Solving Distributed Asymmetric Constraint Satisfaction Problems Using an Evolutionary Society of Hill-Climbers

Gerry Dozier

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

Abstract. The distributed constraint satisfaction problem (DisCSP) 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. The objective in solving a DisCSP is to allow the agents in A to develop a consistent distributed solution by means of message passing. In this paper, we present an evolutionary society of hill-climbers (ESoHC) that outperforms a previously developed algorithm for solving randomly generated DisCSPs that are composed of asymmetric constraints on a test suite of 2,800 distributed asymmetric constraint satisfaction problems.

1 Introduction

A DisCSP [17] 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 [17]. 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 [17].

Many real world problems have been modeled and solved using DisCSPs [1, 2,3,5,6,12]; however, many of these models use mirrored (symmetric) inter-agent constraints. Since these inter-agent constraints are known by the agents involved in the constraint, they cannot be regarded as private. If these constraints were

E. Cantú-Paz et al. (Eds.): GECCO 2003, LNCS 2723, pp. 561–572, 2003.

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 [4,5,12], little research has been done on distributed asymmetric CSPs (DisACSPs).

In this paper, we demonstrate how a distributed restricted form of uniform mutation can be used to improve the effectiveness of a previously developed evolutionary computation (EC) for solving DisACSPs known as a society of hill-climbers (SoHC) [4]. We refer to this new algorithm as an evolutionary SoHC (ESoHC). Our results show that ESoHC outperforms SoHC on a test suite of 2,800 DisACSPs.

The remainder of this paper is organized as follows. In Section 2, we present an overview of constraint processing which includes an introduction to the concept of asymmetric constraints and presents a formula for predicting where the most difficult randomly generated asymmetric CSPs are located, known as the phase transition [4,7,13]. In Section 3, we introduce the SoHC concept and explain how our ESoHC operates. In Section 4, we present the results of applying SoHC and ESoHC to 800 randomly generated distributed DisACSPs. In this section, we also compare SoHC and ESoHC on an additional 2,000 randomly generated DisACSPs in order to better visualize their performance across the phase transition. In Section 5, we present our conclusions and future work.

2 CSPs, Asymmetric Constraints, and the Phase Transition

A CSP [15] can be viewed as 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 values of exactly two variables)[1] $\langle X, D, C\rangle$ where $X = \{E, F, G\}$, $D = \{d_E = \{e_1, e_2, e_3\}$, $d_F = \{f_1, f_2, f_3\}$, $d_G = \{g_1, g_2, g_3\}\}$, and $C = \{c_{EF}, c_{EG}, c_{FG}\}$. Suppose that the constraints c_{EF}, c_{EG}, c_{FG} are as follows:

$c_{EF} = \{\langle$ `e1,f2` $\rangle, \langle$ `e1,f3` $\rangle, \langle$ `e2,f2` $\rangle, \langle$ `e3,f2` $\rangle\}$
$c_{EG} = \{\langle$ `e2,g3` $\rangle, \langle$ `e3,g1` $\rangle\}$
$c_{FG} = \{\langle$ `f2,g1` $\rangle, \langle$ `f2,g3` $\rangle\}$.

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 that involves more than two variables can be transformed into a set of binary constraints [15].

2.1 Asymmetric Constraints

Constraints in a binary constraint network may also be represented as two directional constraints referred to as arcs [8,15]. 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,$p1$,$p2$) [13] 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 [13] 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 [13] 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 [4] 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) [4,11] is a collection of hill-climbers that search in parallel and 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 [10]. Before presenting the society of hill-climbers concept, we must first discuss the distributed hill-climber that makes up the algorithm. In this section, we first introduce a modified version of Yokoo's distributed breakout algorithm with broadcasting [17] (mDBA) which is based on Morris' Breakout Algorithm [10]. After introducing 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 exceptions 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 [17]. 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.

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

[2] A nogood is a tuple that causes a conflict.

[3] In this paper, $Neighbor_i = A - \{a_i\}$.

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 a 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 Evolutionary SoHCs

The SoHCs reported in this paper are based on mDBA. Each 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. Figure 2 provides a simplified view of a simple SoHC. Notice in Figure 2 that 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.

