Distributed Constraint Satisfaction, Restricted Recombination, and Hybrid Genetic Search

Gerry Dozier, Hurley Cunningham, Winard Britt, and Funing Zhang

Department of Computer Science and Software Engineering
Auburn University, AL 36849-5347
gvdozier@eng.auburn.edu

Abstract. In this paper, we present simple and genetic forms of an evolutionary paradigm known as a society of hill-climbers (SoHC). We compare these simple and genetic SoHCs on a test suite of 400 randomly generated distributed constraint satisfaction problems (DisCSPs) that are composed of asymmetric constraints (referred to as DisACSPs). Our results show that all of the genetic SoHCs dramatically outperform the simple SoHC even at the phase transition where the most difficult DisACSPs reside.

1 Introduction

Evolutionary Computation (Bäck 1997; Fogel 2000) is the field of study devoted towards the design, development, and analysis of problem solvers based on simulated genetic and/or social evolution. Evolutionary computations (ECs) have been successfully used to solve a wide variety of problems in the areas of robotics, engineering, scheduling, planning, and machine learning, just to name a few (Bäck 1997; Fogel 2000).

In the Evolutionary Constraint Satisfaction community, we have seen a migration away from pure evolutionary computations for constraint satisfaction towards the hybridization of ECs with traditional CSP techniques and/or the incorporation of heuristics and problem specific knowledge (Dozier, Bowen, and Homaifar 1998) in an effort to solve CSPs more efficiently. To date, much of this research has focused on centralized CSPs. Little research has been conducted by the evolutionary constraint satisfaction community on the development of ECs for solving distributed constraint satisfaction problems (DisCSPs) (Dozier 2002).

A DisCSP (Yokoo 2001) can be viewed as a 4-tuple (X, D, C, A), where X is a set of n variables, D is a set of n domains (one domain for each of the n variables), C is a set of constraints that constrain the values that can be assigned to the n variables, and A is a set of agents for which the variables and constraints are distributed. Constraints between variables belonging to the same agent are referred to as intra-agent constraints while constraints between the variables of more than one agent are referred to as inter-agent constraints. The objective in solving a DisCSP is to allow the agents in A to develop a consistent distributed solution by means of message passing. The constraints are considered private and are not allowed to be communicated to fellow agents due to privacy, security, or representational reasons (Yokoo 2001). When comparing the effectiveness of DisCSP-solvers the number of communication cycles (through the distributed algorithm) needed to solve the DisCSP at hand is more important than the number of constraint checks (Yokoo 2001) .

Many real world problems have been modeled and solved using DisCSPs (Bejar et al. 2001; Calisti and Faltings 2001; Freuder, Minca, and Wallace 2001; Silaghi et al. 2001; Zhang 2002); however, many of these approaches use mirrored (symmetric) inter-agent constraints. Since these inter-agent constraints are known by the agents involved in the constraint, they can not be regarded as private. If these constraints were truly private then the inter-agent constraints of one agent would be unknown to the other agents involved in those constraints. In this case the DisCSP would be composed of asymmetric constraints. To date, with the exception of (Freuder, Minca, and Wallace 2001) and (Silaghi et al. 2001), little research has been done on distributed asymmetric CSPs (DisACSPs).

In this paper, we demonstrate how distributed restricted forms of uniform crossover can be used to improve the effectiveness of a previously developed EC for solving DisACSPs known as a society of hill-climbers (SoHC) (Dozier 2002). We refer to these new algorithms as a genetic SoHCs (GSoHCs). Our results show that the GSoHCs dramatically outperform the simple SoHC even at the phase transition where the hardest DisACSPs reside.

