Genetic Programming in Wireless Sensor Networks

Derek M. Johnson, Ankur M. Teredesai, and Robert T. Saltarelli

{dmj3538,amt,rts9020}@cs.rit.edu
Computer Science Department
Rochester Institute of Technology, Rochester NY 14623, USA

Abstract. Wireless sensor networks (WSNs) are medium scale manifestations of a paintable or amorphous computing paradigm. WSNs are becoming increasingly important as they attain greater deployment. New techniques for evolutionary computing (EC) are needed to address these new computing models. This paper describes a novel effort to develop a series of variations to evolutionary computing paradigms such as Genetic Programming to enable their operation within the wireless sensor network. The ability to compute evolutionary algorithms within the WSN has innumerable advantages including, intelligent-sensing, resource optimized communication strategies, intelligent-routing protocol design, novelty detection, etc to name a few. In this paper we first discuss an evolutionary computing algorithm that operates within a distributed wireless sensor network. Such algorithms include continuous evolutionary computing. Continuous evolutionary computing extends the concept of an asynchronous evolutionary cycle where each individual resides and communicates with its immediate neighbors in an asynchronous time-step and exchanges genetic material. We then describe the adaptations required to develop practicable implementations of evolutionary computing algorithms to effectively work in resource constrained environments such as WSNs. Several adaptations including a novel representation scheme, an approximate fitness computation method and a sufficient statistics based data reduction technique lead to the development of a GP implementation that is usable on the low-power, small footprint architectures typical to wireless sensor motes. We demonstrate the utility of our formulations and validate the proposed ideas using a variety of problem sets and describe the results.

1 Introduction

Amorphous or paintable computers are very large arrays of low powered computers. Computers with a few hundred kilobytes of RAM and short range wireless communications are deployed with a density of tens to hundreds of elements per square centimeter. These computers are unreliable and have no global addressing scheme. This new genre of computing poses many new and interesting problems to the programmer and algorithm designer. How do you take advantage of a massively distributed computer whose individual elements are very

resource constrained? How do you write distributed algorithms without a global addressing scheme or predictable topology? William Butera, V. Michael Bove, and James McBride of the MIT Media Lab proposed a series of algorithms for performing media processing and storage on paintable computers [1]. Their paintable computer architecture is the basis for the one used in this research.

A paintable computing element runs process fragments (pfrags), containing code and data used as part of a global program. These fragments have read-write access to the home-page of the processor they are running on. A home-page contains key value pairs and is somewhat analogous to a tuple space. Process fragments have read-only access to the home-pages of proximal processing elements. Process fragments can migrate to neighboring processing elements.

Currently, no known implementations of paintable computing exist except validated simulations. On the other hand there is the wireless sensor network technology that has been gaining tremendous importance in recent years. Several problems that theoretically present themselves in paintable computing often can be manifested as challenges in wireless sensor network environments due to their similarity in terms of being highly resource constrained. In this paper we develop genetic programming solutions that effectively work in the wireless sensor network environment and demonstrate the utility of continuing this direction of research to realize the goals towards paintable computing.

There are innumerable technological hurdles that must be overcome for ad-hoc sensor networks to become practical. A single unit of WSNs is often termed as a 'mote'. The individual motes are incredibly resource constrained. They are characterized by a limited processing speed, storage capacity, and communication bandwidth. Moreover, their lifetime is determined by their ability to conserve power. Everything we take for granted in personal computing at the PC or desktop level comes at a huge premium in WSNs. All things considered, such constraints demand new hardware designs, network architectures, software applications, and therefore new learning algorithms that maximize the motes capabilities while keeping them inexpensive to deploy and maintain.

Developing GP solutions to work in WSNs is the primary focus of this work. Specifically, we describe the following contributions:

- A novel framework for performing genetic programming on a wireless sensor mote.
- A continuous algorithm to effectively evolve an in-network GP solution.

This paper is organized as follows: In the next section we outline the necessity for developing effective evolutionary computing solutions for wireless sensor networks. In section 3 we outline the changes and adaptations required to develop small-footprint GP. In section 3.1 we outline the details for developing a continuous algorithm that asynchronously computes a symbolic regression solution along with the distributed architecture it follows to do this computation. We then demonstrate the utility of our proposed algorithms by conducting experiments on a variety of problem sets and present the results. A brief discussion of these results concludes the paper.

