Modeling GA Performance for Control Parameter Optimization

Vincent A. Cicirello
Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
vincent+@cs.cmu.edu phone: 412-268-1416

Stephen F. Smith
Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
sfs@cs.cmu.edu phone: 412-268-8811

Abstract

Optimization of the control parameters of genetic algorithms is often a time consuming and tedious task. In this work we take the meta-level genetic algorithm approach to control parameter optimization. We enhance this process by incorporating a neural network for fitness evaluation. This neural network is trained to learn the complex interactions of the genetic algorithm control parameters and is used to predict the performance of the genetic algorithm relative to values of these control parameters. To validate our approach we describe a genetic algorithm for the largest common subgraph problem that we develop using this neural network enhanced meta-level genetic algorithm. The resulting genetic algorithm significantly outperforms a hand-tuned variant and is shown to be competitive with a hill-climbing algorithm used in practical applications.

1 Introduction

Genetic algorithms use a number of parameters to control their evolutionary search for the solution to their given problems. Some of these include rate of crossover, rate of mutation, maximum number of generations, number of individuals in the population, and so forth. There are no hard and fast rules for choosing appropriate values for these parameters. An optimal or near-optimal set of control parameters for one genetic algorithm or genetic algorithm application does not generalize to all cases. Choosing values for the control parameters is often handled as a problem of trial and error. It is common practice to hand optimize the control parameters by tuning each one at a time. This can be a very time consuming and tedious task. And furthermore, this practice of tuning the control parameters one at a time is not likely to result in the optimal parameter set as the parameters are not independent and often interact in complex ways.

The problem of finding optimal control parameters for genetic algorithms has been studied by many (De Jong 1975; 1980; Grefenstette 1986; Schaffer *et al.* 1989; Bramlette 1991; Wu & Chow 1995; Eiben, Hinterding, & Michalewicz 1999). One approach that has shown promise in several contexts is to use the genetic algorithm itself to search for good control parameter settings – a so-called *meta-level* genetic algorithm approach. However, one drawback with this approach is its computational requirements. The process of executing the primary genetic algorithm to evaluate the fitness of a given set of control parameters can be an expensive operation, particularly if the given set of control parameters results in high convergence times.

In this paper, we describe an approach to improving the efficiency of the meta-level genetic algorithm approach to control parameter optimization. Specifically, we incorporate a neural network as an alternative basis for fitness evaluation at the meta-level. This neural network is trained to learn the effects of the complex interactions of the control parameters on both the accuracy of the genetic algorithm as well as its computational time. This neural network is then used by the meta-level genetic algorithm to predict the performance of the genetic algorithm for which optimal control parameters are sought. Thus, fitness evaluation becomes a far less expensive operation.

The motivation of this work is the development of a genetic algorithm for the largest common subgraph problem. We use this primary genetic algorithm as the context for describing and evaluating our approach to evolving control parameter values. We begin by describing a genetic algorithm for the largest common subgraph problem along with our attempt at hand-

tuning the control parameters. A neural network then learns a model of this genetic algorithm's performance relative to control parameter values. Optimal control parameters are then evolved with the meta-level genetic algorithm using this "surrogate" neural network model for fitness evaluation. The resulting genetic algorithm for the largest common subgraph problem is shown to be competitive with, and in some circumstances superior to, a hill-climbing algorithm used in practical applications.

Section 2 describes the genetic algorithm developed for the largest common subgraph problem. In Section 3 we describe the techniques employed to optimize the control parameters. Experimental results of the meta-level optimization process as well as results comparing the performance of the optimized genetic algorithm to a hill-climbing algorithm are presented in Section 4. Related work is discussed in Section 5. And finally Section 6 discusses conclusions.

2 Largest Common Subgraph Genetic Algorithm

The largest common subgraph problem and other closely related problems such as maximum subgraph matching, graph isomorphism, subgraph isomorphism, and error-correcting graph isomorphism are important in many applications. Some of these applications include chemical structure classification, sub-circuit identification (Ohlrich *et al.* 1993), VLSI (Kodandapani & McGrath 1986), CAD model comparison (Elinson, Nau, & Regli 1997; Cicirello & Regli 1999; Cicirello 1999), pattern recognition (Cho & Kim 1992; Lu, Ren, & Suen 1991; Pearce, Caelli, & Bischof 1994), machine vision (Wong 1992; Christmas, Kittler, & Petrou 1995), and case-based reasoning (Andersen *et al.* 1994; Sanders, Kettler, & Hendler 1997).

