An Efficient Heuristic-Based Evolutionary Algorithm for Solving Constraint Satisfaction Problems

Vincent Tam and Peter Stuckey*

Abstract

GENET and EGENET are local search algorithms based on artificial neural networks which have proved very successful in solving hard constraint satisfaction problems (CSPs). In this paper we describe a micro-genetic algorithm for solving CSPs which generalizes the (E)GENET approach. It is based on min-conflict local search together with two methods for escaping local minima: population based learning, and look forward. We show that this approach improves on EGENET in solving hard CSPs.

1 Introduction

A constraint satisfaction problem (CSP) can be specified as a triple $\langle Z, D, C \rangle$ involving a finite set Z of variables, a function D which maps each variable $X \in Z$ to a finite set (domain) D_X of possible values, and a finite set C of constraints defined on these variables. CSPs naturally occur in many important industrial applications, such as planning, scheduling and resource allocation, in Artificial Intelligence and Operations Research. Given a CSP, the task is to assign values, from their associated domains, to all the variables so that no constraints in C is violated. Solving CSPs is, in general, NP-complete.

There are two main approaches to solving CSPs. Complete search methods involve some form of backtracking search that systematically explore the entire search space. Hence they ensure finding a solution if one exists. On the other hand, local search methods, such as artificial neural networks (ANNs), evolutionary algorithms (EAs) and simulated annealing (SA), begin from a valuation and make local improvements until a good solution is found. Local search methods may not be able to find a solution to a CSP even when one exists. Recently local search methods have been shown to be more efficient than the complete search methods on solving some large-scale or hard instances of real-life CSPs [6, 4, 9, 1]. In this paper we concentrate on local search methods.

A basis for many local search methods is the min-conflict heuristic (MCH) [9]. The idea behind MCH is to assign a value, which is locally minimum in terms of constraint violations, to each variable so as to reach a global minimum. When there are several local minima (ties) in terms of constraint violations, a value will randomly be selected among these ties. Thus, MCH performs the steepest-descent step after considering every possible value in the domain of the selected variable. The min-conflict heuristic has been used in a number of successful local search methods, for example [9] shows that it can solve the million-queens problem in minutes.

GENET [3] is a MCH-based ANN for solving general CSPs with binary constraints. It has had remarkable success in solving certain hard CSPs such as hard graph coloring problems. Lee *et. al.* [7] extended GENET to EGENET with a generic representation scheme for handling both binary and non-binary constraints. EGENET has also been successfully applied to solve non-binary CSPs such as the car-sequencing problems and cryptarithmetic problems in an efficient manner. Both GENET and EGENET combine an MCH-based state update rule with a heuristic learning rule. Initially, a complete and random variable assignment is generated. Then, each variable is asynchronously updated by the MCH-based state update rule in each convergence cycle. When there is no change in any value assigned to the variables, the network is trapped in a local minima. If the local minimum does not represents a solution, a heuristic learning rule, is applied to help the network escape from these local minima. The network convergence procedure iterates until a solution is found or a predetermined resource limit is exceeded.

Besides ANNs, evolutionary algorithms are another example of local search methods capable of efficiently solving CSPs [5, 2, 11]. To model a CSP in an EA each variable in a CSP is usually represented by a gene, and a valuation for all variables is thus a chromosome. The evaluation function is usually defined in

*Department of Computer Science, The University of Melbourne, Australia, {vtam,pjs}@cs.mu.oz.au

terms of the number of constraint violations. The evolutionary algorithm basically performs a parallel local search in different parts of the search space by maintaining a population of different chromosomes over the successive generations. In each generation, the evolutionary algorithm applies some genetic operators such as mutation or crossover to produce offspring from the current population. Then, the offspring produced are ranked and the "best" selected according to the evaluation function to form the next generation. This "selection-and-reproduction" process is repeated until all the chromosomes have converged to the same local minima, or some predetermined resource limit is exceeded. A resource limit is usually defined in terms of CPU time or the maximum number of generations allowed.

When evolutionary algorithms are applied to solve CSPs with a large number of variables (say > 500), the total computational costs can be minimized by focusing the search only on a reasonably small population of chromosomes without much impact on the search efficiency. This is the idea behind microgenetic algorithms (MGAs) which is a subclass of evolutionary algorithms based on a small population size (usually < 20). Recently, it has been reported that MGAs can find solutions with fewer iterations than evolutionary algorithms with larger population sizes for some problems [8, 5].