2 Background and Related Work

There is significant interest within the GP community to derive effective formulations that help solve real world problems. The domain of wireless sensor networks and amorphous computing is one such real world domain that is gaining tremendous importance. GP solutions have been previously proposed for several problems that manifest themselves in sensory computing systems. Seok et al describe a technique to perform calibration of sensors using Genetic Programming on evolvable hardware [7]. Ziegler and Banzhaf proposed to use evolutionary techniques to develop a sensory nose for a robot [9]. The Mate system proposed by Levis is a tiny virtual machine for sensor networks [2]. The contribution in this paper is geared more toward adapting the GP system to work in a resource constrained environment and make decisions on the sensory data. For example, consider the problem of determining correlation between light and temperature signals in a sensor network. It is known within the signal processing community that these two parameters display similar variations under most environmental conditions. By adapting GP to work on an intelligent sensor node (termed mote) one can compute the exact correlation function that best describes the relationship between sensory attributes based on input data. Another example of a problem the proposed architecture can help address is the problem of optimized routing to save communication costs in WSNs. In this case, the proposed system can be instructed to compute an optimal routing path computed locally using the signal strength as an input parameter. Distributed systems that compute an optimization problem have been studied extensively and GP solutions have been proposed to solve problems in those domains [8]. Along similar lines, effective decision making using multi-agent teams was proposed by Luke et al [3]. From a computation environment perspective, Nordin et al explore ways to evolve machine code for embedded systems [6]. Our work extends these efforts and focuses on developing a GP system that effectively works in resource constrained environments.

With the MEMS revolution, micro-sensors are now following manufacturing curves that are at least related to Moore's Law. Such current trends in paintable or spray computing are summarized by Mamei [4]. They also highlight the need for intelligent in-network processing architecture such as the one we propose in this paper. Nagpal et al present a programming methodology for self-assembling complex structures from vast numbers of locally-interacting identically-programmed agents, using techniques inspired by developmental biology [5]. Our work is inspired by this effort to develop a GP paradigm that can later be extended to include self-assembly type optimization problems.

Basic and Parallel Evolutionary Algorithm

The basic evolutionary algorithm (BEA) is the most common model for evolutionary computation, however, it is clearly innapropriate for a wireless sensor network. If each mote were running a BEA they would likely take too long to

converge to be effective, nor would the algorithm exploit the parallel nature of a WSN.

Evolutionary algorithms tend to be highly parallelizable and many specific algorithms have been developed which take advantage of this. Two signifant models of parallel algorithms for evolutionary computing are the island model and the cellular evolutionary model:

Island Model has a separate evolutionary algorithm running on each available processing node. If the initial populations on each node are sufficiently randomized, the algorithm will explore different parts of the search space in parallel. Every few generations, one or more of the potential solutions in the population of each node is copied to one or more of the other neighboring nodes. These individuals are chosen using selection. This technique shares the best solutions, which usually contain good partial solutions, with other nodes in the same parallel algorithm. The good partial solutions are either introduced to different gene pools or, if already present, reinforce the good partial solution by increasing their influence. This model not only efficiently distributes the work of the evolutionary algorithm, it has been shown to produce better results then a single basic evolutionary algorithm with a greater population size.

Cellular Model[1] places each potential solution on a separate processing node. Each individual node can choose potential mates from neighboring nodes using some selection method. A typical cellular evolutionary algorithm chooses one population member each generation to participate in selection, crossover, and mutation and this member is replaced by its offspring. The cellular model is very effective on massively parallel computers such as vector computers.

3 Evolutionary Algorithms on a Mote

WSNs are an excellent target for distributed evolutionary computing. WSNs require learning algorithms that are capable of learning independent of the operation of other motes, but are also capable of using information available globally within the network to better optimize for local conditions. A distributed evolutionary algorithm can achieve both of these goals. Each mote can independently evolve, yet recombine genetic information from the surrounding motes to improve its suitability to the local environment.