The ESoHC works exactly like the SoHC mentioned earlier except for on each cycle a distributed restricted uniform mutation operator is applied as follows. Each distributed candidate solution, dCS_j, that has an above average number of conflicts is replaced with an offspring that is a mutated version of the best individual, dCS_q, as follows. Given a distributed individual, dCS_k, that is involved in an above average number of conflicts, with probability μ, agent a_i will randomly assign v_{ik} a value from d_i and with probability $1 - \mu$ agent a_i will set $v_{ik} = v_{iq}$. Of course, μ is referred to as the mutation rate. We refer to this form of mutation as distributed restricted uniform mutation (dRUM-μ).

[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 to 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 ∈ 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 γ_i the number conflicts that (x_i = v_i)
                is involved in;
      Step 4: randomly search for value b_i ∈ d_i that minimizes the
                number of conflicts of x_i (ties broken randomly),
      Step 5: let r_i be the number of conflicts resolved
                by (x_i = b_i);
      Step 6: broadcast γ_i and r_i to other agents;
      Step 7: receive γ_j and r_j from other agents, let f_i = Σ γ_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))†
                v_i = b_i;
    } while (f_i > 0)
}
        † Ties are broken with randomly with a synchronized tie-breaker.
```

Fig. 1. The mDBA Protocol

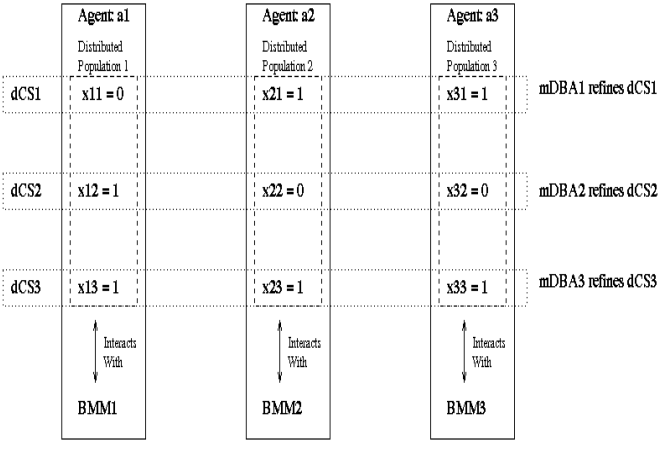


Fig. 2. An Simplified View of a SoHC with a Society Size of 3 dCSs

4 Results

4.1 Experiment I

In our first experiment, our test suite consisted of 800 instances of randomly generated DisACSPs of the form <30,6,1.0,p2> and <30,6,$p1_\alpha$,0.098>. In this experiment, $p2$ took on values from the set, {0.03, 0.04, 0.05, 0.06}, for the 400 instances of <30,6,1.0,p2>, where $\hat{p2}_{crit_A} \approx 0.06$ and $p1_\alpha$ took on values from the set, {0.3, 0.4, 0.5, 0.6}, for the 400 instances of <30,6,$p1_\alpha$,0.098>, where instances of <30, 6, $p1_\alpha = 0.6$, 0.098> were at the phase transition. Each of the 30 agents randomly generated 29 arcs where each arc contained approximately 1.08, 1.44, 1.88, 2.16, and 3.53 nogoods respectively for $p2$ values of 0.03, 0.04, 0.05, 0.06, and 0.098. The arcs were generated according to a hybrid between Models A & B in [7]. 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 $p1_\alpha$.

Tables 1a-1d show the performance results of applying eight algorithms on each of the 100 instances of the <30,6,1.0,p2> classes of DisACSPs. The eight algorithms compared are mDBA-DTB, (the mDBA that uses Yokoo's deterministic tie-breaker algorithm to break ties between agents with the highest r_i value), six SoHCs with values of ρ taken from the set {1, 2, 4, 8, 16, 32}, and an ESoHC with $\rho = 32$ (ESoHC-32) that used dRUM-0.12.

In each table, the first column represents the algorithm. The second column represents the success rate (SR) that an algorithm had in finding a solution on the 100 problems when allowed a maximum of 2000 cycles. In the third column, the average number of cycles per run is recorded, and in the fourth column, the average number of constraint checks made by the algorithm is recorded.

