kyoto 619–0288 Japan
katsu@atr.co.jp

Abstract

This paper addresses *cross-validation* in multiagent-based simulation by analyzing evolutionary agents in a bargaining game in game theory. In particular, this paper focuses on analyzing different *learning mechanisms* and *knowledge representation* capabilities applied to agents for cross-validation. To investigate these issues, we compare the following two cases: (1) agents employing an evolutionary strategy (ES) and agents employing a learning classifier system (LCS) as different learning mechanisms; and (2) agents handling an ordinary explanation of numbers and agents handling a limited explanation of numbers as different knowledge representation capabilities. An intensive comparison of simulation results reveal the following implications: (1) simulation results by ES-based agents show the same tendency in game theory but those by LCS-based agents do not; and (2) even simulation results by ES-based agents become strange when the agents are restricted to handling only a real number with two decimal digits instead of an ordinary real number in a negotiation process between the agents in the bargaining game.

Keywords: multiagent-based simulation, cross-validation, modeling agents, bargaining game, learning mechanism, knowledge representation

1 Introduction

Recently, research based on multiagent-based simulation (Moss and Davidsson, 2001) or agent-based simulation (Axelrod, 1997) has attracted a lot of attention owing to not only improvements in computational power and simulation techniques but also to the availability of an alternative way of understanding complex social phenomena. Such research has employed computational techniques to provide tools for analyzing social dynamics and has contributed to the creation of theories by clarifying vague, intuitive, or under-specified issues in conventional approaches. Although, many useful implications have been found through computer simulation, the *validation*[1] of simulation results and computational models remains an open issue. To overcome these problems, Axtell and his colleagues claim the importance of investigating whether two different models can produce the same results in terms of validating the results and models (Axtell et al., 1996).[2]

It should be noted, however, that results derived by computer simulation are sensitive to how agents are modeled. This indicates that two models may not show the same results even if a different part between two models is very small. Examples include two models which difference is only *learning mechanisms* or *knowledge representation capabilities* of agents. To clarify such parts, this paper starts by comparing two computational models that are implemented differently in terms of the learning mech-

* Paper submitted to the 2002 Genetic and Evolutionary Computation Conference (GECCO'02)

[1] Burton argues that computational validity is a balance of three elements: (1) the question or purpose, (2) the computational model, and (3) the experimental design (Burton and Obel, 1995).

[2] They call this concept the "alignment of computational models" or "docking".

anisms and knowledge representation capabilities of agents.[3] Here, we call this type of investigation *cross-validation*,[4] which means to validate the results and models against different implementation of agents. Note that this differs from the usual meaning of cross-validation that checks the results of a single learning algorithm or a single knowledge representation on a reserved set of data. As a concrete domain, we employ a bargaining game (Muthoo, 2000) for this comparison, because this game is one of the fundamental examples and because rational behaviors of agents have already been analyzed in game theory (Osborne and Rubinstein, 1994).

This paper is organized as follows. Section 2 starts by explaining the bargaining game. A concrete implementation of agents is described in Section 3. Section 4 presents computer simulations and Section 5 discusses the cross-validation of results and models. Finally, our conclusions are made in Section 6.

2 Bargaining game

A bargaining game (Muthoo, 2000) was studied in the context of game theory (Osborne and Rubinstein, 1994). This study addressed the situation where two or more players (or agents) try to reach a mutually beneficial agreement through negotiations, and investigated when and what kinds of offers of an individual player can be accepted by other players.

To understand the bargaining game, let's give an example. In Rubinstein's work (Rubinstein, 1982), he illustrated a typical situation using the following scenario: two players, A_1 and A_2, have to reach an agreement on the partition of a "pie". For this purpose, they alternate offers describing possible divisions of the pie, such as "A_1 receives x and A_2 receives $1-x$ at time t", where x is any value in the interval $[0, 1]$. When a player receives an offer, the player decides whether to accept it or not. If the player accepts the offer, the negotiation process ends, and each player receives the share of the pie determined by the concluded contract. Otherwise, the receiving player makes a counter-offer, and all of the above steps are repeated until a solution is reached or the process is aborted due to some external reason (*e.g.*, the number of negotiation processes is finite or one of the players leaves the process). In the case that the negotiation process is aborted, both players can no longer receive any share of the pie.