Even integrated with some useful heuristics, most MGAs, like the conventional EAs, still depend heavily on the selection criteria of the evaluation function to guide the parallel local searches towards the global optimum of the evaluation function. On solving some moderately or highly constrained real-life CSPs with MGAs, this approach can be inefficient because of the small population size of an MGA.[1] The highly coupled nature of the variables can cause repeated failures in the evolutionary search because of specific combinations of values violating constraints in these hard problems.

To improve the efficiency of EAs on solving real-life CSPs, we propose the integration of MCH and some learning mechanism into MGAs. In this paper, we define a generic MCH-based MGA with two mutation operators and a pivoting scheme which minimizes an evaluation function defined in terms of the number of constraint violations. One of the basic mutation operators is MCH-based while the other mutation operator simply performs probabilistic descent. To our knowledge, our work represents the first attempt to integrate MCH into evolutionary algorithms. An obvious advantage of our generic heuristic-based MGA is that many local search methods such as simulated annealing can readily be integrated as another heuristic operator to work with the MCH-based mutation to optimize the evaluation function.

As in the case of GENET and EGENET, this generic MCH-based MGA will easily be trapped in local minima before finding a solution representing the global minimum of the evaluation function. To improve its performance, we propose two different heuristic operators either to avoid or escape from these local minima. The first, population-based learning, is a generalization of the EGENET learning rule. The second, look-forward, in effect explores a broader local neighbourhood in order to avoid falling in local minima. Our experimental results show that these heuristics are particularly useful in solving some hard instances of CSPs.

This paper is organised as follows. Section 2 describes the generic MCH-based MGA and its two basic heuristic mutation operators for updating the variables. In Section 3, we explain in detail the population-based learning and look-forward heuristic operators for improving the generic MCH-based MGA. Finally, in Section 4 we discuss and analyse our experimental results to comparing EGENET implemented as a specific instance of the generic MCH-based MGA and the generic MCH-based MGA improved with different combinations of our proposed heuristic operators.

2 A Generic MCH-Based Microgenetic Algorithm for Solving CSPs

Given a CSP specified as a triple $\langle Z, D, C\rangle$, where $Z = \{X_1, \ldots, X_n\}$ each variable X_i defines a gene which can take value from the domain D_{X_i}. A chromosome is the sequence of values for each variable, in other words a valuation. The evaluation function $fitness$ is defined to map a chromosome to the number of constraints in C which are violated by the valuation of the chromosome. An evolutionary algorithm to solve the CSP is then a function $evolve(\langle Z, D, C\rangle, fitness, PZ, MAX\_GENS) = \langle best\_fit, P, ngens\rangle$ which takes the CSP and fitness function, together with a population size PZ and maximum number MAX_GENS of generations allowed for finding a solution to the CSP, and returns a triple of the best fitness value $best\_fit$, the resulting popula-

[1] Because of the small population size, the selection pressure exerted by the evaluation function is small. In the other words, the mating pool becomes less competitive for small population size.

tion P with PZ chromosomes and the number $ngens$ of generations used. Whenever the function $evolve()$ returns the $best\_fit$ equal to zero, there are no constraint violations and hence, there is, at least, one solution existing in the population P. Otherwise if the function $evolve()$ returns a positive value for $best\_fit$, the evolutionary algorithm has failed to find a solution. Clearly MGAs are just specific instances of the function $evolve()$ with PZ small.

The principle operator in the MGA is the min-conflict based mutation operator apply_MCH which examines a gene and modifies its value to be that with the lowest total constraint violations. However applying MCH to all the genes of every chromosome in the population for each generation can be computationally expensive. Therefore, in order to reduce the computational cost for most genes we use a cheaper mutation operator single_pt_mutate which performs probabilistic descent. To determine which variable $X \in Z$ will be mutated using the MCH-based operator, we need a pivoting scheme (the function select_pivot) to select the most important variable (called the pivot gene) of a particular chromosome. All of these basic functions are described in detail in the remainder of the section.

We define a generic MGA named "MCH-MGA0" based on MCH, probabilistic descent and a pivoting scheme, as a first step to solve CSPs. Initially, all the chromosomes in the population P of the evolutionary algorithm MCH-MGA0 are assigned values randomly. Then, each chromosome cs is treated in turn. It is evaluated according to the fitness function fitness(cs). If the fitness is zero, then a solution has been found and the algorithm returns the current population. Otherwise, the function select_pivot selects an important variable V (the pivot gene) for the chromosome, to which to apply the MCH-based mutation apply_MCH. All other variables are first possibly updated using the mutation function single_pt_mutate. The procedure iterates until a solution is found, or the number $ngens$ of generations used exceeds the predetermined limit MAX_GENS.

