A Sequential Similarity Metric for Case Injected Genetic Algorithms applied to TSPs

Sushil J. Louis
Genetic Adaptive Systems LAB
Dept. of Computer Science
University of Nevada
Reno, NV 89557
sushil@cs.unr.edu

Yongmian Zhang
Genetic Adaptive Systems LAB
Dept. of Computer Science
University of Nevada
Reno, NV 89557

Abstract

We present and use a sequence similarity metric to solve sets of similar problems with case injected genetic algorithms. Rather than starting anew on each problem, we periodically inject a genetic algorithm's population with *appropriate intermediate solutions* to similar, previously solved problems. Using simple syntactic similarity measures, our experimental results from optimizing a series of traveling salesman problems demonstrates the robustness of our approach. Results show that compared to a randomly initialized genetic algorithm, our system learns to take *decreasing time* to provide *better solutions* to a new problem as it gains experience from solving other similar problems.

1 INTRODUCTION

Problems seldom exist in isolation. Any useful system must expect to confront many related problems over its lifetime and we would like such a system to improve its performance with experience. Such a learning system requires memory; a place for storing past experiences to guide future operations. Genetic algorithms (GAs) are randomized parallel search algorithms that search from a population of points (Holland, 1975; Goldberg, 1989). Current genetic algorithm based machine learning systems use rules to store past experience to improve their performance over time (Holland, 1975; Goldberg, 1989; Smith, 1985; Janikow, 1993; Grefenstette et al., 1990). However, many application areas, especially in the design domain, are more suited to a case-based storage of past experience (Mostow et al., 1992; Huhns and Acosta, 1992; Sycara and Navinchandra, 1992; Goel and Chandresekaran, 1992). In this paper, we describe a system that combines genetic algorithm search with case-based memory to learn, with experience, to improve performance on related tasks. We then introduce and evaluate a *domain independent* similarity metric for indexing cases/solutions stored in case-based memory.

Typically, a genetic algorithm randomly initializes its starting population so that the GA can proceed from an unbiased sample of the search space. However, as pointed out earlier, problems do not usually exist in isolation and we often confront sets of similar problems. It makes little sense to start a problem solving search attempt from scratch when previous search attempts may have yielded useful information about the search space. Instead, periodically injecting a genetic algorithm's population with *relevant* solutions or partial solutions to similar previously solved problems can provide information (a search bias) that reduces the time taken to find a quality solution. Our approach borrows ideas from case-based reasoning (CBR) in which old problem and solution information, stored as cases in a case-base, helps solve a new problem (Riesbeck and Schank, 1989). In our system, the data-base, or case-base, of problems and their solutions supplies the genetic problem solver with a long term memory. The system does not require a case-base to start with and can bootstrap itself by learning new cases from the genetic algorithm's attempts at solving a problem.

The case-base does what it is best at – memory organization; the genetic algorithm handles what it is best at – adaptation. The resulting combination takes advantage of both paradigms; the genetic algorithm component delivers robustness and adaptive learning while the case-based component speeds up the system.

The **C**ase **I**njected **G**enetic **A**lgo**R**ithm (CIGAR) system presented in this paper can be applied in a number of domains from computational science and engineering to operations research (Louis and Johnson, 1997).

Problems in planning, scheduling, circuit design, VLSI layout, architecture, and most kinds of engineering design can be cast as search problems and thus suitable for GAs. Since such problems seldom exist in isolation, many of these application areas are thus also appropriate for CIGAR (Louis, 1993).

We discuss related work in the next section then describe our system. Section 4 explains our similarity metric and we use this similarity metric in applying our system to traveling salesman problems in section 5. The last section presents conclusions and directions for future work.