2 Constraint Networks, Asymmetric Constraints, and the Phase Transition

Constraint satisfaction problems (CSPs) are based on constraint networks (Bowen and Dozier, 1996; Mackworth 1997). A constraint network is a triple $\langle X, D, C\rangle$ where X is set of variables, D is set of domains where each $x_i \in X$ takes its value from the corresponding domain $d_i \in D$, and where C is a set of r constraints. Consider a binary constraint network (one where each constraint constrains the value of exactly two variables)[1]

Constraint networks possess two additional attributes: tightness and density. The tightness of a constraint is the ratio of the number of tuples disallowed by the constraint to the total number of tuples in $d_i \times d_j$. The average constraint tightness of a binary constraint network is the sum of the tightness of each constraint divided by the number of constraints in the network. The density of a constraint network is the ratio of the number of constraints in the network to the total number of constraints possible.

[1] In this paper, we only consider binary constraint networks because any constraint involve more than one varible can be transformed into a set of binary constraints.

2.1 Asymmetric Constraints

Constraints in a binary constraint network may also be represented as two directional constraints referred to as arcs (Mackworth 1977; Tsang 1993). For example, the symmetric constraint c_{EF} can be represented as $c_{EF} = \{c_{\overrightarrow{EF}}, c_{\overleftarrow{EF}}\}$, where $c_{\overrightarrow{EF}} = c_{\overleftarrow{EF}} = \{\langle$ `e1,f2` $\rangle$, $\langle$ `e1,f3` $\rangle$, $\langle$ `e2,f2` $\rangle$, $\langle$ `e3,f2` $\rangle\}$, where $c_{\overrightarrow{EF}}$ represents the directional constraint imposed on variable F by variable E, and where $c_{\overleftarrow{EF}}$ represents the directional constraint imposed on variable E by variable F. This view of a symmetric binary constraint admits the possibility of an asymmetric binary constraint between variables E and F as one where $c_{\overrightarrow{EF}} \neq c_{\overleftarrow{EF}}$.

2.2 Predicting the Phase Transition

Classes of randomly generated CSPs can be represent as a 4-tuple $(n,m,p1,p2)$ (Smith 1994) where n is the number of variables in X, m is the number of values in each domain, $d_i \in D$, $p1$ represents the constraint density, the probability that a constraint exists between any two variables, and $p2$ represents the tightness of each constraint.

Smith (Smith 1994) developed a formula for determining where the most difficult symmetric randomly generated CSPs can be found. This equation is as follows, where $\hat{p2}_{crit_S}$ is the critical tightness at the phase transition for n, m, $p1$.

$$\hat{p2}_{crit_S} = 1 - m^{\frac{-2}{p1(n-1)}} \tag{1}$$

Randomly generated symmetric CSPs of the form $(n,m,p1,\hat{p2}_{crit_S})$ have been shown to be the most difficult because they have on average only one solution. Problems of this type are at the border (phase transition) between those classes of CSPs that have solutions and those that have no solution. Classes of randomly generated symmetric CSPs for which $p2$ is relatively small compared to $\hat{p2}_{crit_S}$ are easy to solve because they contain a large number of solutions. Similarly, classes of CSPs where $p2$ is relatively large compared to $\hat{p2}_{crit_S}$, are easy to solve because the constraints are so tight that simple backtrack-based CSP-solvers (Smith 1994) can quickly determine that no solution exists. Thus, for randomly generated CSPs, one will observe an easy-hard-easy transition as $p2$ is increased from 0 to 1.

Smith's equation can be modified (Dozier 2002) to predict the phase transition in randomly generated asymmetric CSPs as well. This equation is as follows where $p1_\alpha$ represents the probability that an arc exits between two variables and where $\hat{p2}_{crit_A}$ is the critical tightness at the phase transition for n, m, and $p1_\alpha$.

$$\hat{p2}_{crit_A} = 1 - m^{\frac{-1}{p1_\alpha(n-1)}}. \tag{2}$$

3 Society of Hill-Climbers

A society of hill-climbers (SoHC) (Dozier 2002; Sebag and Schoenauer 1997) is a collection of hill-climbers that communicate promising (or futile) directions of search to one another through some type of external collective structure. In the society of hill-climbers that we present in this paper, the external collective structure which records futile directions of search comes in the form of a distributed list of breakout elements, where each breakout element corresponds to a previously discovered nogood[2] of a local minimum (Morris 1993). Before presenting our society of hill-climbers, we must first discuss the hill-climber that makes up the algorithm. In this section, we first introduce a modified version of Yokoo's distributed breakout algorithm with broadcasting (DBA+BC) (Yokoo 2001) which is based on Morris' Breakout Algorithm (Morris 1993). After introducing the modified DBA+BC algorithm (mDBA) we will describe the framework of a SoHC.