We propose two alternative pivot selection mechanisms, namely usage-select and update-select, to efficiently guide the search of our generic MCH-based MGA and their improved versions. The selection criteria used in these mechanisms is based on the number of constraint violations and a hereditary value (h-value) for each gene. The two selection mechanisms only differ in the ways in which the h-value is used to guide the search. The first selection function usage-select is based on the usage of each gene as a pivot gene in the previous generations. When a gene is used as a pivot gene, its h-value will be set to 1. In each generation, usage-select only consider the unused genes (with h-value $= 0$) and chooses the one with the largest number of constraint violations. When there is a tie between the unused genes, the ties are broken randomly. Eventually every gene has been used as a pivot gene and all h-values are reset to 0. The second selection mechanism update_select is based on the effects of reassigning values to the genes. It only considers the genes (variables) possibly affected by updates of other genes in previous generations. The function update_select selects the gene with h-value equal to 0 and the maximum number of constraint violations. As in the previous mechanism, ties are broken randomly. Whenever a gene V is updated, its h-value is set to 1, and the each other gene V' which occurs in a constraint with V has its h-value reset to 0. Hence it will be reconsidered as a pivot gene.

The function apply_MCH() is the same as the variable updating function applied to every variable in a GENET or EGENET convergence cycle. Given a chromosome cs and gene V it generates all the chromosomes cs_i that result from modifying cs so that the gene V takes every possible value $v_i \in D_V$. Let $ties$ be the subset of the cs_i with best fitness (least violations). When the original chromosome cs is an element of $ties$ then it is the result of the operation (thus the operation is stable). Otherwise a random element of $ties$ is the resulting chromosome.

The function single_pt_mutate performs a single-point mutation to achieve an opportunistic local improvement of gene V in the current chromosome. It can be regarded as an alternative variable updating component with less computational cost than the MCH-based variable updating function apply_MCH. The function single_pt_mutate applied to gene V in chromosome cs randomly generates a different value v' from the current value v of the gene V to give a new chromosome cs'. It then compares the fitness value of cs' with the fitness value of cs. If cs' is less fit than cs then cs is returned by the function. Otherwise cs' is returned. Note that this single-point heuristic mutation operator intentionally assigns a new value to a gene even when the fitness value of the new value is the same as that of the previously assigned value (as opposed to apply_MCH) in order to explore other parts of the search space.

Unlike most evolutionary algorithms MCH-MGA0 does not ever rank the population or apply crossover operations to create children from two or more parents.[2] Instead each chromosome is modified to cre-

[2] Our experiments with crossover operators were uniformly

ate its successor. The descending nature of our basic mutation operators, apply_MCH and single_pt_mutate, automatically ensures that an (single) offspring will always have no worse fitness value than its parent. In this way, our evolutionary algorithm MCH-MGA0 can be considered as using a special selection mechanism which allows only offspring with no worse fitness than their parents to enter into the mating pool. Otherwise, the selection function just replicates the parent into the mating pool. When the best fitness value of the population in our proposed MCH-MGA0 remains the same for consecutive evaluations, it is more likely that the "fittest" chromosome(s) in the population represents some local minima.

3 Heuristic Operators for Improving the Generic MGA

Our generic MCH-based evolutionary algorithm MCH-MGA0 can easily get trapped at local minima because it only tries local improvements to the genes of an individual chromosome. To overcome this problem, we propose the population-based learning and look-forward heuristics. The population-based learning is based on the idea of group improvement on the population basis instead of the local improvement on each individual chromosome. The look-forward algorithm is an intelligent search strategy assisting the local improvement method to look for the other opportunistic improvements. Both methods can generally be used alone or combined with the other heuristic operators for improving MCH-MGA0.

3.1 Population-Based Learning

The population-based learning method (popu_learn) is a generalization of the GENET learning mechanism. The basic idea is that when the search becomes trapped at a local minimum the violated constraints at this minimum are penalized more. This changes the landscape of fitness values and pushes search away from the local minimum. In MCH-MGA0 learning is based on the whole population instead of the individual candidate solution. Initially, the penalty values for all constraints are set to 1. Whenever the whole population cannot be further improved by the local improvement operators, that is all are staying at some local minima (equilibrium point), the function popu_learn is invoked to check for each constraint $c \in C$ against the current variable assignments in the chromosomes of the whole population. If any of these variable assignments violates the constraint c then the penalty value for c is increased by 1. Clearly when the population size PZ equals 1 in MCH-MGA0, the function popu_learn is the same as the heuristic learning method used in GENET and EGENET. Hence, both GENET and EGENET can be regarded as specific instances of our generic MCH-based MGA with $PZ = 1$ and improved with popu_learn, where single_pt_mutate is not used and pivot genes are selected by usage-select. Note that population based learning provides the only communication between chromosomes, and is what makes out approach more than simply a parallel search.