The largest common subgraph problem seeks a subgraph from each of a pair of graphs such that these subgraphs are isomorphic and such that this common subgraph has the largest number of edges of all possible common subgraphs. This problem is known to be in the class of NP-complete problems and is computationally intractable (Garey & Johnson 1979). Due to its utility in a wide array of application areas, an efficient approximation to the largest common subgraph problem would be beneficial.

Shoukry and Aboutabl (Shoukry & Aboutabl 1996) describe an inexact solution to the largest common subgraph problem using a two-stage Hopfield neural network. They show that their algorithm is easily parallelized and compare its performance to a similar algorithm. Another inexact solution to the largest common subgraph problem is the hill-climbing approach of Cicirello and Regli (Cicirello & Regli 1999; Cicirello 1999). They take an iterative improvement approach and restart the algorithm at random starting points as an attempt at combating the problem of local extrema. Algorithms for the closely related problem of error-correcting graph isomorphism include those of Messmer and Bunke (Messmer & Bunke 1996), Almohamad and Duffuaa (Almohamad & Duffuaa 1993), and Wang et al. (Wang, Fan, & Horng 1997).

In this work we develop a genetic algorithm approach to the largest common subgraph problem. Chromosomes represent node permutations with individual alleles representing correspondences between nodes. Indices into the chromosome represent the nodes of the graph with the smaller number of nodes. The values at the individual locations within the chromosome represent nodes from the second graph. The complete chromosome represents a mapping from the nodes of the first graph represented by the indices to that of the second graph represented by the values.

The fitness of a chromosome is the number of matched edges in the mapping represented by the given chromosome. Roulette wheel selection is used with each chromosome in the population taking a portion of the roulette wheel based on the value of its fitness evaluation. An elitist strategy is also incorporated in which the best two chromosomes in the population are carried over to the next generation. These two elite individuals do not undergo mutation or crossover.

"Swap" mutation is used in which the value of each allele is swapped with some other allele with some small probability. If the graphs are of differing numbers of nodes there are two choices of mutation operator: 1) swap two alleles or 2) swap the value of one allele with a node not currently in the chromosome.

A crossover operator that is loosely based on partially matched crossover PMX is used. Wang et al. (Wang, Fan, & Horng 1997) make use of PMX as described in Goldberg (Goldberg 1989) within their genetic algorithm for the closely related problem of error-correcting graph isomorphism. In the present work we have adopted a uniform variation of PMX. Instead of choosing a segment of the chromosomes to exchange, each allele is exchanged with some probability. Consider an example.

- Consider chromosomes { 5, 2, 3, 4, 1, 6 } and { 1, 4, 6, 2, 5, 3 }
- Consider crossing over at points 1 and 3 and swap

the values at these positions between chromosomes to result in { 1, 2, 6, 4, 1, 6 } and { 5, 4, 3, 2, 5, 3 }

- As a result it is necessary to crossover at points 5 and 6, also by swapping the values at these positions between chromosomes, to obtain { 1, 2, 6, 4, 5, 3 } and { 5, 4, 3, 2, 1, 6 }

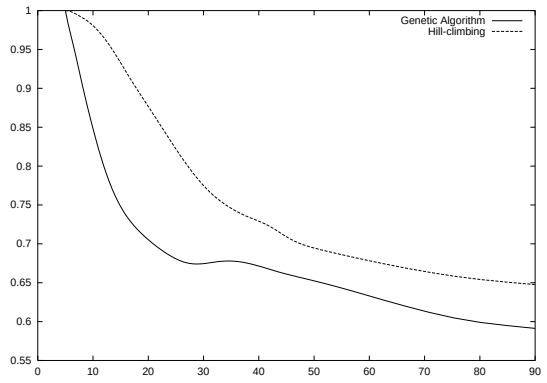
Finally, the following halting criteria are used: 1) the fitness of some individual is equal to the number of edges in one of the graphs (in this case the graphs are actually subgraph isomorphic and that subgraph isomorphism has been found); 2) the fitness of all individuals in the population are equal; 3) the fitness of the least fit chromosome is within some small tolerance of the fitness of the most fit individual; 4) some maximum number of generations have been evolved; or 5) some number of generations evolve with no improvement on the fitness of the most fit individual.