Here, we consider the finite-horizon situation, where the maximum number of steps (`MAX_STEP`) in the game is fixed and all players know this information as common knowledge (Ståhl, 1972). In the case where `MAX_STEP` $= 1$ (also known as the *ultimatum game*), agent A_1 can make the only offer and then A_2 can accept or refuse it. If A_2 refuses the offer, both agents receive nothing. Since a rational agent is based on the notion of "anything is better than nothing", a rational A_1 tends to keep most of the pie to herself by offering only a minimum share to A_2. Since there are no further steps to be played in the game, a rational A_2 inevitably accepts the tiny offer. By applying a backward induction reasoning to the situation above, it is possible to perform simulation for `MAX_STEP` > 1. For the same reason of the ultimatum game, the agent who can make the last offer is better positioned to receive the larger share by offering a minimum offer (Ståhl, 1972). In this case, the last offer is granted to the agent that does not make the first offer if `MAX_STEP` is even, because each agent is allowed to make at most `MAX_STEP`$/2$ offers. On the other hand, the last offer is granted to the same agent that makes the first offer if `MAX_STEP` is odd.

After this session, we use the terms "payoff" and "agent" instead of the terms "share" and "player" for their wide meanings in the bargaining game.

3 Modeling agents

To implement agents in the framework of the bargaining game, this section starts by modeling a basic part of agents and then modeling the parts of our focus.

3.1 Modeling of a basic part

For a basic part of modeling agents, we implement the following components of agents as shown in Figure 1. Note that each agent has the same architecture.

< **Memory** >

- **Strategies memory** stores a set of strategies (the number of strategies is n in this figure), which consist of fixed numbers of pairs of offers (O) and thresholds (T), and the worth of the strategies (w). These strategies are similar to those used in (Oliver, 1996). The offer and threshold values are encoded by floating point numbers in the interval $[0, 1]$, while the worth values are calculated as averages of acquired payoffs. In this model, agents independently store different strategies, which are initially generated at random.

[3] Moss classifies the validation issues as (1) predictions, (2) agent and mechanism designs, and resulting outputs as descriptions (Moss, 2001). From this viewpoint, our focus is related to the second point in his classification.

[4] Carley also claims the same point using the term *cross-model validation* (Carley and Gasser, 1999).

- **Selected strategy memory** stores the one strategy selected to confront the strategy of an opponent agent. Figure 1 shows the situation where agent A_1 selects the xth strategy while agent A_2 selects the yth strategy.

< **Mechanism** >

- **Learning mechanism** varies both offer and threshold values in order to generate good strategies that acquire a large payoff. The detailed mechanism is described later.