For the mDBA, each agent $a_i \in A$ is responsible for the value assignment of exactly one variable. Therefore agent a_i is responsible for variable $x_i \in X$, can assign variable x_i one value from domain $d_i \in D$, and has as constraints $C_{\overrightarrow{x_i x_j}}$ where $i \neq j$. The objective of agent a_i is to satisfy all of its constraints $C_{\overrightarrow{x_i x_j}}$. Each agent also maintains a breakout management mechanism (BMM) that records and updates the weights of all of the breakout elements corresponding to the nogoods of discovered local minima. This distributed hill-climber seeks to minimize the number of conflicts plus the sum of all of the weights of the violated breakout elements.

3.1 The mDBA

The mDBA used in our SoHCs is very similar to Yokoo's DBA+BC with the major exception being that each agent broadcasts to every other agent the number of conflicts that its current value assignment is involved in. This allows the agents to calculate the total number of conflicts (fitness) of the current best distributed candidate solution (dCS) and to know when a solution has been found (when the fitness is equal to zero). The mDBA, as outlined in Figure 1, is as follows.

Initially, each agent, a_i, randomly generates a value $v_i \in d_i$ and assigns it to variable x_i. Next, each agent broadcasts its assignment, $x_i = v_i$, to its neighbors $a_k \in Neighbor_i$ where $Neighbor_i$[3] is the set of agents that a_i is connected with via some constraint. Each agent then receives the value assignments of every neighbor. This collection of value assignments is known as the **agent_view** of an agent a_i (Yokoo 2001). Given the agent_view, agent a_i computes the number of conflicts that the assignment $(x_i = v_i)$ is involved in. This value is denoted as γ_i.

[2] A nogood is a tuple that causes a conflict.
[3] In this paper, $Neighbor_i = A - \{a_i\}$.

Once the number of conflicts, γ_i, has been calculated, each agent a_i randomly searches through its domain, d_i, for a value $b_i \in d_i$ that resolves the greatest number of conflicts (ties broken randomly). The number of conflicts that an agent can resolve by assigning $x_i = b_i$ is denoted as r_i. Once γ_i and r_i have been computed, agent a_i broadcasts these values to each of its neighbors.

When an agent receives the γ_j and r_j values from each of its neighbors, it sums up all γ_j including γ_i and assigns this sum to f_i where f_i represents the fitness of the current dCS. If agent a_i has the highest r_i value of its neighborhood then agent a_i sets $v_i = b_i$, otherwise agent a_i leaves v_i unchanged. Ties are broken randomly using the commonly seeded tie-breaker[4] that works as follows: if t(i) > t(j) then a_i is allowed to change otherwise a_j is allowed to change where t(k) = (k+$rnd()$) mod $|A|$, and where $rnd()$ is a commonly seeded random number generator used exclusively for breaking ties.

If r_i for each agent is equal to zero, i.e. if none of the agents can resolve any of their conflicts, then the current best solution is a local minimum and all agents a_i send the nogoods that violate their constraints to their BMM_i. An agent's BMM will create a breakout element for all nogoods that are sent to it. If a nogood has been encountered before in a previous local minimum then the weight of its corresponding breakout element is incremented by one. All weights of newly created breakout elements are assigned an initial value of one. Therefore the task for mDBA is to reduce the total number of conflicts plus the sum of all breakout elements violated.

After the agents have decided who will be allowed to change their value and invoked their BMMs (if necessary), the agents check their f_i value. If $f_i > 0$ the agents begin a new cycle by broadcasting their value assignments to each other. If $f_i = 0$ the algorithm terminates with a distributed solution.

