Constraint Satisfaction Using A Hybrid Evolutionary Hill-Climbing Algorithm That Performs Opportunistic Arc and Path Revision

James Bowen
Department of Computer Science
National University of Ireland at Cork
Cork, Ireland
J.Bowen@cs.ucc.ie

Gerry Dozier
NASA Center for Autonomous Control Engineering
Department of Computer Science
North Carolina A&T State University
Greensboro, NC 27411, USA
gvdozier@ncat.edu

Abstract

This paper introduces a hybrid evolutionary hill-climbing algorithm that quickly solves Constraint Satisfaction Problems (CSPs). This hybrid uses opportunistic arc and path revision in an interleaved fashion to reduce the size of the search space and to realize when to quit if a CSP is based on an inconsistent constraint network. This hybrid outperforms a well known hill-climbing algorithm, the Iterative Descent Method, on a test suite of 750 randomly generated CSPs.

Introduction

Evolutionary Algorithms (EAs) are powerful, robust search procedures that are capable of finding optimal or near optimal solutions to search and optimization problems very quickly. Over the years, many researchers have applied EAs successfully to a variety of areas such as machine learning, neural networks, and scheduling (Michalewicz 1992). One of the more rapidly growing areas in which EAs are being successfully applied is Constraint Processing (Crawford 1992, Dozier,Bowen & Bahler 1994a, Eiben, Raue & Ruttkay 1994, Homaifar 1992).

This paper introduces a new hybrid evolutionary hill-climbing algorithm that uses information collected during its search opportunistically to reduce the size of the search space and to remove tuples from constraints by applying the concepts of Arc and Path Revision (Mackworth 1977). In addition to improving the efficiency of subsequent evolutionary hill-climbing search, using opportunistic arc and path revision also has the effect that eventually the EA recognizes that a CSP is based on a network which admits no consistent assignment because the network is inconsistent (Mackworth 1977).

The outline of this paper is as follows. In the next section, we provide an overview of Constraint Processing. Then we briefly introduce the concepts of Arc Revision and Path Revision. Our new algorithm, called AGSAP, is built by adding the concept of Path Revision to an earlier EA. This earlier algorithm, which was called AGSA (Bowen & Dozier 1995), was, in turn, built by adding some concepts from hill-climbing and opportunistic Arc Revision to the basic notion of evolutionary search. Thus, to set the scene for presenting AGSAP, we review the salient hill-climbing concepts and briefly present the earlier algorithm, AGSA. Next, we present the new algorithm in detail and describe our test suite of 750 randomly generated constraint networks. We used this test suite to compare a well-known hill-climbing algorithm, IDM (Morris 1993), with AGSA and with the new AGSAP algorithm. Finally, we discuss our results, conclusions and our ongoing research.

An Overview of Constraint Processing

A constraint network (Bowen, Lai & Bahler 1992) is a triple $\langle D, X, C \rangle$ where D is a non-empty, finite tuple of q domains, X is a tuple of q distinct objects, each X_i of which takes its value from the corresponding domain D_i, and C is a non-empty, finite set of r constraints, $C_1(T_1), \ldots, C_r(T_r)$, with the following characteristics:

1. $C_k(T_k)$ restricts the values that may be assumed by the a_k members of T_k, a sub-tuple of X of the form $\langle X_{k,1}, \ldots, X_{k,a_k} \rangle$;
2. $C_k(X_{k,1}, \ldots, X_{k,a_k})$ is a subset of $D_{k,1} \times \ldots \times D_{k,a_k}$.

This definition admits the possibility that domains and constraints may be either finite or infinite sets and may be specified either extensionally or intensionally. The overall network constitutes an intensional specification of a q-ary relation, a subset of $D_1 \times \ldots \times D_q$; in what follows we denote this relation as $\Pi_{D,X,C}$.