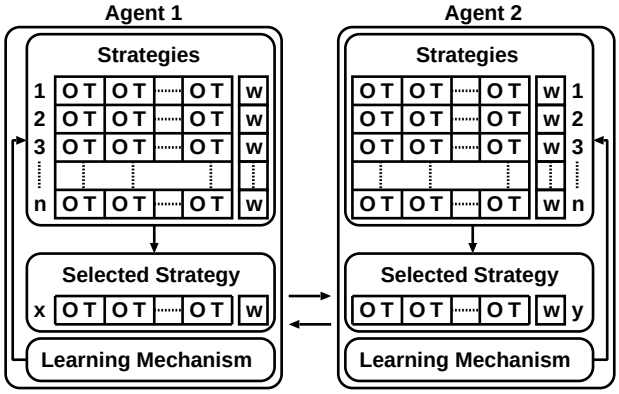


Figure 1: **Agent architecture**

As a concrete negotiation process, agents proceed as follows. Defining $\{O, T\}_i^{A_{\{1,2\}}}$as the ith offer or threshold value of agent A_1 or A_2, agent A_1 starts with the first offer $O_1^{A_1}$. Here, we count one *step* when either agent makes an offer. Then, A_2 accepts the offer if $O_1^{A_1} \geq T_1^{A_2}$; otherwise, it makes a counter-offer $O_2^{A_2}$, *i.e.*, the offer of A_2. This cycle is continued until either agent accepts the offer of the other agent or the maximum number of steps (`MAX_STEP`) is exceeded. To understand this situation, let's consider the simple example where `MAX_STEP`= 10, as shown in Figure 2. Following this example, A_1 starts by offering 0.01 to A_2. However, A_2 cannot accept the first offer because it does not satisfy the inequality of $O_1^{A_1}(0.01) \geq T_1^{A_2}(0.99)$. Then, A_2 counter-offers 0.01 to A_1. Since A_1 cannot accept the second offer from A_2 because of the same reason, this cycle is continued until A_1 accepts the 10th offer from A_2 where the offer satisfies the inequality of $O_{10}^{A_2}(0.01) \geq T_{10}^{A_1}(0.01)$. If the negotiation fails, which means exceeding the maximum number of steps, both agents can no longer receive any payoff, *i.e.*, they receive 0 payoff. Here, we count one *confrontation* when the above negotiation process ends or fails.

Furthermore, the worth of each strategy is calculated by the average of payoffs acquired in a fixed number

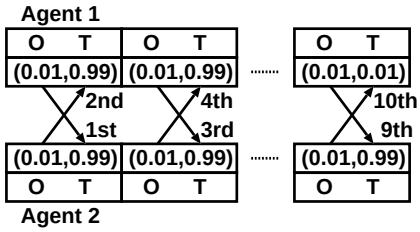


Figure 2: **An example of a negotiation process**

of confrontations (`CONFRONTATION`), where the strategies of the other agents are randomly selected in each confrontation. For example, the xth strategy of A_1 in Figure 1 confronts the randomly selected strategies of the other agents in the `CONFRONTATION` number of confrontations and then the worth of the xth strategy is calculated by the average of payoffs acquired in these confrontations. Since each agent has n number of strategies, the (`CONFRONTATION` $\times\, n \times 2$) number of confrontations is required to calculate the worth of all strategies of two agents. Here, we count one *iteration* when the worth of all strategies of two agents is calculated.

3.2 Modeling of parts of our focus

For our focus of modeling agents, we address the following two parts in modeling agents: (1) learning mechanisms and (2) knowledge representation capabilities.

3.2.1 Learning mechanisms

When implementing learning mechanisms of agents, we can consider several mechanisms. Among the many useful learning mechanisms, we employ the following: (1) evolutionary strategy (ES) (Back et al., 1992) and (2) learning classifier system (LCS) (Goldberg, 1989, Holland et al., 1986). The reasons for this employment are summarized as follows: (1) the ES mechanism performs well with a real number required to represent offer and threshold values in the bargaining game; and (2) the LCS architecture is implemented by modeling human beings (Holland et al., 1986) and several conventional research works employing LCS have already investigated social phenomena (*e.g.*, an artificial stock market (Arthur et al., 1997)). In detail, we employ the conventional $(\mu + \lambda)$ evolution strategies (ES) (Back et al., 1992) for ES and a Pittsburgh-style (Smith, 1983) classifier system instead of a Michigan-style (Holland, 1975) classifier system for LCS.

Under these learning mechanisms, a strategy, the worth of a strategy, and the strategies of an agent

as shown in Figure 1 correspond to a gene, a fitness, and a population in evolutionary computation (EC) literature, respectively. Based on these learning mechanisms, EC-based agents acquire good strategies by varying the numerical values of offer and threshold as shown by the following ordinary procedure: (1) a fixed number (μ or GENERATION_GAP$\times n$) of the best strategies (*parents*) remains in the set from one iteration to the next; (2) a fixed number (λ or GENERATION_GAP$\times n$) of new strategies (*offspring*) is produced from the set of parents at each iteration by applying the mutation operation in $(\mu+\lambda)$-ES and the crossover, mutation, and inversion operations in the Pittsburgh-style LCS; (3) new strategies replace the same number of strategies with low worth values. The detailed implementation of both learning mechanisms is described below.