3.2 The Simple and Genetic SoHCs

The SoHCs reported in this paper are based on mDBA. Each simple SoHC runs ρ mDBA hill-climbers in parallel, where ρ represents the society size. Each of the ρ hill-climbers communicate with each other indirectly through a distributed BMM. In a SoHC, each agent, a_i assigns values variables $x_{i1}, x_{i2}, \cdots, x_{i\rho}$ where each variable x_{ij} represents the i^{th} variable for the j^{th} dCS. Each agent, a_i, has a local BMM (BMM_i) which manages the breakout elements that correspond to the nogoods of its constraints.

There are a total of 4 genetic SoHCs (GSoHCs) reported in this paper. They differ only in the type of recombination operator used. They are as follows. GSoHC$_{spx}$ is similar to the simple SoHC (SoHC) except that it uses a distributed restricted single-point crossover operator (dSPX-μ), where μ is the mutation rate. The dSPX-μ operator works as follows. On each cycle, each dCS$_j$ that has an above average number of conflicts is replaced with an offspring by recombining

[4] In case of a tie between two agents a_i and a_j, Yokoo's DBA+BC will allow the agent with the lower agent address number is allowed change its current value assignment. We refer to this as the deterministic tie-breaker (DTB) method

```
procedure mDBA(Agent $a_i$)
{

     Step 0: randomly assign $v_i \in d_i$ to $x_i$;
     do
         {
             Step 1: broadcast ($x_i = v_i$) to other agents;
             Step 2: receive assignments from other agents, agent_view$_i$;
             Step 3: assign $conflicts_i$ the number conflicts that ($x_i = v_i$)
                     is involved in;
             Step 4: randomly search $d_i$ for a value $b_i$ that minimizes the number
                     of conflicts of $x_i$ (ties broken randomly),
             Step 5: let $r_i$ equal the number of conflicts resolved by ($x_i = b_i$);
             Step 6: broadcast $conflicts_i$ and $r_i$ to other agents;
             Step 7: receive $conflicts_j$ and $r_j$ from other agents,
                     let $f = \sum conflicts_k$;
             Step 8: if (max($r_k$) == 0)
                              for each conflict, ($x_i = v$, $x_j = w$)
                                 update_breakout_elements($BMM_i$,($x_i = v$, $x_j = w$));
             Step 9: if ($r_i$ == max($r_k$))$^\dagger$                          $v_i = b_i$;
         } while ($f > 0$)
}
      † Ties are broken with randomly with a synchronized tie-breaker.
```

Fig. 1. mDBA Agent Protocol

dCS$_j$ with dCS$_q$, which is created as follows. With probability μ, agent a_i will randomly assign v_{ij} a value from d_i. With probability 1-μ, a cut point, *cp*, is selected using a commonly seeded random number generator from 1 to N-1, where N is the number of agents of an individual. All agents a_i where i ¡ *cp* will assign v_{ij} the value v_{iq}. This takes place with probability $\frac{1-\mu}{2}$. With probability $\frac{1-\mu}{2}$, All agents a_i where $i \geq$ *cp* will assign v_{ij} the value v_{iq}. In this fashion, the new dCS$_j$ will have values up to *cp* from dCS$_q$ and values from *cp* to N from dCS$_j$ or it will have have values up to *cp* from dCS$_j$ and values from *cp* to N from dCS$_q$ which is the way single-point crossover works on centralized CSs.

GSoHC$_{tpx}$ uses a distributed restricted two-point crossover operator (dTPX-μ) that is based on two-point crossover, while GSoHC$_{mtpx}$ uses a modified dTPX-μ (referred to as dMTPX-μ) where the first and third segments of dCS$_j$ are assigned values from dCS$_q$.