We can define three forms of Constraint Satisfaction Problem (CSP) based on constraint networks:

1. The Decision CSP: Given a network $\langle D, X, C \rangle$, decide whether $\Pi_{D,X,C}$ is non-empty.
2. The Exemplification CSP: Given a network $\langle D, X, C \rangle$, return some tuple from $\Pi_{D,X,C}$, if $\Pi_{D,X,C}$ is non-empty; return nil otherwise
3. The Enumeration CSP: Given a network $\langle D, X, C \rangle$, return $\Pi_{D,X,C}$.

The rest of this paper will focus on Exemplification CSPs whose networks contain only binary constraints. A binary constraint $C_k(X_{k,1}, X_{k,2})$ restricts the values that can be assigned to $X_{k,1}$ and $X_{k,2}$. The **Tightness** of $C_k(X_{k,1}, X_{k,2})$ is the ratio of the number of

tuples disallowed by the constraint $C_k(X_{k,1}, X_{k,2})$ to the number of tuples in $D_{k,1} \times D_{k,2}$. The **Average Tightness** of a binary constraint network is the sum of the tightness of each constraint divided by the number of constraints in the network (Benson & Freuder 1992).

The **Density** of a constraint network is the ratio of the number of constraints in the network to the total number of constraints possible (Benson & Freuder 1992). The maximum number of constraints possible in a binary constraint network containing q objects is $(q^2 - q)/2$; if the network has r constraints, the density is $2r/(q^2 - q)$.

Arc Revision

A binary constraint network can be viewed as a graph with the objects treated as nodes. A constraint $C_j(A, B)$ between two objects A and B can be regarded as two directed arcs, $\vec{AB}$ and $\vec{BA}$. A value in the domain of the object B at the head of an arc $\vec{AB}$ is said to *support* through the arc a value in the domain of the object A at the tail if the directed pair containing these values is present in the relation corresponding to the arc.

Arc revision (Mackworth 1977) is a method of constraint processing which removes from the domain of the object at the tail of an arc any value which is not supported through the arc by at least one value in the domain of the object at the head of the arc. Arc revision is an important constraint processing technique because, by reducing the domains of objects in a CSP, it reduces the size of the search space. It can also detect when some networks admit no consistent assignment, which happens when arc revision leads to the removal of all members from some object's domain. When this occurs, the constraint network is said to be **arc inconsistent**.

Path Revision

Consider a network containing three objects, F, G and H, with domains $D_F = \{f_1, f_2, f_3\}$, $D_G = \{g_1, g_2, g_3\}$ and $D_H = \{h_1, h_2, h_3\}$, respectively. Suppose that
$C_l(F, G) = \{\langle f_1, g_1\rangle, \langle f_2, g_2\rangle, \langle f_3, g_3\rangle\}$
$C_m(F, H) = \{\langle f_1, h_1\rangle, \langle f_2, h_2\rangle, \langle f_3, h_3\rangle\}$
$C_n(G, H) = \{\langle g_1, h_2\rangle, \langle g_2, h_1\rangle, \langle g_3, h_3\rangle\}$
Arc revision cannot remove any value from any domain in this situation. Note, however, that, although $\langle f_1, g_1\rangle \in C_l(F, G)$, there is no value $h \in D_H$ such that $\langle f_1, h\rangle \in C_m(F, H) \wedge \langle g_1, h\rangle \in C_n(G, H)$. Similarly, although $\langle f_2, g_2\rangle \in C_l(F, G)$, there is no value $h \in D_H$ such that $\langle f_2, h\rangle \in C_m(F, H) \wedge \langle g_2, h\rangle \in C_n(G, H)$. Thus, the intent of the network would not be altered by removing the pairs $\langle f_1, g_1\rangle$ and $\langle f_2, g_2\rangle$ from $C_l(F, G)$. Modifying a constraint relation in this fashion is the basic operation of path revision used in the path consistency algorithms (Mackworth 1977). Path consistency algorithms, which must use the path revision operation many times, are computationally expensive but we can reduce the search space in a less expensive manner if we use the path revision operation sparingly and interleave its application with opportunistic applications of arc revision (Bowen, Lai & Bahler 1992). For example, in the above example, the operation of replacing $C_l(F, G)$ by $C'_l(F, G) = \{\langle f_3, g_3\rangle\}$ can be followed by applications of arc revision which lead to replacing D_F, D_G and D_H by $D'_F = \{f_3\}$, $D'_G = \{g_3\}$ and $D'_H = \{h_3\}$, respectively.