- **Evolutionary strategy:** Two bargaining agents are equipped with their own $(\mu + \lambda)$-ES; the framework is based on the works of (Gerding et al., 2000, Bragt et al., 2000). When producing offspring strategies, the mutation operation adds to or subtracts from the offer and threshold values. These added and subtracted values are calculated from a Gaussian distribution with standard deviation σ, which is kept by each strategy. After these offspring strategies are produced, the standard deviations of the offspring are set as the averages of those in the parents at each iteration, while the standard deviations of the parents are maintained. Note that if an offer or threshold value becomes inappropriate (*e.g.*, a minus value or a value more than 1), it is reset to 0 or 1, whichever is closer to the current value.

- **Learning classifier system:** Two bargaining agents are equipped with their own Pittsburgh-style LCS. To conduct computer simulations in the same framework of ES, we set or modify each LCS as follows: (1) the LCS in our simulations applies the crossover operation at each iteration to produce offspring strategies the every iteration. Mutation and inversion operations are probabilistically applied to the offspring strategies generated by the crossover operation; (2) although the pair of offer and threshold can be considered as one *if-then* rule from the viewpoint of LCS, the selected order of these rules (pairs) is determined in advance; and (3) the concept of *don't care* is not employed in our simulations. For the second point, in particular, a preliminary research found that LCS-based agents cannot learn good strategies if they are allowed to select the rules (the pair of offer and threshold) in the ordinary LCS way.

3.2.2 Knowledge representation capabilities

When implementing agents, we have to consider their knowledge representation capabilities. In the bargaining game, in particular, a representation of strategies of agents must be considered, though there are no standard guidelines. From this fact, we start by employing the following two types of knowledge representation capabilities:[5] (1) an ordinary explanation of numbers (*e.g.*, $0.01\cdots$) and (2) a limited explanation of numbers, which are restricted to a real number with two decimal digits (*e.g.*, 0.01) in this simulation. We focus on this knowledge representation because (1) social scientists may take the latter case for a concise representation (Indeed, we have met such situations); and (2) a real number in offer and threshold values is critical in the bargaining game.

4 Simulation

4.1 Simulation design

Computer simulations are conducted to compare the following two cases. Note that the first simulation is performed without any restriction of a real number represented in strategies of agents (*i.e.*, the simulation with an ordinary real number).

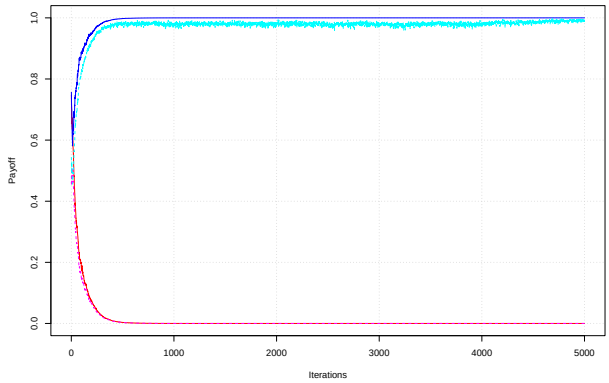
- **ES vs. LCS:** An investigation on the influence of different learning mechanisms of agents.
- **An ordinary real number vs. a real number of two decimal digits in ES:** An investigation on the influence of different knowledge representation capabilities of agents.

In each simulation, the following three cases are investigated. Note that all simulations are conduced until 5000 iteration and their results show average values over 10 runs.

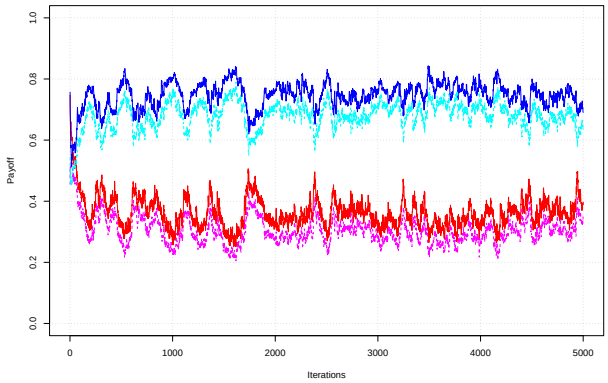
- **Case (a):** A payoff
- **Case (b):** An average negotiation process size
- **Case (c):** An accumulated number of each negotiation process size at the final (5000) iteration

As the parameter setting, the variables are set as follows. Note that preliminary examinations found that the tendency of the results does not drastically change according to the parameter setting.