In addition, evolutionary algorithms or learning algorithms in general must be designed to address the resource constraints present in a wireless sensor network, while taking advantage of the unique properties of a wireless network. Specifically, the ability to wirelessly broadcast information is a feature of wireless sensor motes not generally used in traditional distributed computing systems.

The parallel evolutionary algorithm described in this paper is based on traditional parallel algorithms like the Island Model [2], but modifications are made

[1] The cellular model is so named because it has been shown to simulate a certain class of cellular automata

to allow the algorithm to operate in a resource constrained wireless sensor network. Because wireless sensor motes are capable of broadcasting data, rather than point-to-point communication, we make use of this by distributing genetic information in a broadcast manner. This reduces the total bandwidth requirements, as well as converves the limited battery power of the wireless modes, as wireless transmitting is generally the most power hungry mode of these devices.

3.1 Broadcast-Distributed Parallel Genetic Programming

The Broadcast-Distributed Parallel genetic programming model (BDP) runs simultaneously on each mote in the network. The primary phases of the algorithm are fitness evaluation, genetic reproduction with local and remote genetic information and broadcast of genetic information. See Figure 1.

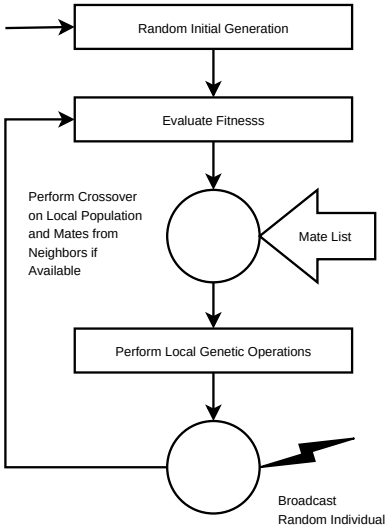

Fig. 1. Broadcast-Distributed Parallel Algorithm

The broadcast-distrbuted parallel evolutionary algorithm (BDP) is based on an island parallel model. Each mote carries its own population, and distributes genetic information in an asynchronous fashion. There is no need to have any physical clock on the motes running the algorithm. Exchanges of genetic information are made when the mote completes the computation of a each generation, which involves local reproduction, reproduction with local and remote genetic information, and finally calculating the fitness for the entire population. Conceptually, if the motes in a WSN running the BDP algorithm are too far apart to be able to communicate, each will operate as if they were running the BEA because they are unable to inject any external genetic information.

After each generation of the BEA on a mote M_b a random member of the population $\{M_b^{p_i} \mid i < |M_b^p|\}$ is selected and broadcast to remote motes[2]. The entire individual is sent. However, the size of any given individual is particularly small, on the order of a few bytes.

Each mote M_r within wireless range of a broadcasting mote (henceforth referred to as *neighbors*) receives the broadcast $M_b^{p_i}$ and appends it to a list of incoming genetic information (the *mate* list), M_r^m. When enough genetic material is received by a mote, that is when $|M_r^m| > a$, where a is some arbitrary constraint based on the available memory on the mote, a selection is performed on M_r^m and the internal population M_r^p and a crossover operation is performed.

$$crossover(M_r^{p_j}, M_r^{m_k}) \qquad \text{for some } j < |M_r^p|, k < |M_r^m|$$

No data is exchanged in the reverse direction from M_r to M_b as a result of this crossover operation. However, at the end of the next internal generation, mote M_r will broadcast a random mote as described above. Because M_r was a neighbor of M_b so that it received $M_b^{p_i}$ during a broadcast transmission, it is likely that M_b will remain a neighbor when their roles are reversed. Thusly, crossover will generally be equilibrious as exchanges will likely occur in both directions, although not necessarily in the same discreet generation.

It is worth noting that some motes will have an advantage if they have more neighbors. They will not only exchange more of their own genetic information, but they also will receive more genetic information, therefore they may exhibit quicker fitness improvements when compared with lone mote that lacks many neighbors.

The number of neighbors may vary within the set of motes with time. This modifies the chance a particular member is selected given its current position and the current time. This makes an examination of the survivability of a particular member in time difficult and has not yet been addressed but the overall result of selection is not affected. That is, the most fit member of a motes population is selected and its partial solutions survive. This maintains the selective pressure of the algorithm which causes the overall fitness of the population to improve. Since the individuals involved in crossover depend entirely and only on the results of selection then, as selection is fundamentally the same in a BDP and an BEA, crossover is also comparable.