Solving CSPs Using Hill-Climbing

The Min Conflicts Hill Climbing algorithm (MCHC) (Minton, 1992) and the Iterative Descent Method (IDM) (Morris 1993) are two algorithms which are very effective in solving exemplification CSPs. Both of these algorithms use a heuristic which guides the search process. This heuristic causes the hillclimbing algorithm to reassign values only to objects that are involved in constraint violations (CV).

Since both of these algorithms are hill-climbers, they tend to get trapped at local optima. MCHC cannot escape local optima; however, IDM does so by using a list of breakouts. A breakout has two parts. The first part is a 2-tuple, known as a nogood, that violates a constraint. The second part of a breakout is its weight.

When trapped at a local optimum, IDM invokes a Breakout Algorithm. This creates a breakout for each nogood of the equilibrium point representing the local optimum that has never appeared in any previously discovered equilibrium point; the weight of each newly created breakout is assigned a value of one. The Breakout Algorithm increments by one the weights of any nogoods in the local optimum that appeared in previously discovered local optima.

A breakout is said to be violated by a candidate solution (CS) if its nogood is present as a sub-tuple within the CS. CSs are evaluated based on the number of CVs they cause plus the sum of the weights of the breakouts they violate.

Evolutionary Algorithms

Evolutionary Algorithms (EAs) are search procedures based on natural selection (Goldberg 1989). Unlike most search algorithms, which operate on a single candidate solution (CS), EAs operate on a population of individuals where each individual represents a CS.

After an initial population of randomly generated individuals has been produced, each individual in the population is assigned a fitness by an evaluation function. Individuals within the population are selected to be parents based on their fitness. The parents create offspring (which are also individuals that represent CSs) by either mutating themselves slightly (asexual procreation), mating with other parents (sexual procreation) or both. The offspring are then evaluated to determine their fitness and are added to the population, usually replacing lesser fit individuals in or-

der to keep the population size constant. This process of selecting parents and allowing them to procreate based on their fitness is continued until an individual representing an optimal or near optimal CS has been evolved.

Solving CSPs Using Hybrid EAs

The results which we report in this paper stem from a research program directed at developing hybrid algorithms for solving CSPs that combine the advantages of evolutionary search with appropriate concepts from other constraint processing techniques. Before developing the algorithm which we report in this paper, a series of algorithms had been developed, (Dozier, Bowen & Bahler 1994a, Dozier, Bowen & Bahler 1994b, Dozier, Bowen & Bahler 1995), by combining evolutionary search with some techniques from the above hill-climbing algorithms. These earlier hybrids used a heuristic to guide their primary evolutionary operator, singlepoint mutation; this heuristic is similar to that mentioned above as being used by MCHC and IDM. In addition, they evolved a small population of six individuals and were able to escape from local optima because they used a Breakout Management Mechanism similar to IDM's Breakout Algorithm.

When applied to a test suite of 750 randomly generated exemplification CSPs, these earlier hybrids outperformed pure evolutionary search, in that they found solutions with less effort – our measure of work is the number of candidate solutions evaluated. Also, on all but the easiest problems they outperformed the IDM hill-climber (we did not compare them with MCHC since MCHC cannot escape local optima) and as the problems got harder the extent to which the hybrid EAs outperformed IDM became more pronounced. Furthermore, even in those problems where IDM performed fewer evaluations, IDM's better performance was only of statistical, rather than practical, significance; this is because these were easy problems where none of the algorithms needed many evaluations.