[5] In addition to these issues, we should also investigate an influence of a modeling of strategies that are currently composed of the combination of offer and threshold as shown in Figure 1.

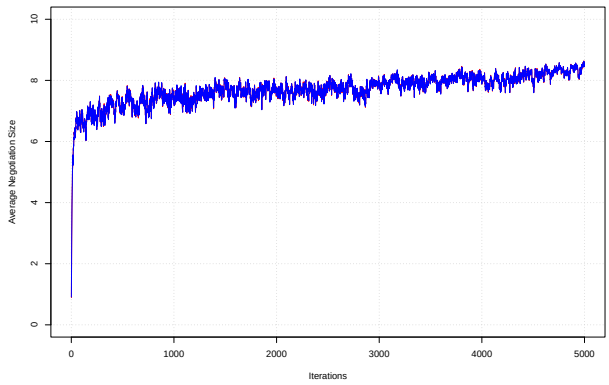


ES (a payoff)

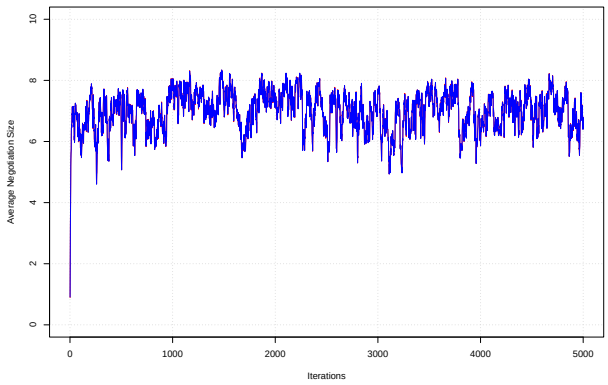


LCS (a payoff)

Case (a)

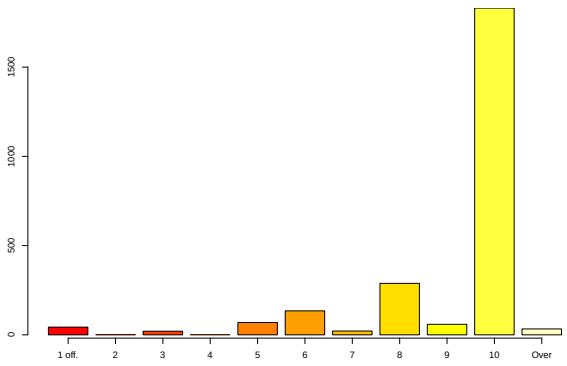


ES (average negotiation process size)

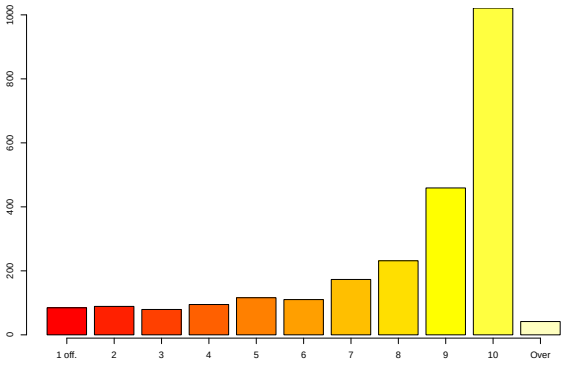


LCS (average negotiation process size)

Case (b)



ES (accumulated number of each negotiation process size at the 5000 iteration)



LCS (accumulated number of each negotiation process size at the 5000 iteration)

Case (c)

Figure 3: **Simulation results of ES vs. LCS: Average values over 10 runs at the 5000 iteration.**

- **Common parameters:** n (the number of strategies) is 50; `MAX_STEP` (the maximum number of steps in one confrontation) is 10; and `CONFRONTATION` (the number of confrontations for each strategy) is 20.
- **ES parameters:** μ (the parent population size) is 25; λ (the offspring population size) is 25; and σ (the initial standard deviation of a Gaussian distribution) is 0.5.
- **LCS parameters:** `GENERATION_GAP` (the percentage of replaced strategies) is 50%; `CROSSOVER_RATE` (the percentage of crossover operations) is 100%; `MUTATION_RATE` (the percentage of mutation operations) is 5%; and `INVERSION_RATE` (the percentage of inversion operations) is 5%.