2 RELATED WORK

Reusing old solutions has been a traditional performance improvement procedure. One early attempt can be found in Ackley's work with SIGH (Ackley, 1987). Ackley periodically restarts a search in an attempt to avoid local optima and increase the quality of solutions. Eshelman's CHC algorithm, a GA with elitist selection and cataclysmic mutation, also restarts search when the population diversity drops below a threshold (Eshelman, 1991). Other related work includes Koza's automatically defined functions (Koza, 1993) and Schoenauer's constraint satisfaction method (Schoenauer and Xanthakis, 1993). More recently, Ramsey and Grefenstette come closest to our approach and use previously stored solutions to initialize a genetic algorithm's initial population and thus increase a GA's performance in an anytime learning environment that changes with time (Ramsey and Grefensttete, 1993; Grefensttete and Ramsey, 1992). Automatic injection of the best solutions to previously encountered problems biases the search toward relevant areas of the search space and results in the reported consistent performance improvements. Sheppard and Salzburg combined memory-based learning with GAs in finding better plans for the pursuer-evader game within a reinforcement learning framework (Sheppard and Salzburg, 1995). Their experiments indicate that the combined approach performed better than either approach alone. To our knowledge, the earliest work in combining GAs and case-based reasoning was done by Louis, McGraw, and Wyckoff who used case-based reasoning principles to explain solutions found by GA search (Louis et al., 1992; Louis et al., 1993). Further work in this area solved pairs of similar problems with a clear performance advantage for the combined system (Louis and Johnson, 1997). Louis' work showed that choosing appropriate cases for injection reduced problems with premature convergence and used a probabilistic similarity measure to select appropriate cases to seed the initial population of the genetic algorithm.

The work reported in this paper differs in that 1) we solve *sets* of similar problems rather than one problem, 2) the system described in the next section uses encoded solution similarity which is domain independent in contrast to domain dependent problem similarity measures, and 3) we periodically inject solutions into the running GA instead of only biasing the initial population.

3 THE SYSTEM

Figure 1 shows a conceptual view of CIGAR. We periodically inject a small number of *solutions* similar to the current best member of the GA population (closest to the best) into the *current* population, replacing the worst members. The GA continues searching with this combined population. During GA search, whenever the fitness of the best individual in the population increases, the new best individual is stored in the case-base.

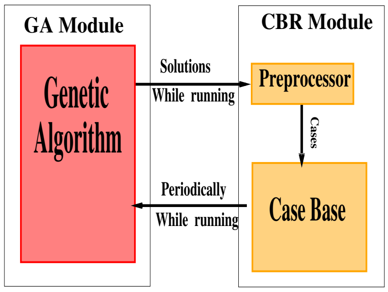

Figure 1: Conceptual view of CIGAR.

Genetic algorithms exploit and process similarities (schemas) in the encoded genotype. The problem is defining a similarity metric in terms of the GA's encoding which will will allow us to find and inject solutions *similar* to the current best member of the population. A solution (or GA encoding) similarity metric also avoids the domain (application) dependent issue of defining similarity among problems – a difficult issue in CBR systems. Hamming or euclidean distance suffices for problems where the meaning of a gene depends on its location in a chromosome. This paper proposes a similarity metric for genetic algorithm encodings where meaning also depends on the order or sequence of alleles in the chromosome. We use the traveling salesperson problem as our sequential coding example and evaluate CIGAR's performance on several TSP benchmarks.

4 SIMILARITY METRIC

We borrow from work on DNA sequencing to define our similarity metric for order based representations in case injected genetic algorithms. Without loss of generality, we restrict our attention to strings encoded as a sequence of numbers. We assume that the solutions of two problems are encoded as two strings $X[x_1x_2..x_n]$ of length n and $Y[y_1y_2..y_m]$ of length m respectively. We further assume that the elements x_i, for $i = 1, 2, ..n$ and y_i, for $i = 1, 2, ..m$ are natural numbers drawn from a finite sequence $\mathcal{N}$. Two solutions are similar to each other, if $D(X, Y) \leq \theta$, where θ is given distance threshold and $D(X, Y)$ is a distance between X and Y as measured by our distance metric. In DNA matching the distance metric is based on the (minimum) number of transformation needed to transform one string to another. Levenshtein defined three basic operations needed for this transformation (Levenshtein, 1966).