One of these earlier algorithms, named AGS (A Genetic/Systematic search hybrid), recognized family relationships between candidate solutions and used this recognition to reduce the likelihood of a candidate solution being evaluated more than once, by maintaining a data structure called a Family Table to record information about the evolution history of families that were present in the population at any time; the table never grew large since a family's entry in it was discarded when the last member of the family present in the population was removed to provide niche space for new arrivals.

The possibility of interleaving evolutionary search with opportunistic consistency processing was discovered when it was observed that AGS, while doing its evolutionary search, collects information which can be used in an opportunistic fashion to simplify the CSP at hand by pruning from object domains members which were not supported by neighbouring objects. With this insight, AGSA was developed (Bowen & Dozier 1995) (A Genetic/Systematic Arc revising hybrid). AGSA was very effective in reducing search space size and detecting when some CSPs had no solution. Our new algorithm, AGSAP (A Genetic/Systematic Arc and Path revising hybrid), was prompted by the realization that, if we slightly modified the Family Table used by AGSA, the resultant information could be used to perform path revision opportunistically and that path revisions could be interleaved with arc revisions.

A New Hybrid EA

In AGSAP, a CS is represented as a chromosome in which there is one gene for each object of the constraint network. Associated with each chromosome is a marker called a pivot which tells the heuristic-based mutation operator which gene to mutate. (A full discussion of this representation is beyond the scope of this paper. For further detail see (Dozier, Bowen & Bahler 1994a).)

The evaluation function determines the objective fitness of a CS by subtracting the weights of all breakouts violated by the CS from the number of constraints satisfied by the CS.

The population of CSs are ranked by objective fitness from 1 to P, where P is the population size. When selecting CSs from the population to serve as parents, AGSAP uses linear rank selection (Goldberg 1989), the subjective fitness sf of a CS whose rank within the population is ρ being determined by $sf(\rho) = P - \rho + 1$.

Families: The concept of family is not new to EAs. Researchers have used various mechanisms based on relatedness (Goldberg 1989) to reduce the number of duplicates within a population, to allow convergence upon multiple optima and to promote diversity for crossover. A family in AGSAP can be defined as a collection of unique individuals which all have the same pivot and which differ only in that they have different values for the pivot gene.

The family concept is implemented by using a Family Table (FT) to keep a record of each family represented in the population. The FT is a data structure containing an entry for each family that has at least one member in the population. The information about a family stored in the FT includes:

- a pointer to the family's pivot gene;
- the set of values in the domain of the family's pivot gene that have not yet been assigned to that gene;
- the number of family members in the current population;
- a boolean matrix, called the compatibility matrix, which contains a boolean compatibility vector for each of the $(q-1)$ non-pivot genes; each such vector has a boolean entry for each value in the domain of the pivot gene.

A family's entry is removed from the FT whenever the number of its members in the population falls to zero.

The subset of the search space which is intensionally defined by an order-$(q-1)$ schema is the family defined by the schema. The size of this family is the cardinality of the domain for the family's pivot gene. When a parent in the population creates an offspring by single-point mutation, the parent (which must belong to some family having an entry in the FT) randomly selects for the pivot gene a value from the set which is indicated by the FT entry as not having yet been used by any other member of the family to create an offspring and then removes this value from the set in the entry; this prevents, during any continuous representation of the family within the population, the redundant repeated evaluation of any family member.

It may happen that not all members of a family enter the population. However, if all members of the family have entered the population during one continuous representation of the family within the population, this may, depending on the extent to which the values of the pivot gene are incompatible with the values assigned to the other genes, either (a) enable the domains of some of these other genes to be pruned of their current assignment or (b) enable some of the constraint relations between pairs of these other genes to be pruned of those 2-tuples that correspond to the current assignment of the genes. The requisite information will have been collected, while the various values of the pivot gene were generated during mutation, in the family's compatibility matrix.