When comparing the SoHCs, one can see that the larger the society size the better the performance is with respect to SR and average number of cycles. However using larger society sizes also results in an increased number of constraint checks and larger message sizes. In Table 1d, one can see that the classes of DisACSPs where $p2 = 0.06$ contains the most difficult problems.

It is important to realize that when comparing DisCSP-solvers that the most important criterion is success rate followed by communication cycles followed by the total number of constraint checks. For this reason, SoHC-32 has been selected as the overall best SoHC for the fully connected DisACSPs.

To make this point clearer, consider the communication medium used by the agents to be a network. With this being the case, we can compute the utilization of a link between any two agents given a normal 20 byte internet protocol (IP) packet header [16]. SoHC-01 will utilize $\frac{1}{20+1}$ of the bandwidth while SoHC-32

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	Checks
mDBA-DTB	0.74	563.35	623849
SoHC-01	0.83	384.49	450684
SoHC-02	0.95	139.29	379128
SoHC-04	1.00	28.20	238510
SoHC-08	1.00	24.34	441775
SoHC-16	1.00	20.47	790013
SoHC-32	1.00	17.93	1448802
ESoHC-32	1.00	12.78	916539

(a) Performances on <30,6,1.0,0.03>

Alg.	SR	Cycles	Checks
mDBA-DTB	0.60	989.30	1433954
SoHC-01	0.71	928.94	1422377
SoHC-02	0.93	444.35	1423055
SoHC-04	0.94	313.52	2122349
SoHC-08	1.00	124.22	1915596
SoHC-16	1.00	72.62	2456001
SoHC-32	1.00	54.28	3909942
ESoHC-32	1.00	26.09	1830178

(b) Performances on <30,6,1.0,0.04>

Alg.	SR	Cycles	Checks
mDBA-DTB	0.07	1917.75	4218533
SoHC-01	0.10	1845.13	4149324
SoHC-02	0.11	1856.94	8466398
SoHC-04	0.17	1829.86	16838690
SoHC-08	0.22	1694.59	31591364
SoHC-16	0.36	1548.81	58666291
SoHC-32	0.52	1323.80	101607214
ESoHC-32	0.98	263.20	16101363

(c) Performances on <30,6,1.0,0.05>

Alg.	SR	Cycles	Checks
mDBA-DTB	0.00	2000.00	5529200
SoHC-01	0.00	2000.00	5573943
SoHC-02	0.00	2000.00	11252052
SoHC-04	0.00	2000.00	22665737
SoHC-08	0.00	2000.00	45861472
SoHC-16	0.01	1982.02	91767502
SoHC-32	0.02	1981.06	184858509
ESoHC-32	0.11	1849.02	134172949

(d) Performances on <30,6,1.0,0.06>

will utilize $\frac{32}{20+32}$ of the bandwidth. For this reason alone it is a welcomed result to see that larger society sizes lead to better performance results.

In Tables 1a-1d, the results also suggest that breaking ties randomly is more effective than breaking ties using Yokoo's deterministic tie-breaking method. This can be seen by comparing the success rates (SR) of mDBA-DTB and SoHC-01. Notice that in Tables 1a-1c that SoHC-01 has a slightly higher success rate.

When comparing SoHC-32 and ESoHC-32 in Tables 1a-1d, one can see that ESoHC-32 has the better performance on each of the four class of DisACSPs. As the tightness is increased from 0.03 to the critical value of 0.06, the difference in the performance as compared with SoHC-32 becomes more pronounced. At the phase transition, the SR of ESoHC-32 is $5\frac{1}{2}$ times better than the SR for SoHC-32.