1. Substitution(Sub): replace an element of X by an element of Y.
2. Deletion(Del): delete an element of X or
3. Insertion(Ins): insert an element of Y.

The distance between two solutions/strings is thus the minimal number of such operations needed to transform one string (tour) to the other.

We quantize these operations as follows:

- $Sub(x_i, x_i) = 0$;
- $Sub(x_i, y_j) = 1$ for $x_i \neq y_i$ and $i < n$ and $j < n$;
- $Sub(x_i, y_j) = 0$ for $i > n$ or $j > m$;
- $Del(x_i) = Ins(y_j) = 1$ for $i < n$ and $j < m$;
- $Del(x_i) = Ins(y_j) = 0$ for $i > n$ and $j > m$.

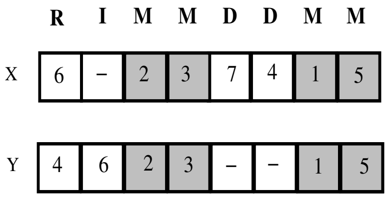

Figure 2: String X can be edited to become string Y. D denotes the delete operation; I denotes the insert operation; R the replace operation, and M defines matches.

A match of string X and Y consists of an alignment between X and Y using the three basic operation as illustrated in Figure 2. The problem can be formally stated in terms of a directed acyclic graph $G = \{(V, E) : u, v \in V, (u, v) \in E\}$ and weights $w(u, v)$ associated with the three operations described above. Finding the distance between two strings can then be viewed as the problem of minimizing the sum of weights when finding an acyclic subset $T \subset E$ whose start vertex is $(0, 0)$ and destination vertex is associated with (n, m).

$$w(T) = \sum_{(u,v)\in T} w(u, v) \tag{1}$$

The similarity s between the solutions of two problems is then defined in percentage as

$$s = \left(1 - \frac{w(T)}{min(n, m)}\right) \times 100 \tag{2}$$

4.1 Similarity For TSPs

In the remainder of the paper we use the traveling salesman problem (TSP) to illustrate the the use of this metric with CIGAR. We believe that this metric can be used to catalyze the convergence of other optimization problems which use order-based encodings.

The TSP can be defined as follows. Let $G = (V, E)$ be a graph where V is a set of vertices and E is a set of arcs between vertices and each arc is associated with a non negative cost. The TSP consists of finding the tour of minimum length that passes through every vertex exactly once. The cost function can be defined as

$$\sum_{i=1}^{n} d_i v(i) \tag{3}$$

where n is the number of cities, $v(i)$ is the successor of city i and d denotes the distance of two cities.

5 METHODOLOGY AND RESULTS

Our first series of experiments determined the number of solutions to inject (injection percentage) every generation. We use a 76 city TSP as a basis from which to generate a set of 50 similar TSPs to use a testbed for testing injection. Results, illustrated in Figure 3, indicate that replacing 15% of the worst individuals in the current population by solutions most similar to the current best member of the population, provides good performance.

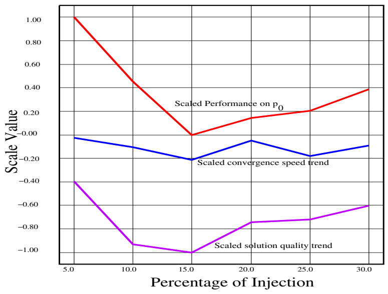

Figure 3: Performance with different amount of injection across 50 TSPs.