Arc Revision in AGSAP: AGSAP performs arc revision in the following way. The compatibility matrix of a family contains one compatibility vector associated with each non-pivot gene of the schema defining the family. When the family entry is first established in the FT, each of the elements in the compatibility vectors is set to "True." Then, as the various members of the family arrive in the population the constraints are checked and each time a value assigned to a non-pivot gene is found to be incompatible with the pivot gene value of a new arrival belonging to the family the element of the non-pivot gene's compatibility vector corresponding to the pivot gene value of a new arrival is set to "False." Thus, if it happens that all members of the family enter the population during one continuous representation of the family in the population, the compatibility vector for each non-pivot gene will indicate which and how many of the values in the pivot domain were incompatible with the current assignment of the corresponding non-pivot gene. If any such vector has all of its elements set to "False," then the current value of the corresponding non-pivot gene is not supported by any value in the pivot gene's domain; thus, the current value of the non-pivot gene may be removed from its domain.

Pruning gene domains in this opportunistic fashion reduces the search space for subsequent evolutionary search, since the search space is the cross-product of the domains. Of course, if any domain gets pruned to the empty set, AGSAP terminates its search since it recognizes that the overall search space is also empty. Finally, it is important to note that this opportunistic pruning comes at no extra cost in terms of evaluations or constraint checks.

Path Revision in AGSAP: AGSAP's approach to path revision can be understood by referring to the presentation of path revision given earlier in terms of three objects F, G and H. If all members of a family have entered the population during one continuous representation of the family in the population, each pair of non-pivot genes F and G that are connected to the pivot gene H by constraints $C_m(F,H)$ and $C_n(G,H)$ have their compatibility vectors checked, provided the pair of genes are also connected to one another by a constraint $C_l(F,G)$. If, when the pair of compatibility vectors for $\langle F,G\rangle$ are compared, no pair of corresponding entries are both True, this means that there is no member of H's domain which is simultaneously consistent with the current assignments of F and G. Thus, that pair of values may be removed from constraint $C_l(F,G)$, if present.

Using opportunistic path revision does not reduce the size of the search space but, as we saw earlier, path revision creates favorable conditions for opportunistic arc revision to do so. However, it is possible for opportunistic path revision to detect inconsistent networks. AGSAP will terminate its search if by using opportunistic path revision all the tuples of a constraint are removed. Like opportunistic arc revision, opportunistic path revision comes at no extra cost.

The Test Suite

Each of the randomly generated CSPs used to test IDM, AGS, AGSA and AGSAP can be viewed as a triple $\langle n,d,\bar{t}\rangle$ where n is a randomly selected value within the interval 10..20 and represents the number of objects in the CSP as well as the domain size. For each of those objects, d is the network density and $\bar{t}$ is the average constraint tightness; values for d and $\bar{t}$ are taken from the set $\{0.1, 0.3, 0.5, 0.7, 0.9\}$. Since there are five values that can be assigned to d and $\bar{t}$, 25 $\langle d,\bar{t}\rangle$ classes of randomly generated CSPs can be created. For each of these $\langle d,\bar{t}\rangle$ classes a total of 30 instances were randomly generated, making a total of 750 randomly generated CSPs.

Results and Conclusions

We compared AGSAP (a) with the IDM hill-climbing algorithm from the literature and, to measure the differing effects of arc and path revision, (b) with the hybrid algorithm AGS that does not do any consistency processing as well as (c) with AGSA, which sits

between AGS and AGSAP because it does arc revision but no path revision. Figure 1 shows the performance of these four algorithms on the set of randomly generated CSPs. The results are organized into a matrix where each location in the matrix corresponds to a specific pair of values for density and average tightness. Each location of the matrix contains results for the four algorithms which show the first, second and third quartiles of the number of evaluations each algorithm performed when run 100 times on each of the 30 instances of that particular class of CSP, making 3000 runs for each problem class. Also recorded is the percentage of the runs on which an algorithm found a solution. If this entry for a particular algorithm is blank it means that the algorithm solved 100% of the exemplification CSPs in that class. Each run of an algorithm was allowed a maximum of 50,000 evaluations to find a solution. If a run exceeded this amount it was terminated and recorded as failing to find a solution.