3 Control Parameter Optimization

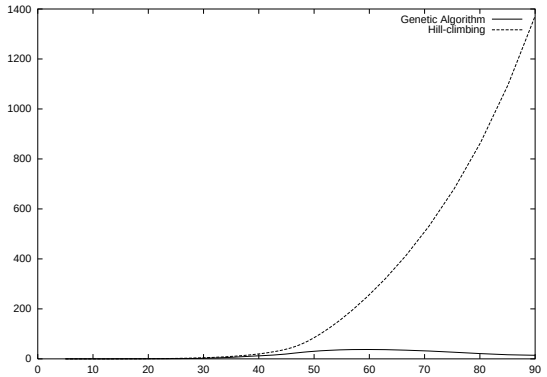
The genetic algorithm that is described in Section 2 can be defined by the control parameter set $\Pi = \{P, C, U, M, T, S\}$, where:

- P is the size of the population.
- C is the crossover rate.
- U is the probability any given allele is involved in crossover.
- M is the mutation rate.
- T is the halting tolerance described in Section 2. That is, the genetic algorithm halts if $\frac{Fitness(MostFit) - Fitness(LeastFit)}{Fitness(MostFit)} < T$.
- S controls which mutation operator is used in the case of graphs of differing numbers of nodes. That is, when mutation occurs the operator that swaps the values of two alleles is chosen with probability S and the operator that swaps the value of an allele with a node not in the current mapping is chosen with probability $1 - S$.

Our first attempt at the optimization of these control parameters was one of hand-tuning. We began with an initial set of control parameter settings taken from standard practice and gradually perturbed one parameter at a time keeping the result provided it improved the average performance of the genetic algorithm. The



(a)



(b)

Figure 1: Plots comparing the genetic algorithm with hand-tuned control parameters to a hill-climbing algorithm: (a) Accuracy vs graph size; (b) CPU time in seconds vs graph size.

result of this hand-optimization was the control parameter set $D = \{P = 50, C = 0.75, U = 0.50, M = 0.001, S = 0.50, T = 0.01\}$. Figure 1 compares the genetic algorithm for the largest common subgraph problem using these hand-tuned control parameters to the hill-climbing approach of (Cicirello & Regli 1999; Cicirello 1999). Using these hand-tuned parameters, the genetic algorithm is less accurate than the hill-climbing algorithm. However, it requires far less CPU time to convergence. Perhaps it is possible to trade off more CPU time for added accuracy in the solutions through the use of a more sophisticated control parameter optimization strategy.

In this section, we formulate our approach to optimizing the control parameters of the genetic algorithm for the largest common subgraph problem. A meta-level genetic algorithm approach is taken. A neural network is used to generate predictions of how well the genetic algorithm will perform given a set of control parame-

ters. This neural network prediction is used for fitness evaluation by the meta-level genetic algorithm.

3.1 Neural Network Prediction of Performance

The fitness function of the meta-level genetic algorithm uses a neural network prediction of the performance of the primary genetic algorithm given a set of control parameters. The idea is that since it would be a computationally expensive operation for the meta-level genetic algorithm to actually execute the primary genetic algorithm for each control parameter set that it examines, a neural network can perhaps learn the complex interactions of the control parameters.

The neural network we develop has six input units. These six input units are the values of the control parameters $\Pi = \{P, C, U, M, T, S\}$. There is a hidden layer of six sigmoid units in the network and four sigmoid output units. These four output units encode the value $Fitness = round(15\frac{9.0t_1+t_2}{10})$. t_1 is the average accuracy of the genetic algorithm using the given control parameters after m runs of the algorithm. Accuracy is defined as $\frac{|E_s|}{|E|}$ where E_s is the edge set of the resulting approximation to the largest common subgraph and E is the edge set of the actual largest common subgraph. t_2 is defined as the ratio $1 - \frac{C}{C_m}$ where C is the average CPU time to convergence using the given control parameters and C_m is the maximum CPU time to convergence of any control parameter set in the neural network training set. These output units are treated as a binary representation of the value of $Fitness$.

Training data for the neural network is gathered by first generating a large number of control parameter sets at random. Then the genetic algorithm is executed several times using each of these random control parameter sets and the "true" value for $Fitness$ is finally calculated.