For each of our selected benchmarks from the TSP library, we generated 50 similar TSP problems denoted $P = \{p_1, p_2, ..., p_n\}$, for $1 \leq n \leq 50$ to examine CIGAR's performance under the proposed similarity metric (Reinelt, 1996). Each problem consists of a sequence of vertices $(v_1, v_2, ..., v_n)$, where v_i is given by its x- and y-coordinates (x_i, y_i). We compare performance against a Randomly Initialized Genetic Algorithm (RIGA) across the same set of problems using the same parameters as CIGAR. We generate the first 49 problems using a Gaussian distribution with parameters

$$P = G(p_0, \sigma) \tag{4}$$

where $G(p_0, \sigma)$ denotes the Gaussian distribution with mean number of cities p_0 and standard deviation σ; p_0 is the number of cities on the particular benchmark. The problems are generated by first drawing the new number of cities (p_i) from the Gaussian distribution and then randomly deleting or adding cities from or to p_0 to bring the number of cities to p_i. Cities are added at randomly generated coordinates limited by the maximum and minimum city coordinates in p_0. We always assign problem p_0 to be the last problem attempted in P since we know its optimal tour length (or best known tour length) from the literature. Let p_0 be called the *reference problem* and thus $P = \{p_1, p_2, ..., p_{n-1}, p_0\}$.

Many simulations were conducted to evaluate the performance of the system under varying problem sets. Each problem set was executed over 20 times, each time starting with a different initial population generated randomly (different random seed) so that an accurate ensemble average of the results could be obtained. We chose GA parameters that achieved good results on our initial tests and used these parameters for all experiments. The parameters are: (1) population size: 100; (2) number of generation: 150; (3) mutation probability: 0.05; (4) crossover probability: 0.99; (5) Selection Scheme: CHC (Eshelman, 1991).

We defined two performance metrics to compare the performance of CIGAR and RIGA: 1) The solution quality which is the average length of the best tours found over all runs and 2) the average time taken to find these solutions. We fit these performance measures on consecutive problems over time to a line $y = ax + b$ and then utilized the line slope a as an indicator of the system's performance trend as we solved more problems. This line is shown in most of our performance graphs.

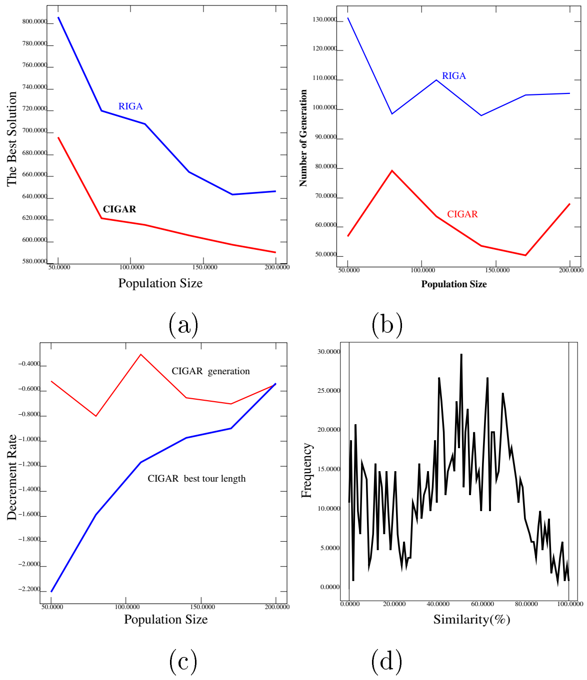

Figure 4: Overall performance of CIGAR vs. RIGA under different population size

We next investigated the effect of population size on performance and examined solution quality and time taken to find the best solution on problem p_0 for different population sizes. Note that we are solving p_0 after attempting 49 other similar problems. A plot of the output quality and time to the best solution with various population sizes is given in Figure 4. Figure 5 visually depicts the overall performance of both CIGAR and RIGA as a function of problems being solved when population size is 50 and 200, respectively. The *reference problem* p_0 in this example is the 76 city problem, while the other 49 problems are generated by using a Gaussian deviation $\sigma = 8$. The sizes of the 49 problems varies from 55 to 95 cities. Figure 4(d) shows the distribution of similarities among the cases/tours in our case base.