In selecting the best performing algorithm for each class of CSP we first compared the percentage of the 3000 runs on which a solution was found and selected the algorithm with the highest percentage. To break any ties we selected the algorithm that performed the fewest evaluations at the second quartile. If there was still a tie then we selected the algorithm that performed the fewest evaluations at the third quartile, and if they were still tied we compared their first quartile performance. The best algorithm for each class of problem has its results printed in boldface in Figure 1.

In Figure 1, AGSAP has the best performance on 11 of the 25 classes, AGSA and IDM have the best performance on six classes each and AGS has the best performance on two classes. Figure 1 shows that AGS outperforms IDM on all classes except for $\langle d \geq 0.1, \bar{t} = 0.1\rangle$ and $\langle d = 0.1, \bar{t} = 0.3\rangle$. As $\bar{t}$ increases the difference in the performance of AGS and IDM becomes more dramatic.

When one compares the performances of AGS and AGSA, one can see the impact that opportunistic arc revision has on hybrid evolutionary search. Notice that, as the density and tightness are increased beyond $\langle d = 0.1, \bar{t} = 0.7\rangle$, AGSA outperforms AGS. This is because AGS has no mechanism with which to detect when a CSP has no consistent solution. Also notice that there are several classes where AGSA was able to conclude that around 99% of the runs involved arc inconsistent CSPs and did so after a relatively small number of evaluations, while AGS (and IDM) used their entire budgets of 50000 evaluations on each run without learning anything.

When one compares the performances of AGSA and AGSAP one can see the impact that opportunistic arc and path revision have on hybrid evolutionary search. Notice that as the density is increased beyond $\langle d = 0.1, \bar{t} = 0.7\rangle$ that the difference in the peformances of AGSA and AGSAP becomes increasingly more pronounced in favor of AGSAP. Also notice that, on the classes of CSPs with the highest tightness ($\bar{t} = 0.9$), AGSAP finds a solution 100% of the time, except for the class $\langle d = 0.1, \bar{t} = 0.9\rangle$ where it finds a solution 99.6% of the time.

Upon closer inspection, one can see that AGSA marginally outperforms AGSAP on the following classes: $\langle d = 0.1, \bar{t} = 0.5\rangle$, $\langle d = 0.1, \bar{t} = 0.7\rangle$, $\langle d = 0.3, \bar{t} = 0.3\rangle$, $\langle d = 0.7, \bar{t} = 0.3\rangle$, $\langle d = 0.7, \bar{t} = 0.5\rangle$, $\langle d = 0.9, \bar{t} = 0.1\rangle$, $\langle d = 0.9, \bar{t} = 0.5\rangle$. For the classes of CSPs where opportunistic path revision does not play a major role in simplifying CSPs ($\langle d \geq 0.1, \bar{t} = 0.5\rangle$ and $\langle d = 0.1, \bar{t} = 0.7\rangle$) AGSAP has no significant advantage over AGSA. Since, these two algorithms are stochastic and almost identical (with the exception of opportunistic path revision) the results are not unexpected. However, for those classes with higher averages of constraint tightness ($\langle d \geq 0.3, \bar{t} \geq 0.7\rangle$), where opportunistic path revision can play an important role in simplifying a CSP we begin to see the performance of AGSAP distance itself from the performance of AGSA.