GSoHC$_{ux}$ works exactly like a the other GSoHCs except that on each cycle a distributed restricted uniform crossover operator is applied as follows. Each distributed candidate solution that has an above average number of conflicts, dCS$_j$, is replaced with an offspring that is a recombination of the best individual, dCS$_q$, and dCS$_j$ as follows. An agent a_i will assign v_{ij} the value from v_{iq} with probability $\frac{1-\mu}{2}$ and will leave v_{ij} unchanged with probability $\frac{1-\mu}{2}$. With probability μ agent a_i will randomly assign v_{ij} a value from d_i the domain of values for variable x_i. We refer to this form of recombination as distributed restricted uniform crossover (dRUC-μ).

The simple and genetic SoHCs compared in this paper all use a society size of 32. The GSoHCs all use $\mu = 0.06$[5].

4 Results

4.1 Experiment I

In our first experiment, our test suite consisted of 400 instances of randomly generated DisACSPs of the form <30,6,1.0,p2>. In this experiment, $p2$ took on values from the set {0.03, 0.04, 0.05, 0.06} for the 400 instances (100 instances for each class of DisACSP) of <30,6,1.0,p2>, where $\hat{p2}_{crit_A} \approx 0.06$. Each of the 30 agents randomly generated 29 arcs where each arc contained approximately 1.08, 1.44, 1.88, and 2.16 nogoods respectively for $p2$ values of 0.03, 0.04, 0.05, and 0.06. The arcs were generated according to a hybrid between Models A & B in (Macintyre et al. 1998). This method of constraint generation is as follows. If each arc was to have 1.08 nogoods (which is the case when $p2 = 0.03$) then every arc received at least 1 nogood and was randomly assigned an additional nogood with probability 0.08. Similarly, if the average number of nogoods needed for each constraint was 2.16, (which is the case when $p2 = 0.06$) then every constraint received at least 2 nogoods and was randomly assigned and additional nogood with probability 0.16. The probability that an arc existed was determined with probability $p1_\alpha$.

In this section, we compare SoHC and the GSoHCs on the 400 randomly generated DisACSPs described earlier. Table 1 presents the performance results of these four SoHCs. In Tables 1a-1d, the first column represents the algorithm, the second column represents the success rate of an algorithm when given a maximum of 2000 cycles to solve each of the 100 problems within a class, and the third column represents the average number of cycles needed to solve the problems within a class.

In Tables 1a-1d, one can see that the GSoHCs outperform the SoHC on each of the four classes of DisACSPs. This suggests that using restricted distributed crossover results in improved performance. For each of the GSoHCs, we developed a 'headless' version (HSoHC) and the GSoHCs all had a statistically significant better performance (Zhang, F. 2003). In the 'headless' form of the distributed restricted recombination operators, the best dCS is crossed with a randomly generated individual. The purpose of these 'headless' operators is to validate the effectiveness of operator. If a HSoHC using 'headless' recombination outperforms a similar GSoHC that uses distributed restricted recombination then one can conclude that the recombination operator is not an effective for

[5] In (Zhang, F. 2003), GSoHCs using distributed restricted forms of single-point, two-point, and uniform crossover were compared. The society sizes, ρ, and mutation rates, μ, for the GSoHCs were taken from the sets {2,4,8,16,32} and {0.0,0.03,0.06,0.12,0.25} respectively. The best overall society size and mutation rate for the GSoHCs was $\rho = 32$ and $\mu = 0.06$. Therefore we only show the results of the GSoHCs where $\rho = 32$ and $\mu = 0.06$.