3.2 Resource Constraints

The motes in a wireless sensor network are typically very low power compared to traditional PCs. They generally have have far less total storage, perhaps no secondary storage (i.e. disk storage), and may rely on the operating system, software and data all fitting in a small amount of solid-state primary storage.

[2] M_b is a broadcasting mote, M_r is a receiving mote, M^p is the population on a mote, M^m is the mate list on a mote

Many enhancements can be made to reduce the space requirements of both the evolutionary algorithm software binary, as well as the data representation and run-time memory requirements.

There are obvious memory usage improvements acheivable by using a steady state algorithm with an in place replacement strategy instead of a generational algorithm that replicates the entire population with each generation, before replacing the old population with the new; roughly half the amount of memory is needed when using a steady state algorithm.

The encoding strategy for individuals can significantly alter the memory requirements for a single mote in a BDP network. Particularly for evolutionary algorithms such as genetic programming, where the size of an individual is not fixed. It is necessary to set reasonable upper bounds on the allowable size of an individual, this is achieved by adding a limit to the allowable tree depth of a candidate solution.

It is also necessary to weigh the differences in using an interpreted language versus a compiled language to develop an implementation. An interpreted language (e.g. Perl, Lisp or Java) requires an entire virtual machine to be running on a mote; this would lead to a very sizeable increase in the memory footprint as well as executable code size. However a compiled language has the advantage of being customized and run to a specific platform, and therefore omits any penalties introduced by having a virtual machine present.

The obvious approach in a language offering dynamic memory allocation such as C is to store individuals as trees of dynamically allocated nodes in the program heap. Under this scheme each node is composed of a datum and two pointers to its left and right children. On the target system in question this solution weighs in at nine bytes per node.

Maintaining the program trees in blocks of statically allocated memory is attractive because the code for dynamically allocating memory (`malloc` and `free` in C) can be omitted from the final binary, provided dynamically allocated memory is not used elsewhere. Dynamic functionality is not available at all in the standard libraries of the smallest conceivable target platforms; it would require a significant increase in source code size.

The most compact memory usage for program tree storage is to using a constant size for each operator or terminal in the program tree. The offset of the left sub-tree is therefore constant, and can be used to form the structure of the tree. The offset for the right sub-tree is related to the size of the left sub-tree and can be calculated by recursing down the left sub-tree. Since most operations already involve an in order traversal of the program tree this representational scheme requires little additional code. This method is essentially a form of prefix notation. Because operators and terminals are well defined and each operator requires exactly two operands it is possible to evaluate the tree without any additional structure other than order.

The downside of this strategy is that any operation such as mutation or crossover which adjusts the size of the tree at anything other that the right most

leaf node will require a resizing of the entire data structure. This is a reasonable tradeoff in severely memory constrained motes.

By using a prefix notation and reducing the size of symbols in the program tree, we can reduce the memory requirements to just 5.6% of the memory consumed by most traditional GP representations using dynamic memory allocation. In terms of computation this method is no more expensive than any other representation for operations which otherwise require a traversal of the tree.

3.3 Memory Usage Requirements

By using the techniques described the memory requirements of the basic and parallel algorithms was significantly reduced. Equation 1 shows the total memory requirements for each node[3].

$$mem = b + v + p + i \tag{1}$$

where:

$$b \simeq 6\text{kB} \quad \text{program binary}$$

$$v = |vars| \times \text{sizeof(float)} \quad \text{input variables}$$

$$p = |pop| \times (2^{depth_{max}} - 1) \times \text{sizeof(tree node)} + (2 \times \text{sizeof(uint)}) \quad \text{population}$$

$$i = |mates| \times (2^{depth_{max}} - 1) \times \text{sizeof(tree node)} + (2 \times \text{sizeof(uint)}) \quad \text{mate list}$$

For typical parameter sizes the memory requirements of the algorithm are tenable even on very low-powered WSN devices. The memory requirements are even more impressive when compared with parallel GP implementations that make little or no attempt to restrict memory usage. See Table 1.