Tables 2a-d show the performances of the eight algorithms on 100 instances of the <30,6,$p1_\alpha$,0.098> classes of DisACSPs. Notice that as $p1_\alpha$ is increased, in Tables 2a-2d, that the success rates of the algorithms decreases. Notice once again that the hardest DisACSPs seem to be located at the predicted phase transition, <30,6,0.6,0.098>. Also in Table 2, one can see that mDBA-DTB outperforms SoHC-01 on the <30,6,0.3,0.098> class of DisACSPs but loses to SoHC-01 on the <30,6,0.4,0.098> and <30,6,0.5,0.098> classes. When compar-

Table 2. Performances on the <30,6,$p1_\alpha$.3,0.098> DisACSPs

Alg.	SR	Cycles	Checks
mDBA-DTB	0.64	758.98	896922
SoHC-01	0.63	790.50	904392
SoHC-02	0.97	307.26	735397
SoHC-04	1.00	36.37	280431
SoHC-08	1.00	30.57	490479
SoHC-16	1.00	21.53	791964
SoHC-32	1.00	18.20	1422854
ESoHC-32	1.00	12.77	894213

(a) Performances on <30,6,0.3,0.098>

Alg.	SR	Cycles	Checks
mDBA-DTB	0.47	1197.23	1784095
SoHC-01	0.51	1163.29	1757059
SoHC-02	0.67	841.67	2534178
SoHC-04	0.89	450.68	2954802
SoHC-08	0.97	172.90	2461325
SoHC-16	1.00	129.86	4031357
SoHC-32	1.00	60.12	4184293
ESoHC-32	1.00	29.83	2009218

(b) Performances on <30,6,0.4,0.098>

Alg.	SR	Cycles	Checks
mDBA-DTB	0.05	1960.59	4177278
SoHC-01	0.10	1864.31	3991908
SoHC-02	0.12	1859.26	8124593
SoHC-04	0.24	1700.12	14848626
SoHC-08	0.34	1600.72	28663041
SoHC-16	0.50	1432.35	52936951
SoHC-32	0.56	1154.75	85608964
ESoHC-32	0.96	279.36	16493167

(c) Performances on <30,6,0.5,0.098>

Alg.	SR	Cycles	Checks
mDBA-DTB	0.00	2000.00	5329815
SoHC-01	0.00	2000.00	5459918
SoHC-02	0.01	1987.72	10844644
SoHC-04	0.00	2000.00	22344655
SoHC-08	0.02	1977.10	44391268
SoHC-16	0.01	1983.21	89557109
SoHC-32	0.03	1974.31	180966004
ESoHC-32	0.17	1755.67	126561756

(d) Performances on <30,6,0.6,0.098>

ing SoHC-32 and ESoHC-32, the results are similar to what has been observed earlier. ESoHC-32 outperforms SoHC-32 on all classes of DisACSPs and has an SR that is $5\frac{2}{3}$ times greater than SoHC on the <30,6,0.6,0.098> class.

4.2 Experiment II

In the previous section, we presented the results of applying eight SoHCs to 800 randomly generated DisACSPs. In that presentation, we were only able to show the side of the phase transition where classes were likely to contain at least one solution [7,13]. This was done because the distributed hill-climbers presented in this paper are not complete; they cannot determine if the problem at hand has no solution at all. In order to visualize the phase transition in DisACSPs we used a technique introduced by Solnon in [14]. This technique is simple. When randomly generating a CSP make sure that it has at least one solution. Using the above approach will allow incomplete search algorithms to experience the easy-hard-easy behavior across the phase transition as tightness and/or density is increased from 0.0 to 1.0.

In order to visualize the phase transition for the <30,6,1.0,p2> classes of DisACSPs we randomly generated an additional 1,100 DisACSPs where p2 took values from the set, {0.03, 0.04, 0.05, 0.06, 0.065, 0.07, 0.08, 0.09, 0.1, 0.17, 0.25}.

For each value of p2, 100 DisACSPs were generated and guaranteed to have at least 1 solution. SoHC-32 and ESoHC-32 were then run on each of the problems with a maximum number of 2000 cycles in which to find a solution.

Figure 3 shows the performance results of SoHC-32 and ESoHC-32 on the 1,100 DisACSPs of the class <30,6,1.0,p2>. The range of values of p2 (from 0.03 to 0.25) can be seen along the x-axis. Figure 3a shows the results in terms of the average number of cycles needed solve a DisACSP (shown on the y-axis) and Figure 3b shows the performance results in terms of failure rate (as shown on the y-axis).