4.2 Simulation results

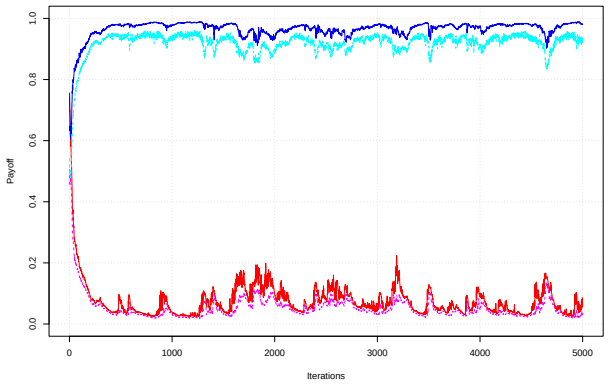
Figure 3 shows simulation results of both ES and LCS. In detail, figures (a), (b) and (c) indicate the results of the payoff, the average negotiation process size, and the accumulated number of each negotiation process size at the final (5000) iteration, respectively. The vertical axis in all of the figures indicates the indexes in the above three cases, while the horizontal axis in figures (a) and (b) indicates the iterations and the horizontal axis in figure (c) indicates the negotiation process size with negotiation failure represented as "Over" at the most right side. In particular, Figure 3(a) shows the payoff of agent A_1 in the lower lines and that of A_2 in the upper lines. The difference between the solid and light dash lines indicates that the former shows the best results and the latter shows the average results over 10 runs. Furthermore, Figure 4 shows simulation results of ES restricted to a real number with two decimal digits. Cases (a), (b) and (c) in Figure 4 has the same meaning of those in Figure 3.

From these results, we find that simulation results do not show the same tendency when different learning mechanisms or knowledge representation capabilities are applied to agents.

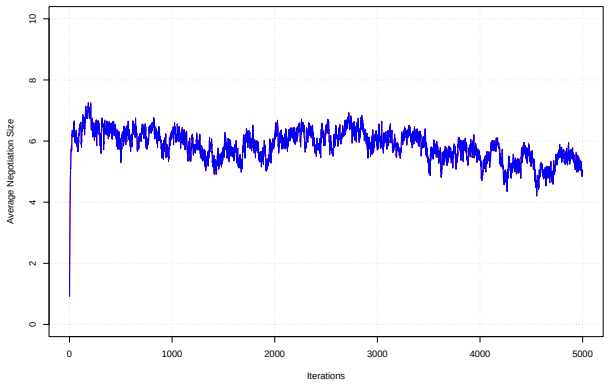
5 Discussion

5.1 Learning mechanisms

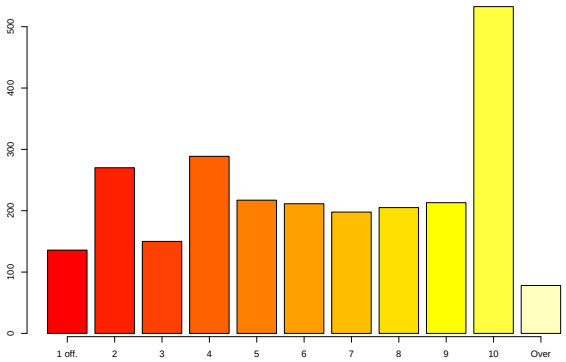
First, when focusing on the simulation results on different learning mechanisms of agents in Figure 3, the following implications are revealed: (1) the payoff of ES-based agents finally converges at the mostly maximum or minimum value (*i.e.*, 1 or 0), while that of LCS-based agents neither converges at a certain value



ES (a payoff)
Case (a)



ES (average negotiation process size)
Case (b)



ES (accumulated number of each negotiation process size at the 5000 iterations)
Case (c)

Figure 4: **Simulation results of ES with two decimal digits: Average values over 10 runs at the 5000 iteration.**

nor close to the maximum or minimum value; (2) the average negotiation size of ES-based agents increases, while that of LCS-based agents does not but simply oscillates; and (3) although the accumulated number of the 10th negotiation process size at the 5000 iteration is high both in ES-based and LCS-based agents, the accumulated number of another negotiation process size of ES-based agents is mostly close to 0, while that of LCS-based agents is not.