population on mote	typical parallel GP memory usage[4]	BDP memory usage	percentage of typical
50	187kB	14.4kB	7.7%
100	296kB	20.4kB	6.8%
200	514kB	32.5kB	6.3%
500	1168kB	68.9kB	5.9%

[4] for a standard parallel GP implementation with program binary ∼80kB and using 9B per tree node

Table 1. Typical Memory Requirements

[3] For the TinyOS architecture: sizeof(float) = 32 bits, sizeof(tree node) = 4 bits, sizeof(uint) = 16 bits

3.4 Improving Training Efficiency

The major impediment to compute an evolutionary algorithm over large datasets is the the amount of data required to be present in-memory while training. We attempt to alleviate this problem by reducing the amount of training data necessary to achieve converge. This involves reducing the training set to a minimal set of variant data. The training set must be diverse enough to encompass the entire search space, but also sparse enough to avoid over training on any particular area of the search space.

In an attempt to minimize memory usage due to training data storage while maintaining speed and scalability we examined a clustering approach to training data sampling. Test data was generated by choosing cluster centroids and creating a random set of points within a certain distance of those centroids. The number of points per cluster and the number of clusters are variable.

Figure 2 shows a visualization of the full data from one test run, consisting of 500 training data points over four variables. In creating the three dimensional image the fourth value and correct answer are ignored, however the images serve to represent both the clustered nature of the training data and the significant reduction in the number of points thanks to the clustering approach.

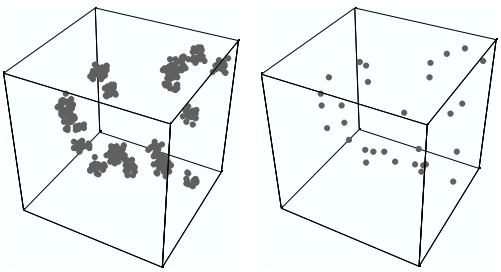

Fig. 2. Result of Clustering to Reduce Training Data

4 Performance Analysis

Both the BEA and BDP algorithms were run with a variety of equations, ranging from simple 3 variable to large 10 variable symbolic regression problems. The convergence properties of both algorithms were experimentally measured. The number of motes in the network was varied as well as the population on each mote.

It is clear that the broadcast distribution does not have a negative effect on the ability of the algorithm to converge on a solution. The evolutionary pressure is still sufficient, and the distributed algorithm significantly outperforms a single BEA running with the same population size indicating that the effect of broadcast distribution is positive.

Because generations is no longer a valid term when referring to a BDP we instead refer to the count of genetic operations (mutation and crossover) either on a single mote or all of the motes in the network.

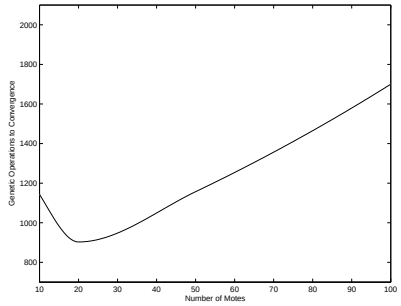

Fig. 3. Varying the Number of Motes in Network

Figure 3 shows that more genetic operations are performed before converging on the solution when the number of motes in the network is increased. This is intuitive; when there are more motes in the network, each is independently computing, resulting in a greater number of genetic operations as a whole, before a solution is found. This does not mean that the time required for the network to converge increases, because as more motes are added, more of the genetic operations are being done in parallel.

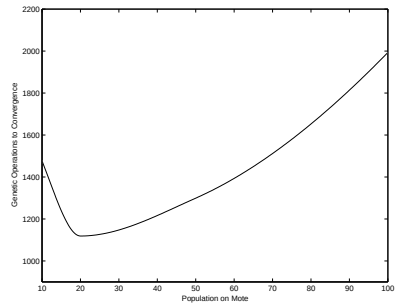

Fig. 4. Varying the Population on Each Mote

The same effect is also observed with the population size is increased on each mote. Figure 4 shows this behavior.

Figure 5 show the convergence properties of the BEA and BDP. The graphs are an aggregate of the run results of several different problems. The BDP is slower to improve fitness, but makes more steady progress towards an optimal solution.