Table 1. Performances on the <30,6,1.0,0.03>, <30,6,1.0,0.04> <30,6,1.0,0.05> and <30,6,1.0,0.06> DisACSPs

Alg.	SR	Cycles
SoHC	1.00	17.93
GSoHC$_{spx}$	1.00	14.41
GSoHC$_{tpx}$	1.00	14.00
GSoHC$_{mtpx}$	1.00	13.15
GSoHC$_{ux}$	1.00	14.32

(a) On the <30,6,1.0,0.03> DisACSPs

Alg.	SR	Cycles
SoHC	1.00	54.28
GSoHC$_{spx}$	1.00	31.01
GSoHC$_{tpx}$	1.00	30.35
GSoHC$_{mtpx}$	1.00	28.32
GSoHC$_{ux}$	1.00	31.29

(b) On the <30,6,1.0,0.04> DisACSPs

Alg.	SR	Cycles
SoHC	0.52	1323.80
GSoHC$_{spx}$	0.93	402.28
GSoHC$_{tpx}$	0.94	359.4
GSoHC$_{mtpx}$	0.98	273.62
GSoHC$_{ux}$	0.96	329.81

(c) On the <30,6,1.0,0.05> DisACSPs

Alg.	SR	Cycles
SoHC	0.02	1981.06
GSoHC$_{spx}$	0.12	1861.59
GSoHC$_{tpx}$	0.10	1880.95
GSoHC$_{mtpx}$	0.11	1901.72
GSoHC$_{ux}$	0.09	1881.65

(d) On the <30,6,1.0,0.06> DisACSPs

the types of problems within the test suite and that actually macromutation is responsible for the performance improve over the simple SoHC (Jones 1995).

Notice also that GSoHC$_{mtpx}$ has the best performance in term of SR and Cycles for all classes of DisACSPs except for when $p2 = 0.06$. It seems that by taking 67% of the genes from the best individual in the population improves search performance on the easier classes of DisACSPs. However, at the phase transitions this performance improvement disappears. It would be interesting to see how a biased uniform crossover operator would perform on this test suite. Instead of taking values from the best individual for 50% of the genes, it would be interesting to see if a bias of 67% would improve performance the way that the modified two-point crossover did. It would also be interesting to see if a bias of less than 50% (but not 0%) would improve the performance on the DisACSPs located at the phase transition.

4.2 Discussion

The increased performance of the GSoHCs over SoHC is primarily due to the way in which their operators intensify search around the current best individual in the population. The basic assumption made by anyone applying an EC to a problem is that optimal (or near optimal) solutions are surrounded by good solutions. However, this assumption is not always true for constrained problems. Even for problems where this is the case ECs typically employ local search in an effort exploit promising regions. Thus, the EC will intensify search periodically in some region. Actually, the search behavior of the GSoHCs is no different. The hill-climbers associated with individuals that are involved in a below average number of conflicts are allowed to continue their search (via distributed hill-climbing) while hill-climbers associated with individuals that are involved in an above average number of conflicts have their associated individuals replaced by offspring that more closely resemble the current best individual in the population.

5 Conclusions

In this paper, we have introduced the concept of DisACSPs and have demonstrated how distributed restricted operators can be used to improve the search of a society of hill-climbers on easy and difficult DisACSPs. We also provided a brief discussion of some of the reasons why the performances of the GSoHCs were so dramatically superior to SoHC. We also showed that a modified version of two-point crossover outperforms two-point crossover (and all other recombination operators) on DisACSPs that are near the phase transition. We discussed how this result may lead to the development of a biased dRUC operator. Perhaps the bias may be adapted to the problem type. For easy problems the bias should be higher than 0.5 and for harder problems the bias should be lower than 0.5.

Acknowledgements. This research was supported by the National Science Foundation under grant #IIS-9907377.

References

Bäck, T., Hammel, U., and Schwefel, H.-P. (1997). "Evolutionary Computation: Comments on the History and Current State", *IEEE Transactions on Evolutionary Computation*, 1:1-23, IEEE Press.

Bejar, R., Krishnamachari, B., Gomes, C., and Selman, B. (2001). "Distributed Constraint Satisfaction in a Wireless Sensor Tracking System", *Proceedings of the IJCAI-2001 Workshop on Distributed Constraint Reasoning*, pp. 81-90.

Bowen, J. and Dozier, G. (1996). "Constraint Satisfaction Using A Hybrid Evolutionary Hill-Climbing Algorithm That Performs Opportunistic Arc and Path Revision," *Proceedings of AAAI-96*, pp. 326-331, AAAI Press / The MIT Press.