Several important observations can be made from these results. First, over all our experiments CIGAR

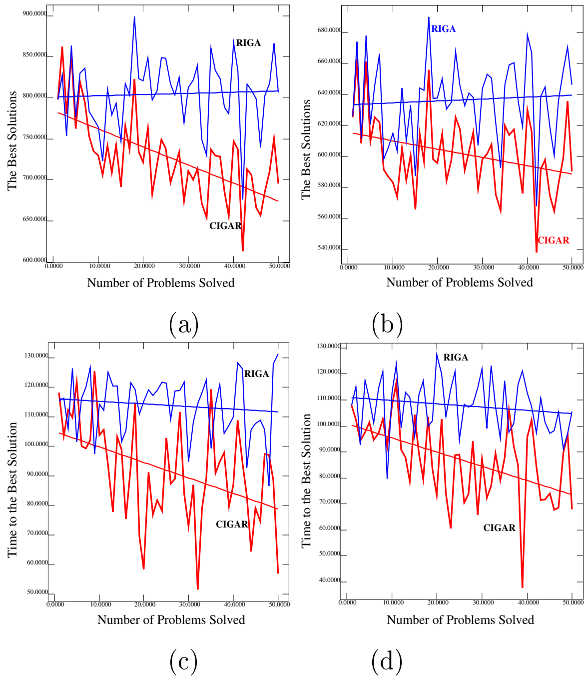

Figure 5: CIGAR vs. RIGA. (a)(c) population size 50. (b)(d) population size 200.

always outperformed the corresponding RIGA over the set of problems solved. We can see from Figure 5 that CIGAR learns to take *decreasing* time to provide a solution to a new problem as it stores and reuses information from solving similar problems. In addition, the solution quality also shows the same trend. Our similarity metric is evidently effective, since CIGAR's performance improves with experience and CIGAR works by looking for cases that are similar (according to our metric) to the current best individual in the case-base. Thus CIGAR, using relevant solutions or partial solutions of previous solved problems, can catalyze convergence on new problems and guide search away from (probably) unimportant regions of the search space.

Second, Figure 4(a)(c) show that, for CIGAR, solution quality improvement when solving new problems is especially significant when population size is smaller. Given a threshold population size that produces acceptable results, CIGAR does not need larger populations sizes to produce better results as long as it can gain experience from solving similar problems. That is, CIGAR when trained well, can produce better results without using additional resources. The randomly initialized GA needs more resources to produce better results – the exact opposite. It is reasonable to conclude then, that with experience, as CIGAR runs with a particular population size, it is able to achieve performance equivalent to larger population sizes for the randomly initialized GA.

Finally, Figure 4(b)(c) which depicts the relationship between the average number of generation to the best solution and population size shows that compared with RIGA, CIGAR always takes less time to approach the best solution.

We also constructed problem sets with poor similarity among member problems, to examine change in performance over time. Figure 6 plots the result for $p_0 = 76$ and $\sigma = 10$. Figure 6(a) shows that, CIGAR still manages an improvement in quality after solving about 15 problems. A similar plot of time to the best solution as a function of the number of problems solved is given in Figure 6(b), showing the same trend (but after a larger number of problems are attempted). Figure 7 and Figure 8 plot our performance metrics for $p_0 = 76$ across a range of deviations. The figures (c) also depict the distribution of similarity among the tours found by CIGAR.

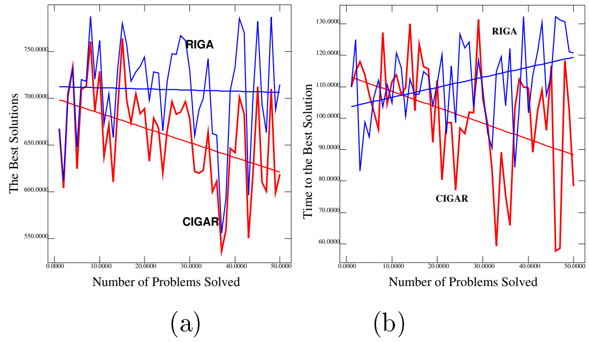

Figure 6: CIGAR performance on poor similarity problem set

Table 1: Performance under different σ ($p_0 = 105$)

Problem Dev σ	CIGAR Best Tour Length	CIGAR Avg # of Gens	RIGA Best Tour Length	RIGA Avg # of Gens
12	18831	95	20269	127
16	17530	66	20054	87
20	17967	83	19826	112
24	17359	88	20137	102

These general trends hold across a range of problem distributions and sizes. Table 1 lists the the convergence speed and solution quality on the $p_0 = 105$ problem for various deviations (σ). Table 2 fixes the deviation (σ) and lists performance across TSPs with different sizes. These results provide further evidence that CIGAR improves both performance measures with experience and obtains statistically significant improvements over the randomly initialized genetic algorithm.