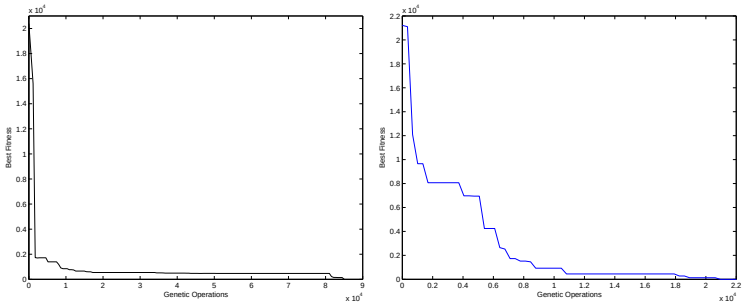

Fig. 5. Convergence of the BEA and BDP Algorithms

Problem	BEA	BDP
P1	6692	8350
P2	7207	9208
P3	84942	18828
P4	130759	23649
P5	315275	43997
P6	DNF	45756
P7	DNF	99893

Table 2. Aggregate Genetic Operations Until Convergence Over Various Symbolic Regression Problems

It is also worth noting that we observed the BDP algorithm with total population size p where each mote has a population of $|M|/p$ to be less prone to stagnation than a population of size p running the BEA algorithm. This is due to the propensity of good solutions to distribute slowly throughout the network, this mitigates factors that can occasionally over-emphasize highly fit solutions in the BEA. The effect of this is shown in Table 2, the problems that could not be solved by the BEA in a reasonable amount of time were due to over-emphasis of highly fit but sub-optimal solutions.

5 Conclusions and Future Work

In this work we argue that broadcast-distributed parallel genetic programming shows promise as a model for evolutionary computing on wireless sensor networks. We show empirically via simulations that it is possible to use the broadcast nature of communication between motes to improve the ability of single motes to find a solution via exchange of genetic information across the network. It is also possible to build GP implementations that are practicable on resource constrained WSN motes.

Future work will attempt to determine how well BDP will allow motes to evolve solutions that are ideal for their local conditions, and whether such motes will benefit from the receipt of genetic information for neighboring motes that

likely share similar conditions. These experiments will also focus on obtaining data from actual motes in real environments.

References

[1] William Butera, V. Michael Bove Jr., and James McBride. Extremely distributed media processing. In *Proceedings of SPIE Media Processors*, 2002.

[2] P. Levis and D. Culler. Mate: A tiny virtual machine for sensor networks. In *International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA*, Oct. 2002. To appear.

[3] Sean Luke and Lee Spector. Evolving teamwork and coordination with genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 150–156, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[4] Marco Mamei and Franco Zambonelli. Spray computers: Frontiers of self-organization for pervasive computing. Web: http://polaris.ing.unimo.it/Zambonelli/spray.html.

[5] R. Nagpal, A. Kondacs, and C. Chang. Programming methodology for biologically-inspired self-assembling systems. In *AAAI Spring Symposium on Computational Synthesis, March 2003.*, 2003.

[6] Peter Nordin, Wolfgang Banzhaf, and Francone Francone. Efficient evolution of machine code for CISC architectures using instruction blocks and homologous crossover. In Lee Spector, William B. Langdon, Una-May O'Reilly, and Peter J. Angeline, editors, *Advances in Genetic Programming 3*, chapter 12, pages 275–299. MIT Press, Cambridge, MA, USA, June 1999.

[7] Ho-Sik Seok and Byoung-Tak Zhang. Evolutionary calibration of sensors using genetic programming on evolvable hardware. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 630–634, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 May 2001. IEEE Press.

[8] Ivan Tanev and Katsunori Shimohara. On role of implicit interaction and explicit communications in emergence of social behavior in continuous predators-prey pursuit problem. In *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 74–85, Berlin, 12-16 July 2003. Springer-Verlag.

[9] Jens Ziegler and Wolfgang Banzhaf. Evolving a "nose" for a robot. In *Evolution of Sensors in Nature, Hardware, and*, pages 226–230, Las Vegas, Nevada, USA, 8 July 2000.