A collection of isomorphic pairs of random graphs ranging in size from 5 to 20 nodes were generated. This collection was used to test the performance of sets of control parameters. A total of 1250 control parameter sets were generated randomly to be used as training data. These control parameter sets were distributed across the value ranges defined by the meta-level genetic algorithm chromosome representation (see Section 3.2). $Fitness$ for each of these 1250 control parameter sets were then calculated in batch by executing the largest common subgraph genetic algorithm on each pair of graphs in the randomly generated collection.

The data was then divided into a training set of 1000 control parameter sets and a hold-out set of 250 control parameter sets. Back-propagation was used to train the neural network with a learning rate of 0.5 and momentum of 0.5 for approximately 4000 epochs. Accuracy of 69.8% was obtained over the training set and 62.0% accuracy was obtained over the hold-out set where we define accuracy as the percentage of parameter sets for which the neural network correctly predicts the performance of the genetic algorithm. A correct prediction means that the fitness value predicted by the neural network is within 0.03 of the actual fitness value. The hold-out set was used to prevent overfitting of the training data. That is, the hold-out set was not used as training data but the neural network that resulted in the best performance over the hold-out set was maintained during this training phase.

3.2 Meta-level Genetic Algorithm

For the meta-level genetic algorithm, we use a two-point crossover operator with a crossover rate of 0.95, a mutation rate of 0.0025, and a population size of 100. Roulette wheel selection based on the fitness of the individuals of the population is used but with an elitist strategy ensuring that the two most fit individuals of the population survive in tact to the next generation. The only stopping criterion is that the entire population is of the same fitness. The meta-level genetic algorithm uses the neural network developed in Section 3.1 to evaluate the fitness of individuals in the population.

Individuals are encoded as binary strings with each control parameter encoded by some number of bits. The population size is encoded using 4 bits. Each of these 16 substrings correspond to a population size ranging from 10 to 160 in increments of 10. Crossover rate is encoded with the next 4 bits representing values between 0.25 and 1.0 in increments of 0.05. The uniform parameter for the uniform crossover operator and the mutation operator selector are each encoded with 4 bits representing values between 0.15 and 0.9 in increments of 0.05. The mutation rate is encoded with 4 bits representing the 16 values 2^{-i} for $i = 0, \ldots, 15$. Fitness tolerance is encoded in the same manner as is the mutation rate. The complete individual is 24 bits in length allowing for 2^{24} unique individuals.

Table 1: Sample of control parameter sets evolved by the meta-level genetic algorithm.

	P	C	U	M	S	T
A	60	0.55	0.70	0.03125	0.50	0.003906
B	20	0.55	0.85	0.0625	0.50	0.000061
C	100	0.95	0.50	0.0625	0.80	0.000031

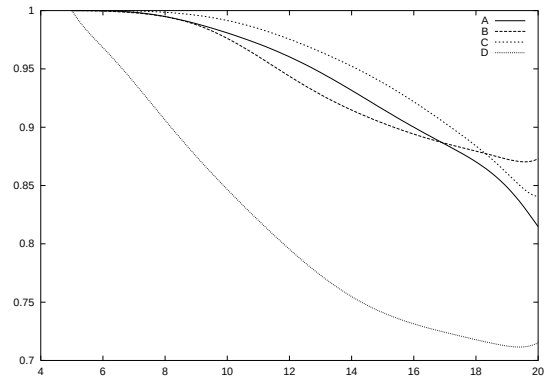
4 Experimental Results

4.1 Control Parameter Experiments

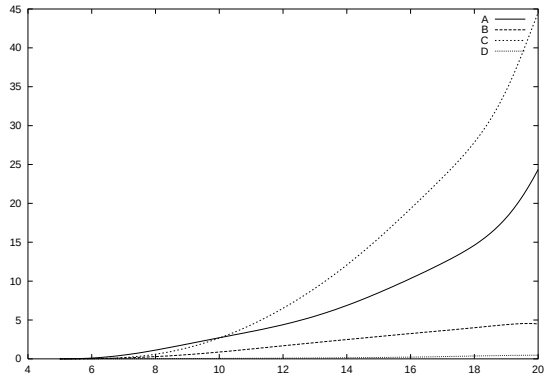
Using the meta-level genetic algorithm described in Section 3.2 with the neural network as described in Section 3.1 for fitness evaluation, a population of 100 sets of control parameters were evolved. A small sample of this final population can be seen in Table 1. The thing that stands out the most in the final population of parameter sets is that all individuals in this population state that the optimal mutation rate is either 0.03125 or 0.0625. This essentially signifies convergence for the value of the mutation rate in that these two values are consecutive given the encoding scheme used by the meta-level genetic algorithm. Looking over these lists with the human eye reveals no obvious relationship between the values of the parameters with respect to performance of the genetic algorithm. The final population evolved in 67 generations of the meta-level genetic algorithm.