Table 2: Performance under different p_0 ($\sigma = 20$)

Refer. Problem P_0	CIGAR		RIGA	
	Best Tour Length	Avg # of Gens	Best Tour Length	Avg # of Gens
$TSP124$	67606	55	84668	100
$TSP136$	125144	67	143087	100
$TSP144$	70678	20	86143	89
$TSP152$	91891	79	110138	115

Table 3 and Table 4 show the rate of change in performance by providing the slope for the line of best fit (linear regression). Table 3 shows the change in slope across different values of σ for $p_0 = 105$. The second and fourth columns list the slope for solution quality. Columns three and five show convergence speed.

Table 3: Performance with different σs ($p_0 = 105$)

Problem Deviation σ	CIGA		RIGA	
	Quality slope	Time slope	Quality slope	Time slope)
8	-10.8	-0.22	7.6	0.06
12	-12.6	-0.56	7.0	0.04
16	-29.3	-0.69	24.9	-0.03
20	-31.0	-0.60	8.3	-0.11
24	-46.0	-0.39	-0.29	-0.06

Table 4: Performance with different p_0 ($\sigma = 20$)

Reference Problem size	CIGAR		RIGA	
	Quality slope	Time slope	Quality slope	Time slope
$TSP124$	-234.99	-0.73	56.49	-0.034
$TSP136$	-367.86	-0.70	-82.32	0.145
$TSP144$	-227.25	-1.23	-34.89	-0.068
$TSP152$	-353.47	-0.49	-82.53	0.088

These tables show that CIGAR always improves performance when compared with RIGA (CIGAR's performance slopes are more negative). RIGA may have negative slope since the 50 problems attempted may be more or less difficult depending on the structure of the problems generated by our procedure. However, CIGAR's slope always being more negative than RIGA's indicates that, as expected, CIGAR learns to exploit similarities among problems to provide increasing performance with increasing experience. Our similarity metric is effective.

6 CONCLUSIONS

We presented a similarity metric for order-based representations in CIGAR systems. The metric, based on DNA sequencing research, allows us to extend the application of CIGAR systems to problems with order-based encodings in genetic search. Using this representation we showed that, with experience, CIGAR systems learn to improve performance on sets of similar problems. Although we only considered synthetically generated classical TSP problems, we suggest that this similarity measure and the CIGAR approach can be extended to other order-encoded problems in genetic search. We believe that the positive results obtained open the way for further research in this area.

Other work on problems with location dependent but order independent encodings indicates that simple distance metrics like hamming and euclidean distance work well for CIGAR applications. We are currently applying the CIGAR approach to problems in engineering design, object identification, and spectral analysis. These problems will test CIGAR's robustness and applicability in the real-world.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 9624130.

References

Ackley, D. A. (1987). *A Connectionist Machine for Genetic Hillclimbing.* Kluwer Academic Publishers.

Eshelman, L. J. (1991). The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In Rawlins, G. J. E., editor, *Foundations of Genetic Algorithms-1*, pages 265–283. Morgan Kauffman.

Goel, A. and Chandresekaran, B. (1992). Case-based design: A task analysis. In Tong, C. and Sriram, D., editors, *Artificial Intelligence in Engineering Design, Vol II*, pages 165–184. Academic Press, Inc.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley.

Grefenstette, J., Ramsey, C., and Shultz, A. (1990). Learning sequential decision rules using simulation models and competition. *Machine Learning*, 5:355–381.