Figure 1 suggests that using interleaved opportunistic arc and path revision can help solve some exemplification CSPs with higher averages of constraint tightness. In Figure 1, the most difficult CSPs seem to be in the classes $\langle d \geq 0.7, \bar{t} = 0.5\rangle$. These classes lay well within Smith's "mushy region" (Smith 1994) and represent a phase transition (Cheeseman 1991) between classes of exemplification CSPs based on networks with non-empty intents and those based on networks with empty intents.

Ongoing Work

Our ongoing work involves providing hybrid EAs with other beneficial aspects of systematic search and providing systematic search techniques with beneficial aspects discovered by hybridizing evolutionary search.

References

Benson Jr., Brent W., and Freuder, Eugene C. (1992). Interchangeability Preprocessing Can Improve Forward Checking Search, *Proc. ECAI'92.*

Bowen J., and Dozier, G. (1995). Solving Constraint Satisfaction Problems Using A Genetic/Systematic Search Hybrid That Realizes When to Quit, *Proc. of the 6th Intl. Conf. on Genetic Algorithms, ICGA-95*

Bowen,J., Lai, R. and Bahler D. (1992). Lexical Imprecision in Fuzzy Constraint Networks, *Proc. AAAI-92*

Cheeseman, P., Kanefsky, B., and Taylor, W. (1991). Where the *Really* Hard Problems Are, *Proc. IJCAI-91, 1991.*

Crawford K. D. (1992), Solving the N-Queens Problem Using Genetic Algorithms, *Proc. ACM/SIGAPP Symposium on Applied Computing*, Kansas City, MO.

Dozier, G., Bowen J., and Bahler D. (1994a). Solving

Small and Large Scale Constraint Satisfaction Problems Using a Heuristic-Based Microgenetic Algorithm, *Proc. of the 1994 IEEE Intl. Conf. on Evolutionary Computation.*

Dozier, G., Bowen J., and Bahler D. (1994b). Constraint Processing Using Heuristic Microgenetic Algorithms, *Proc. of the ECAI'94 Workshop on Applied Genetic and Other Evolutionary Algorithms.*

Dozier, G., Bowen J., and Bahler D. (1995). Solving Randomly Generated Constraint Satisfaction Problems Using a Micro-Evolutionary Hybrid That Evolves a Population of Hill-Climbers, *Proc. of the 1995 IEEE Intl. Conf. on Evolutionary Computation.*

Eiben, A. E., Raue P-E. and Ruttkay, Zs. (1994). Solving Constraint Satisfaction Problems Using Genetic Algorithms, *Proc. of the 1994 IEEE Intl. Conf. on Evolutionary Computation.*

Goldberg, David E. (1989). *Genetic Algorithms in Search, Optimization & Machine Learning* (Addison-Wesley Publishing Company).

Homaifar, A., Turner, J. and Ali, S. (1992). The N-Queens Problem and Genetic Algorithms, *Proc. IEEE Southeast Conf., Vol 1.*

Mackworth, A. K. (1977). Consistency in Networks of Relations. *Artificial Intelligence 8* (1977).

Michalewicz, Zbigniew (1992). *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag Artificial Intelligence Series.

Minton, S. (1992). Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems, *Artificial Intelligence 58* (1992).

Morris, Paul. (1993). The Breakout Method For Escaping From Local Minima, In *Proc. of AAAI-93.*

Smith, Barbara. (1994). The Phase Transition and the Mushy Region in Constraint Satisfaction Problems, *Proc. of ECAI'94.*