3.2 Look-Forward Algorithm

The population-based learning rule is aimed at avoiding some previously assigned values involved in constraint violation. This adjusts the whole landscape of fitness values so as to consider the other chromosomes which have not been visited in the search. In some highly constrained CSPs, using this learning rule to escape from local minima can be a fairly slow process. Hence, we consider *looking a step forward* by modifying the pivot gene update to examine other values for the pivot gene. If another value could lead to a globally better chromosome if other genes were also modified then the resulting chromosome is the result of look-forward. This, in effect, wider local search can escape local minima that would trap MCH because it looks further away from the current chromosome.

The look-forward algorithm look_forward applies to each chromosome in the population individually. It examines the set *ties* built in apply_MCH. If the set is a singleton, that is there is only one value for the pivot gene with best fitness, it sets *candidates* to all the chromosomes cs_i with second best fitness. Otherwise *candidates* is set to *ties* with the current chromosome cs removed.

For each element in *candidates* a small local search is performed to see if it can lead to a better solution. Starting from a candidate, for each gene other than the pivot a single_pt_mutate operation is applied. Let *results* be the set of resulting chromosomes. The result of the look-forward operation is the best chromosome in $results \cup \{cs\}$, with ties broken randomly. Looking forward in this manner is feasible because the mutation operator single_pt_mutate is computationally cheap. Also the set *candidates* tends to be small.

We also use a a simplified version of the look-forward algorithm (lazy_look_forward) which is only invoked when the best fitness value of the pivot gene is

unsuccessful on improving over EGENET

already zero (violating no constraint) and there are alternative values collected in the array *ties* for further consideration. This means there is no broader search of the neighbourhood except for genes and values which satisfy all constraints. It is at this point where sideways moves can be most valuable.

4 Experimental Results

To demonstrate the efficiency of our generic MCH-based MGA (MCH-MGA0) with the improving heuristic operators, we build a prototype implementation with GCC version 2.7.2 on Digital Unix version 3.2F. We compare the benchmarks with EGENET implemented as a specific instance of MCH-MGA0 [3] and the different improved versions of MCH-MGA0 on a number of real-life CSPs on a DEC AlphaServer 8400 running at 300Mhz. The reported CPU time is measured in seconds. All the data are averages over 10 runs.

We compare the original heuristic-based MCH-MGA0, EGENET with a fixed order of selecting pivot genes (EGENET-FIX) and random order of selecting pivot genes (EGENET-RND), the MCH-MGA0 improved with the population-based learning method and look_forward (PL+LF) or lazy_look_forward (PL+LLF) on a set of hard graph-coloring problems and a set of hard job-shop scheduling problems. All the benchmarks are executed using the update_select pivoting strategy, because it uniformly improved upon usage_select. We also tried incorporating a number of versions of multiple-point crossover operator into our generic MGA. But they all failed to solve any CSPs in our benchmarks.

4.1 Hard Graph Coloring Problems

The following hard graph coloring problems on which we compared the different MGAs are obtained from [4]. These problems have been shown to defeat many algorithmic complete search methods.

Table 1 shows the average timings required for each of our tested MGAs to solve these problems. The MCH-MGA0 improved with popu_learn and lazy_look_forward performs the best among the other MGAs since the lazy look-forward algorithm is invoked less frequently than the full look-forward algorithm. EGENET-FIX betters EGENET-RND on these problems with some particular fixed ordering of variables for updating. PL+LF failed to find a solution (before exhausting the generation limit) in the harder case showing the disadvantage of the frequent invocations of the full look-forward algorithm in MCH-MGA0 may lead to some unpromising branches in the search tree for certain hard instances of CSPs.

4.2 Hard Job-shop Scheduling Problems

The benchmark problems are selected from a set of job-shop scheduling problems [10] from Carnegie Mellon University. These problems have been widely used for comparison of the performance of different schedulers. The goal for each problem is to fulfil all operations with a given time bound.

For these problems an additional operation was used, bounds propagation (BP) [12]. Bounds propagation removes values from the domains of variables which are not bounds consistent. It is applied before MCH-MGA0 is applied to the problem. The reduced domains of variables reduces the search space. For all the MGAs we compare on these job-shop scheduling problems, the suffix "+BP" is added to denote the integration of BP technique into the corresponding solver.