Grefensttete, J. and Ramsey, C. (1992). An approach to anytime learning. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 189–195, San Mateo, California. Morgan Kauffman.

Holland, J. (1975). *Adaptation In Natural and Artificial Systems*. The University of Michigan Press, Ann Arbour.

Huhns, M. and Acosta, R. (1992). Argo: An analogical reasoning system for solving design problems. In Tong, C. and Sriram, D., editors, *Artificial Intelligence in Engineering Design, Vol II*, pages 105–144. Academic Press, Inc.

Janikow, C. Z. (1993). A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13:189–228.

Koza, J. R. (1993). *Genetic Programming*. MIT Press.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions insertions and reversals. *Soviet Physics-Doklandy*, 10(8):707–710.

Louis, S. J. (1993). *Genetic Algorithms as a Viable Computational Tool for Design*. PhD thesis, Department of Computer Science, Indiana University. Bloomington, IN.

Louis, S. J. and Johnson, J. (1997). Solving similar problems using genetic algorithms and case-based memory. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 283–290. Morgan Kauffman, San Mateo, CA.

Louis, S. J., McGraw, G., and Wyckoff, R. (1993). Case-based reasoning assisted explanation of genetic algorithm results. *Journal of Experimental and Theoretical Artificial Intelligence*, 5:21–37.

Louis, S. J., McGraw, G., and Wyckoff, R. (April, 1992). Automating explanation of genetic algorithm results (two paradigms collide). In *Proceedings of the Fifth Florida Artificial Intelligence Research Symposium*, pages 201–205. FLAIRS, FL.

Mostow, J., Barley, M., and Weinrich, T. (1992). Automated reuse of design plans in bogart. In Tong, C. and Sriram, D., editors, *Artificial Intelligence in Engineering Design, Vol II*, pages 57–104. Academic Press, Inc.

Ramsey, C. and Grefensttete, J. (1993). Case-based initialization of genetic algorithms. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 84–91, San Mateo, California. Morgan Kauffman.

Reinelt, G. (1996). Tsplib.

Riesbeck, C. K. and Schank, R. C. (1989). *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, Cambridge, MA.

Schoenauer, M. and Xanthakis, S. (1993). Constrained ga optimization. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 573–580. Morgan Kauffman, San Mateo, CA.

Sheppard, J. W. and Salzburg, S. L. (1995). Combining genetic algorithms with memory based reasoning. In Forrest, S., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 452–459, San Mateo, California. Morgan Kauffman.

Smith, D. (1985). Bin packing with adaptive search. In *Proceedings of an International Conference on Genetic Algorithms*, pages 202–206. Morgan Kauffman.

Sycara, K. and Navinchandra, D. (1992). Retrieval strategies in case-based design system. In Tong, C. and Sriram, D., editors, *Artificial Intelligence in Engineering Design, Vol II*, pages 145–164. Academic Press, Inc.