Algorithms	d	$\bar{t}=0.1$				$\bar{t}=0.3$				$\bar{t}=0.5$				$\bar{t}=0.7$				$\bar{t}=0.9$			
		Q1	Q2	Q3	%	Q1	Q2	Q3	%	Q1	Q2	Q3	%	Q1	Q2	Q3	%	Q1	Q2	Q3	%
AGSAP		6.0	7.0	9.0		17.0	25.0	37.0		48.0	85.0	145.0		244.0	500.0	1113.0		1779.5	3927.0	7033.5	99.6
AGSA	0.1	8.0	11.0	15.0		17.0	25.0	37.0		46.0	83.0	146.0		231.0	494.0	1070.5		1922.0	4132.0	7509.0	96.6
AGS		6.0	7.0	9.0		17.0	25.0	36.0		48.0	88.0	155.0		254.0	580.0	1312.0		50000	50000	50000	0.0
IDM		2.0	4.0	6.0		13.0	23.0	37.0		47.0	89.0	155.0		335.5	951.0	3124.5	99.2	50000	50000	50000	0.0
AGSAP		8.0	11.0	15.0		37.0	66.0	103.0		136.0	264.0	529.0		1998.0	11024.0	25543.5	61.8	1628.0	3083.0	4897.0	
AGSA	0.3	8.0	11.0	15.0		37.0	65.0	100.0		137.0	265.0	534.0		1802.5	11889.0	25363.5	60.7	1954.0	4282.5	6646.5	98.4
AGS		7.0	11.0	15.0		37.0	65.0	103.0		138.0	268.0	566.0		319.5	771.0	1455.0	16.7	50000	50000	50000	0.0
IDM		4.0	7.0	11.0		39.0	81.0	137.0		212.0	457.0	1042.0		557.0	1299.0	3074.5	16.7	50000	50000	50000	0.0
AGSAP		10.0	16.0	24.0		64.0	121.0	206.0		299.5	846.0	3156.0	84.5	5599.0	8027.0	14887.0	67.2	1453.5	2691.5	4301.5	
AGSA	0.5	10.0	16.0	24.0		62.0	123.0	211.0		287.0	838.5	3071.5	84.4	6193.5	8605.5	14905.5	64.3	1933.5	3662.0	6298.0	99.2
AGS		10.0	16.0	24.0		62.0	121.0	213.0		302.0	848.0	3347.0	84.8	50000	50000	50000	0.0	50000	50000	50000	0.0
IDM		7.0	13.0	21.0		80.0	181.0	363.0		498.5	1539.0	5462.0	82.3	50000	50000	50000	0.0	50000	50000	50000	0.0
AGSAP		14.0	21.0	31.0		113.0	229.0	499.0		632.0	1477.0	3177.0	16.7	5117.0	10826.0	23396.0	75.3	1085.0	1717.5	2653.5	
AGSA	0.7	14.0	21.0	31.0		115.0	224.0	495.0		595.5	1362.0	3212.5	16.7	5912.5	11495.5	24861.5	70.1	1542.5	2473.0	3927.0	98.8
AGS		15.0	21.0	31.0		114.0	233.0	534.0		736.5	1578.0	3168 .0	16.7	165.0	245.0	355.0	3.3	50000	50000	50000	0.0
IDM		10.0	18.0	28.0		184.0	404.0	1020.0		1211.5	2478.0	4938.0	16.7	200.5	300.0	425.0	3.3	50000	50000	50000	0.0
AGSAP		21.0	32.0	45.0		150.0	699.0	3157.0	99.5	152.0	201.0	293.5	3.3	10203.5	17776.5	29984.0	85.1	968.5	1569.0	2725.0	
AGSA	0.9	21.0	32.0	44.0		148.0	737.5	2912.0	99.1	130.0	173.0	236.0	3.3	9953.0	19277.0	31972.5	73.2	1348.0	2296.0	4136.0	99.2
AGS		21.0	32.0	44.0		149.0	784.0	2811.0	99.6	142.5	209.0	256.5	3.3	50000	50000	50000	0.0	50000	50000	50000	0.0
IDM		17.0	30.0	45.0		250.0	1404.0	4901.0	98.1	229.5	332.0	481.0	3.3	50000	50000	50000	0.0	50000	50000	50000	0.0

Figure 1: Comparison of IDM and the Hybrid EAs