During the entire run of the meta-level genetic algorithm, 6700 not necessarily unique parameter sets were examined (population of 100 and 67 generations). Factoring in elitism results in 6566 not necessarily unique parameter sets. With the crossover rate of 0.95, approximately 95% of each population is the result of crossover leaving 6238 possibly unique control parameter sets. A number of these sets are likely to be duplicates leaving a very rough estimate of 5000-6000 unique parameter sets examined by the meta-level genetic algorithm. Recall that the neural network was trained with only 1000 training examples. Training a neural network to learn how the control parameters interact required only executing the largest common subgraph genetic algorithm with 1000 control parameter sets rather than the 5000 that would have been required without using the neural network.

Let us now take a look at just how well the evolved control parameters perform. First note that none of the control parameter sets in Table 1 were in the neural network training set. These three were chosen at random from the final population. An experiment was conducted to compare their performance to that of a control parameter set that was derived through hand-tuning.



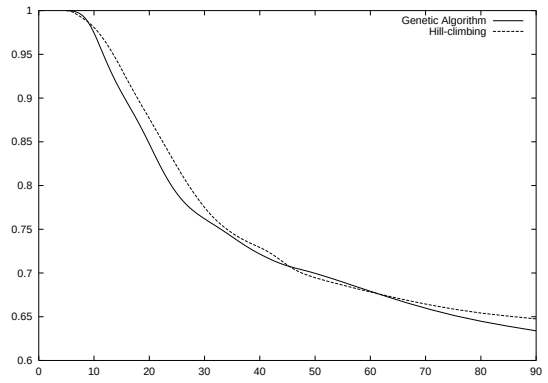
(a)



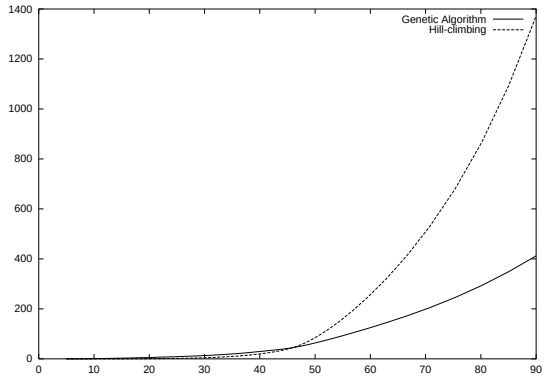
(b)

Figure 2: Plots comparing the performance of the control parameter sets in Table 1 with that of the hand-tuned set, $D = (P = 50, C = 0.75, U = 0.50, M = 0.001, S = 0.50, T = 0.01)$: (a) plot of accuracy vs graph size; (b) plot of CPU time in seconds vs graph size.

As can be seen in Figure 2 (a), all three of these evolved control parameter sets perform better than the hand-tuned parameter set in terms of accuracy. It is difficult to judge which of these three is best in terms of accuracy. But choosing one based only on accuracy is not necessary. Figure 2 (b) shows a plot of the average CPU time required by the largest common subgraph genetic algorithm using each of these four parameter sets. Control parameter set B is clearly the winner among sets A, B, and C. It is not, however, faster than the hand-tuned set. But taking both accuracy



(a)



(b)

Figure 3: Plots comparing the genetic algorithm approach to a hill-climbing algorithm. The control parameters obtained through means of meta-level optimization are used: (a) Accuracy vs graph size; (b) CPU time in seconds vs graph size.

and CPU time into consideration, there is a clear argument in favor of control parameter set B as the best of these four.