The reasons for the above results are summarized as follows: (1) the standard deviation of a Gaussian distribution in ES decreases as the iterations become large, while the crossover, mutation and inversion operations in LCS are constantly performed. Since most of these operations work as a divergent or explored factor, the decrease of such influence makes simulation results converge; (2) the offer and threshold values in all offspring are modified at every iteration in ES, while they are modified only by a mutation operation executed in a low probability in LCS. Furthermore, ES modifies such values like in a gradient search, while LCS modifies them randomly.

Here, we consider that game theory proves that rational agents A_1 and A_2 receive the maximum and minimum payoffs at the final negotiation process, respectively. This is because A_1 in our simulations has to accept any small offer proposed by A_2 at the 10th negotiation process; otherwise, A_1 cannot receive any payoff, *i.e.*, it receives 0 payoff. We therefore expect the following simulation results: (1) learning agents can acquire the maximum and minimum payoffs; (2) the average negotiation size increases if agents learn strategies appropriately; and (3) learning agents complete their negotiation process at the final offer of A_2 and the acceptance of A_1. In analyzing the simulation results according to the above three assumptions, we can consider that the ES-based agents show the same tendency in game theory but that LCS-based agents cannot. From this analysis, we can first conclude that simulation results are sensitive to the learning mechanisms applied to agents.

5.2 Knowledge representation capabilities

Next, when focusing on the simulation results on different knowledge representation capabilities of agents in Figure 3 (the normal case employing an ordinary real number) and Figure 4 (the restricted case employing a real number with two decimal digits), the following implications are revealed: (1) the payoff in the normal case finally converges at the mostly maximum or minimum value (*i.e.*, 1 or 0), while that in the restricted case does not completely converge; (2) the average negotiation size in the normal case increases, while that in the restricted case decreases; and (3) the accumulated number of each negotiation process size except the final (*i.e.*, the 10th) process in the normal case is mostly close to 0, while that in the restricted case is not.

To seek the reasons for the above different results, let's move our focus on to the 10th offer in Figure 2, where the values of offer and threshold are set here as 0.012 and 0.011, respectively. In this case, the agent who receives the offer from the opponent agent cannot accept it when employing an ordinary real number because the inequality of $O(0.012) \geq T(0.011)$ described in Section 3.1 is not satisfied. In contrast, the same agent accepts the offer when employing a real number with two decimal digits because the inequality of $O(0.01) \geq T(0.01)$ is satisfied. The same story can be told for other steps where both the offer and threshold values are close to each other. Due to this fact, agents have a possibility of accepting offers in each negotiation process. For this reason, agents with restricted knowledge representation capabilities cannot learn good strategies appropriately and thus they may accept unwilling (*i.e.*, small) offers in each negotiation process size. This increases the accumulated number of each negotiation process size except the final (the 10th) process in Figure 4(c) in comparison with that in Figure 3(c).

This finding indicates that simulation results can become strange when agents are restricted to handling only a real number with two decimal digits instead of an ordinary real number. Considering the fact that the previous section indicates that ES-based agents show the same tendency in game theory, we can secondary conclude that simulation results are sensitive to the knowledge representation capabilities of agents, even employing the same mechanism.

5.3 Essential factors for modeling agents

From the above analysis, both (1) learning mechanisms and (2) knowledge representation capabilities are important factors for cross-validating simulation results and computational models. This indicates that it is necessary to investigate such factors before investigating social phenomena arising from learning agent interaction.

6 Conclusion

This paper addressed cross-validation in multiagent-based simulation by analyzing evolutionary agents in a bargaining game. In particular, this paper focused

on analyzing different learning mechanisms and knowledge representation capabilities applied to agents for cross-validation. To investigate the importance of the above issues, we compared the following two cases: (1) agents employing an evolutionary strategy (ES) and agents employing a learning classifier system (LCS) as different learning mechanisms; and (2) agents handling an ordinary explanation of numbers and agents handling a limited explanation of numbers as different knowledge representation capabilities. Through an intensive comparison of the above simulation results, we found that both learning mechanisms and knowledge representation capabilities are important factors for cross-validating simulation results and computational models.