Table 2 shows the timings required for the different heuristic-based MGAs to solve these hard problems. For each case, the problem size $(m \times n)$ is shown, where m denotes the number of jobs in each problem and n stands for the number of tasks contained in each job. For all the problems, EG-RND spends about $\frac{1}{4}$ of the time required by EG-FIX showing the importance of pivot gene selection on the performance of GENET or EGENET computation.

The bounds propagation technique generally improved the performance of all the MGAs on these difficult problems since it can prune the search space by a significant amount. When applied to the EGENET solvers, the bounds propagation technique achieved a more significant improvement as compared to the other improved versions of the generic MGAs. This is because the EGENET computation checks every possible value for each variable, and hence is more sensitive to the domain sizes of the variables. The other MGAs depend more on the single-point mutation operator which only picks one random value for a variable at a time. Thus, they are less sensitive to the domain sizes of the variables.

All our improved versions of MCH-MGA0 perform 10 to 100 times faster than the EGENET solvers showing the advantages of applying the population-based learning and look-forward algorithm on these hard CSPs. The MCH-MGA0 improved

[3]For EGENET, the population size $PZ = 1$ with $MAX\_GENS = 1,000,000$. For the other MGAs, $PZ = 6$ (using the same population size as in [5]) with $\frac{1}{6}$ of the resource limit for EGENET.

Nodes	Colors	EGENET-FIX/RND	PL + LF	PL + LLF
125	18	7628s/9209s	7572s	1638s
125	17	47.63hr/137.9hr	failed	37.39hr

Table 1: Results on hard graph coloring problems

Problem	EG-FIX/RND	EG-FIX/RND + BP	PL + LF	PL + LF + BP	PL + LLF	PL + LLF + BP
j3 (10x5)	114.3s/13.29s	6.633s/9.300s	1.850s	1.370s	1.340s	0.890s
j4 (10x5)	241.5s/10.90s	13.47s/6.480s	1.620s	1.380s	1.270s	0.930s
j8 (10x10)	28139s/7292s	18101s/2879s	191.0s	126.3s	127.4s	112.6s
j9 (10x10)	21981s/4254s	16027.2s/2380s	131.0s	70.76s	104.7s	70.36s

Table 2: Results on hard job-shop scheduling problems

with the population-based learning, lazy look-forward algorithm and bounds propagation technique performs the best among the improved MGAs demonstrating the efficiency of this combination on solving hard CSPs. This can be explained by the fact that whenever the pivot gene V satisfies the constraints locally, the lazy look-forward algorithm can actively and carefully look into the other possible values in *ties* for V before invoking the generalized population-based learning mechanism to avoid the current local minima in the future search.

References

[1] A. Davenport, E.P.K. Tsang, C.J. Wang, and K. Zhu. GENET: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. In *Proceedings of AAAI'94*, 1994.

[2] Bowen James and Gerry Dozier. Solving Constraint Satisfaction Problems Using A Genetic/Systematic Search Hybrid That Realizes When to Quit. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 122 – 129, 1995.

[3] C. Wang and E. Tsang. Solving satisfaction problems using neural-networks. In *Proceedings of IEE Second International Conference on Artificial Neural Networks*, 1991.

[4] D. Johnson, C. Aragon, L. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation; Part II, graph coloring and number partitioning. *Operations Research*, 39(3):378 – 406, 1991.

[5] G. Dozier, J. Bowen, and D. Bahler. Solving small and large scale constraint satisfaction problems using a heuristic-based microgenetic algorithm. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, 1994.

[6] H. Adorf and M. Johnston. A discrete stochastic neural network algorithm for constraint satisfaction problems. In *Proceedings of the International Joint Conference on Neural Networks*, 1990.

[7] J.H.M. Lee, H.F. Leung, and H.W. Won. Towards a more efficient stochastic constraint solver. In *Proceedings of Principles and Practice of Constraint Programming (CP96)*, 1996.

[8] C. L. Karr. Air-injected hydrocyclone optimization via genetic algorithm. In *Handbook of Genetic Algorithms*.

[9] S. Minton, M. Johnston, A. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161 – 205, 1992.

[10] N. Sadeh. A set of hard job-shop scheduling problems. Available by anonymous FTP from "cimds3.cimds.ri.cmu.edu".

[11] A. Thornton. Genetic algorithms versus simulated annealing : Satisfaction of large sets of algebraic mechanical design constraints. In *Artificial Intelligence in Design'94*, pages 381–398, 1994.

[12] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.