4.2 Comparison with Hill-Climbing

Here we experimentally compare the genetic algorithm described in this work using the control parameter settings derived in Section 4.1 to the hill-climbing algorithm described in (Cicirello & Regli 1999; Cicirello 1999). The results are shown in Figure 3. The algorithms were executed on randomly generated isomorphic pairs of graphs from size 5 nodes to 90 nodes. Isomorphic pairs were chosen for the experiment to allow for some basis of computing the accuracy of the result. The experiments were performed on a Pentium III using the Linux operating system and C++ for implementation.

The genetic algorithm and the hill-climbing algorithm compare similarly in terms of accuracy. However, the more interesting result is that of the CPU time comparison. For graphs with 45 nodes or less the hill-climbing algorithm requires less CPU time then does the genetic algorithm. But for graphs with more than 45 nodes, the genetic algorithm developed using our meta-level optimization approach far out-performs the hill-climbing algorithm.

5 Related Work

One of the earliest studies of genetic algorithm control parameters is that of De Jong (De Jong 1975). He analyzed a class of genetic algorithms for function optimization. De Jong's optimal control parameters became widely used despite lack of knowledge of optimality with respect to problems outside of his test collection. Schaffer et al. (Schaffer *et al.* 1989) expanded De Jong's test suite and performed a more systematic study of the effects of the control parameters.

Grefenstette (Grefenstette 1986) saw the problem of tuning the control parameters of the primary genetic algorithm as a secondary or meta-level optimization problem and was one of the first to apply a meta-level genetic algorithm approach. Grefenstette's meta-level genetic algorithm was parameterized with De Jong's optimal control parameters: population of 50, cross-over rate of 0.6, mutation rate of 0.001, generation gap of 1.0, scaling window of 7, and an elitist strategy (De Jong 1975). Grefenstette's results showed a slight improvement over De Jong's with control parameters: population size of 30, cross-over rate of 0.95, mutation rate of 0.01, and an elitist strategy (Grefenstette 1986).

Others have also applied meta-level genetic algorithms to control parameter optimization. Bramlette (Bramlette 1991) presents modified methods of selecting initial populations and mutation operators for improving the performance of genetic algorithms for function optimization. He tests his techniques on a meta-level genetic algorithm to optimize the control parameters for some other genetic algorithm. Wu and Chow (Wu & Chow 1995) apply a meta-level genetic algorithm approach to optimizing the control parameters of genetic algorithms for nonlinear mixed discrete-integer optimization problems. De Jong (De Jong 1980) applies a genetic algorithm approach to control parameter optimization of dynamical systems, not necessarily to control parameters of a genetic algorithm.

6 Conclusions

In this work we have developed a neural network enhanced meta-level genetic algorithm approach to control parameter optimization. For fitness evaluation, the meta-level genetic algorithm makes use of a neural network prediction of the performance of the primary genetic algorithm given a set of control parameters as input. By training a neural network to learn the complex interactions of the control parameters of the genetic algorithm, we are able to provide an efficient alternative to running the actual algorithm for purposes of meta-level optimization.

This approach to meta-level control parameter optimization has allowed us to develop a genetic algorithm for evolving inexact solutions to the largest common subgraph problem efficiently and accurately. The overall approach of using a genetic algorithm for finding approximate solutions to the largest common subgraph problem has been shown to be asymptotically more efficient than hill-climbing. The solutions produced by this genetic algorithm are also just as accurate as the solutions of the hill-climbing algorithm. The results of the experiments presented in this work suggest that for relatively small graphs of roughly 40 nodes or less the hill-climbing algorithm performs best, but for larger graphs the genetic algorithm of this work far out-performs the hill-climbing algorithm.

Further experimental evidence is necessary to fully understand the benefits of this evolutionary approach to the largest common subgraph problem. For example, it would be beneficial to conduct experiments examining the performance of the genetic algorithm for other types of random graphs such as varying the density of edges of these graphs as well as labeling the nodes.

More broadly, the approach we have described to optimization of search control parameters appears quite general. It may also be beneficial to apply the techniques of this work to other control parameter optimization problems. For example, neural network models of the performance of such algorithms as simulated annealing and tabu search relative to their respective control parameters may be built and used for the purpose of control parameter optimization.

References

Almohamad, H. A., and Duffuaa, S. O. 1993. A linear programming approach for the weighted graph matching problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15(5):522–525.