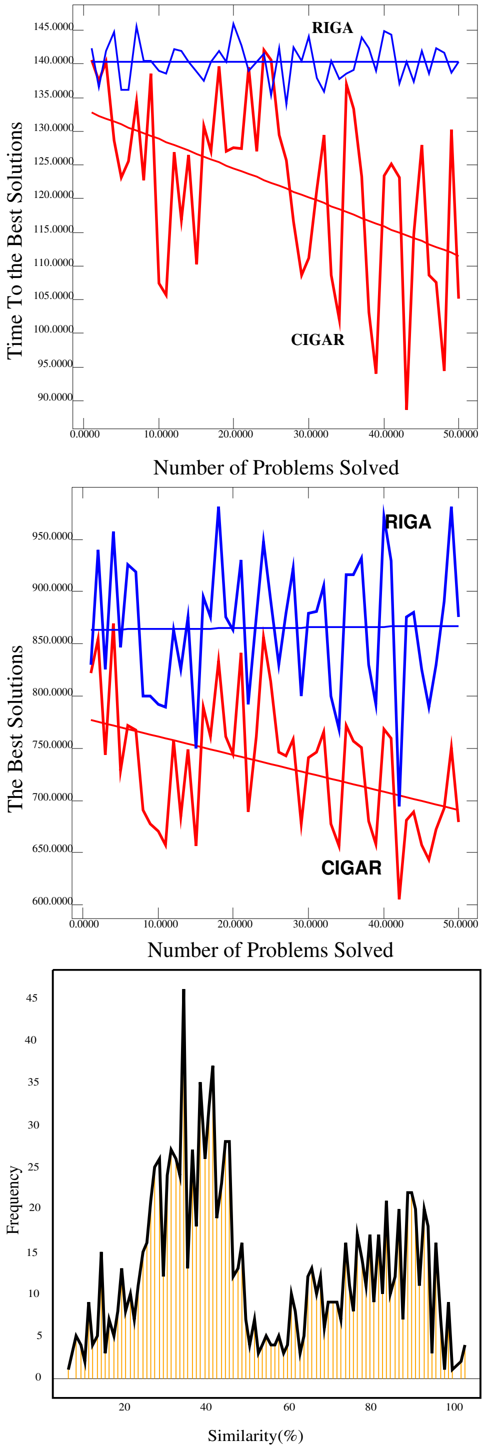

Figure 7: CIGAR vs. RIGA (p_0 = 76 cities, σ = 6). Top: Tour length. Middle: Convergence time. Bottom: Solution similarity distribution

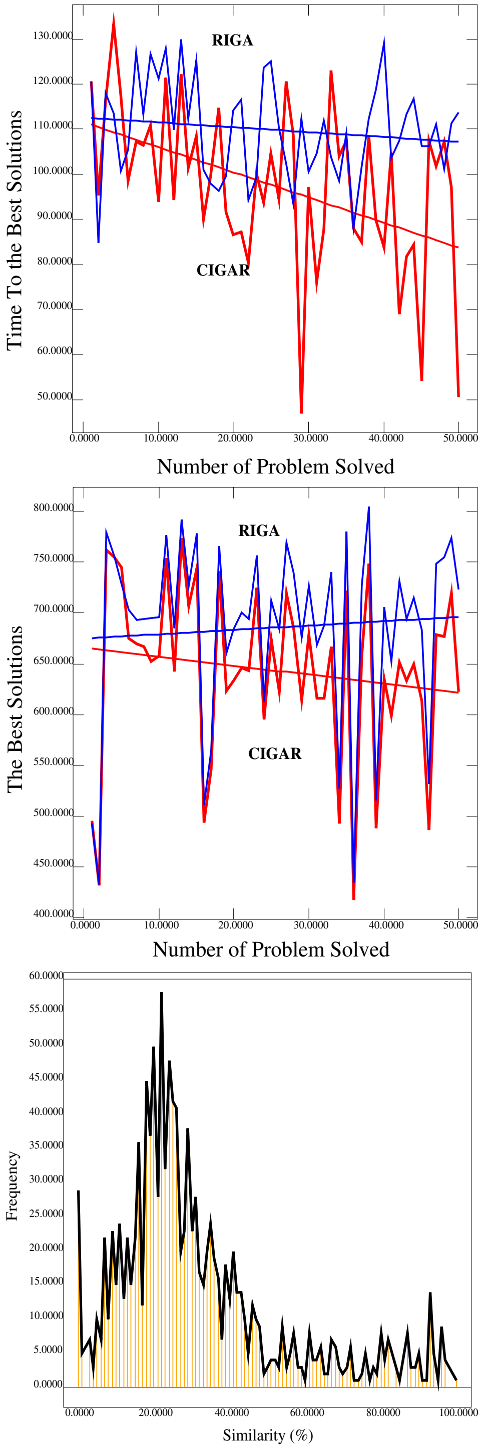

Figure 8: CIGAR vs. RIGA (p_0 = 76 cities, σ = 14). Top: Tour length. Middle: Convergence time. Bottom: Solution similarity distribution