Calisti, M., and Faltings, B. (2001). "Agent-Based Negotiations for Multi-Provider Interactions", *Journal of Autonomous Agents and Multi-Agent Systems*.

Dozier, G. (2002). "Solving Distributed Asymmetric CSP Using via a Society of Hill-Climbers", *Proceedings of the 2002 International Conference on Artificial Intelligence*, pp. 949-953, June 24-27, Las Vegas, NV, CSREA.

Dozier, G., Bowen, J. and Homaifar, A. (1998). "Solving Constraint Satisfaction Problems Using Hybrid Evolutionary Search", *IEEE Transactions on Evolutionary Computation*, Vol. 2, No. 1, pp. 23-33, April 1998, Institute of Electrical & Electronics Engineers.

Fogel, D. B. (2000). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, 2nd Edition, IEEE Press.

Freuder, E. C., Minca, M., and Wallace, R. J. (2001). “Privacy/Efficiency Tradeoffs in Distributed Meeting Scheduling by Constraint-Based Agents”, *Proceedings of the IJCAI-2001 Workshop on Distributed Constraint Reasoning*, pp. 63-71.

Jones, T. (1995). “Crossover, Macromutation and Population-based Search”, *Proceedings of The Sixth International Conference (ICGA 1995)* pp. 73-80, Morgan Kaufmann.

MacIntyre, E., Prosser, P., Smith, B., and Walsh, T. (1998). “Random Constraint Satisfaction: Theory Meets Practice,” *The Proceedings of the 4th International Conference on Principles and Practices of Constraint Programming (CP-98)*, pp. 325-339, Springer-Verlag.

Mackworth, A. K. (1977). “Consistency in networks of relations”. *Artificial Intelligence*, 8 (1), pp. 99-118.

Modi, P. J., Jung, H., Tambe, M., Shen, W.-M., and Kulkarni, S. (2001). “Dynamic Distributed Resource Allocation: A Distributed Constraint Satisfaction Approach” *Proceedings of the IJCAI-2001 Workshop on Distributed Constraint Reasoning*, pp. 73-79.

Morris, P. (1993). “The Breakout Method for Escaping From Local Minima,” *Proceedings of AAAI’93*, pp. 40-45.

Sebag, M. and Shoenauer, M. (1997). “A Society of Hill-Climbers,” *The Proceedings of the 1997 International Conference on Evolutionary Computation*, pp. 319-324, IEEE Press.

Silaghi, M.-C., Sam-Haroud, D., Calisti, M., and Faltings, B. (2001). “Generalized English Auctions by Relaxation in Dynamic Distributed CSPs with Private Constraints”, *Proceedings of the IJCAI-2001 Workshop on Distributed Constraint Reasoning*, pp. 45-54.

Smith, B. (1994). “Phase Transition and the Mushy Region in Constraint Satisfaction Problems,” *Proceedings of the 11th European Conference on Artificial Intelligence*, A. Cohn Ed., pp. 100-104, John Wiley & Sons, Ltd.

Solnon, C. (2002). “Ants can solve constraint satisfaction problems”, *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 4, pp. 347-357, August, IEEE Press.

Tsang, E. (1993). *Foundations of Constraint Satisfaction*, Academic Press, Ltd.

Yokoo, M. (2001). Distributed Constraint Satisfaction, Springer-Verlag Berlin Heidelberg.

Zhang, F. (2003). “A Comparison of Distributed Restricted Recombination Operators for Genetic Societies of Hill-Climbers: a DisACSP Perspective,” *Auburn University Masters Thesis*, Department of Computer Science & Software Engineering.

Zhang, X. and Xing, Z. (2002). “Distributed Breakout vs. Distributed Stochastic: A Comparative Evaluation on Scan Scheduling,” *AAMA-02 Third International Workshop on Distributed Constraint Reasoning*, July 16, 2002, Bologna, Italy, pp.192-201.