Andersen, W. A.; Hendler, J. A.; Evett, M. P.; and Kettler, B. P. 1994. Massively parallel matching of knowledge structures. In Kitano, H., and Hendler, J., eds., *Massively Parallel Artificial Intelligence*. Menlo Park, California: AAAI Press/The MIT Press. 52–73.

Bramlette, M. F. 1991. Initialization, mutation and selection methods in genetic algorithms for function optimization. In Belew, R. K., and Booker, L. B., eds., *Proceedings of the Fourth International Conference on Genetic Algorithms*, 100–107. San Mateo, CA: Morgan Kaufmann.

Cho, C. J., and Kim, J. J. 1992. Recognizing 3-D objects by forward checking constrained tree search. *Pattern Recognition Letters* 13(8):587–597.

Christmas, W. J.; Kittler, J.; and Petrou, M. 1995. Structural matching in computer vision using probabilistic relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17(8):749–764.

Cicirello, V. A., and Regli, W. C. 1999. Resolving non-uniqueness in design feature histories. In Bronsvoort, W. F., and Anderson, D. C., eds., *Proceedings of the Fifth Symposium on Solid Modeling and Applications*, 76–84. New York: ACM SIGGRAPH.

Cicirello, V. A. 1999. Intelligent retrieval of solid models. Master's thesis, Drexel University, Philadelphia, PA.

De Jong, K. A. 1975. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Ph.D. Dissertation, University of Michigan, Ann Arbor, MI.

De Jong, K. 1980. Adaptive system design: A genetic approach. *IEEE Transactions on Systems, Man, and Cybernetics* 10(9):566–574.

Eiben, A. E.; Hinterding, R.; and Michalewicz, Z. 1999. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* 3(2):124–141.

Elinson, A.; Nau, D. S.; and Regli, W. C. 1997. Feature-based similarity assessment of solid models. In Hoffman, C., and Bronsvoort, W., eds., *Fourth Symposium on Solid Modeling and Applications*, 297–310. New York: ACM.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman and Company.

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

Grefenstette, J. 1986. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics* 16(1):122–128.

Kodandapani, K. L., and McGrath, E. J. 1986. A wirelist compare program for verifying VLSI layouts. *IEEE Design and Test* 3(3):46–51.

Lu, S. W.; Ren, Y.; and Suen, C. Y. 1991. Hierarchical attributed graph representation and recognition of handwritten Chinese characters. *Pattern Recognition* 24:617–632.

Messmer, B., and Bunke, H. 1996. Fast error-correcting graph isomorphism based on model precompilation. Technischer Bericht IAM-96-012, Institut für Informatik, Universität Bern, Schweiz.

Ohlrich, M.; Ebeling, C.; Ginting, E.; and Sather, L. 1993. Subgemini: Identifying subcircuits using a fast subgraph isomorphism algorithm. In *Proceedings of the 30th International Design Automation Conference*, 31–37.

Pearce, A.; Caelli, T.; and Bischof, W. F. 1994. Rulegraphs for graph matching in pattern recognition. *Pattern Recognition* 27(9):1231–1246.

Sanders, K. E.; Kettler, B. P.; and Hendler, J. A. 1997. The case for graph-structured representations. In *Proceedings of the Second International Conference on Case-based Reasoning (ICCBR)*. Berlin-Heidelberg-New York: Springer-Verlag.

Schaffer, J. D.; Caruana, R. A.; Eshelman, L. J.; and Das, R. 1989. A study of control parameters affecting online performance of genetic algorithms for function optimization. In Schaffer, J. D., ed., *Proceedings of the Third International Conference on Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann.

Shoukry, A., and Aboutabl, M. 1996. Neural network approach for solving the maximal common subgraph problem. *IEEE Transactions on Systems, Man, and Cybernetics: Part B - Cybernetics* 26(5):785–790.

Wang, Y. K.; Fan, K. C.; and Horng, J. T. 1997. Genetic-based search for error-correcting graph isomorphism. *IEEE Transactions on Systems, Man, and Cybernetics: Part B - Cybernetics* 27(4):588–597.

Wong, E. K. 1992. Model matching in robot vision by subgraph isomorphism. *Pattern Recognition* 25(3):287–304.

Wu, S. J., and Chow, P. T. 1995. Genetic algorithms for nonlinear mixed discrete-integer optimization problems via meta-genetic parameter optimization. *Engineering Optimization* 24(2):137–159.