However, the results obtained in this paper do not cover all factors for cross-validation, and thus further careful qualifications and justifications such as experiments in other domains are needed to generalize our results. Such important directions must be pursued in the near future, but the following implications are potentially suggested from the current results: (1) simulation results by ES-based agents show the same tendency in game theory but those by LCS-based agents do not; and (2) even simulation results by ES-based agents become strange when the agents are restricted to handling only a real number with two decimal digits instead of an ordinary real number in a negotiation process between agents in the bargaining game.

Future research will include the following: (1) many simulations in other domains to generalize our results; (2) a comparison with other learning mechanisms such as reinforcement learning or other knowledge representations such as discrete numbers; (3) a validation of results and models with more than two agents; and (4) an investigation on the influence of the discount factor in ES- and LCS-based agents.

Acknowledgements

The research reported here was supported in part by a contract with the Telecommunications Advancement Organization of Japan entitled, "Research on Human Communication."

References

Arthur, W. B. et al. (1997). "Asset Pricing Under Endogenous Expectations in an Artificial Stock Market," in W. B. Arthur et al. (Eds.), *The Economy as an Evolving Complex System II*, Addison-Wesley, pp. 15–44.

Axelrod, R. M. (1997). *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*, Princeton University Press.

Axtell, R. et al. (1996). "Aligning Simulation Models: A Case Study and Results," *Computational and Mathematical Organization Theory (CMOT)*, Vol. 1, No. 1, pp. 123–141.

Bäck, T. et al. (1992). "Evolutionary Programming and Evolution Strategies: Similarities and Differences," *The 2nd Annual Evolutionary Programming Conference*, pp. 11–22.

van Bragt, D. D. B. et al. (2000). "Equilibrium Selection in Alternating-Offers Bargaining Models: The Evolutionary Computing Approach," *The 6th International Conference of the Society for Computational Economics on Computing in Economics and Finance (CEF'00)*.

Burton, R. M. and Obel, B. (1995). "The Validity of Computational Modes in Organization Science: From Model Realism to Purpose of the Model," *Computational and Mathematical Organization Theory (CMOT)*, Vol. 1, No. 1, pp. 57–71.

Carley, K. M. and Gasser, L. (1999). "Computational and Organization Theory," in Weiss, G. (Ed.), *Multiagent Systems – Modern Approach to Distributed Artificial Intelligence* –, The MIT Press, pp. 299–330.

Gerding, E. H. et al. (2000). "Multi-Issue Negotiation Processes by Evolutionary Simulation: Validation and Social Extensions," *CWI, Centre for Mathematics and Computer Science*, No. SEN R0024.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*, University of Michigan Press.

Holland, J. H. et al. (1986). *Induction*, The MIT Press.

Moss, S. and Davidsson, P. (2001). *Multi-Agent-Based Simulation*, Lecture Notes in Artificial Intelligence, Vol. 1979, Springer-Verlag.

Moss, S (2001). "Editorial Introduction: Messy Systems – The Target for Multi Agent Based Simulation" in (Moss and Davidsson, 2001), pp. 1–14.

Muthoo, A. (2000). "A Non-Technical Introduction to Bargaining Theory," *World Economics*, pp. 145–166.

Oliver, J. R. (1996). "On Artificial Agents for Negotiation in Electronic Commerce," Ph.D. Thesis, University of Pennsylvania.

Osborne, M. J. and Rubinstein, A. (1994). *A Course in Game Theory*, MIT Press.

Rubinstein, A. (1982). "Perfect Equilibrium in a Bargaining Model", *Econometrica*, Vol. 50, No. 1, pp. 97–109.

Smith, S. F. (1983). "Flexible Learning of Problem Solving Heuristics through Adaptive Search," *The 8th International Joint Conference on Artificial Intelligence (IJCAI '83)*, pp. 422–425.

Ståhl, I. (1972). *Bargaining Theory*, Economics Research Institute at the Stockholm School of Economics.