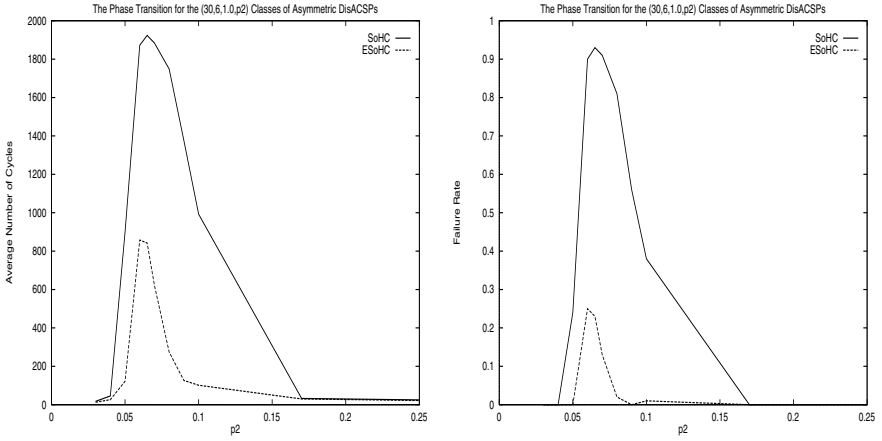
In Figure 3a, one can see that the average number of cycles increases rapidly as $p2$ is increased from 0.03 to about 0.065. For these problems, the actual phase transition seems to occur at $p2 = 0.065$. As $p2$ is increased beyond 0.065 one can see a rapid reduction in average number of cycles needed to find a solution. As the constraints become increasingly tighter, it becomes easier for SoHC-32 and ESoHC-32 to find the one and only solution. The shape of the curve in Figure 3b is very similar to the one shown in Figure 3a. Notice also that the performance of ESoHC-32 is superior to SoHC-32 over the total range of values for $p2$.

In order to visualize the phase transition as $p1_\alpha$ is increased, we created 900 randomly generated DisACSPs of the form <30,6,$p1_\alpha$,0.098> where the arc density, $p1_\alpha$, took on values from the set {0.3, 0.4, 0.5, 0.6, 0.62, 0.7, 0.8, 0.9, 1.0 }. Once again for each value of $p1_\alpha$ 100 DisACSPs were randomly generated, and the SoHCs were run on each problem with an allowed a maximum of 2000 cycles to find a solution.

Figure 4 shows the search behavior of SoHC-32 and ESoHC-32 as $p1_\alpha$ was increased from 0.3 to 1.0 in terms of the average number of cycles needed to find a solution as well as the failure rate. The results are similar to those shown in Figure 3; ESoHC-32 dramatically outperforms SoHC-32. However, Figures 3 and 4 differ in that the easy-hard-easy transition is less abrupt. The reason for this is that constraint tightness is a more sensitive predictor of the relative hardness of a CSP.

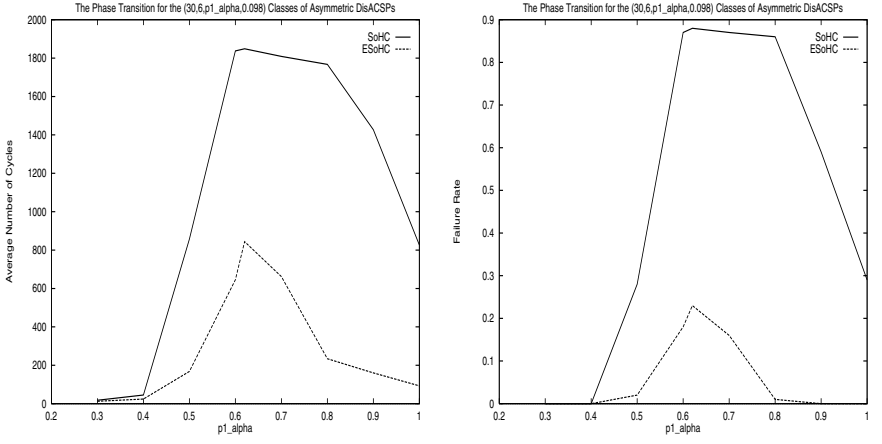
4.3 Discussion

The increased performance of ESoHC over SoHC is primarily due to the way in which the dRUM-μ operator intensifies 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 true for constrained problems. Even for problems where this is the case ECs typically employ local search in an effort to exploit promising regions. Thus, the EC will intensify search periodically in some region. Actually, the search behavior of ESoHC is no different. The individuals that are involved in a below average number of conflicts are allowed to continue to be refined by mDBA while individuals that are involved in an above average number of conflicts are replaced by offspring that more closely resemble the current best individual in the population. Upon closer inspection of the results in Tables 1 and 2 one can see that as ρ is increased in the SoHCs the



(a) Avg. Number of Cycles (b) Failure Rate

Fig. 3. Phase Transition Based on Avg. Number of Cycles and Failure Rate



(a) Avg. Number of Cycles (b) Failure Rate

Fig. 4. Phase Transition Based on Avg. Number of Cycles and Failure Rate

performance gain diminishes. Therefore it seems reasonable, given a sufficiently large ρ, that half of the individuals can be used to intensify search without adversely affecting the convergence rate.

5 Conclusions and Future Work

In this paper, we have introduced the concept of DisACSPs and have demonstrated how distributed restricted uniform mutation 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 performance of ESoHC-32 is superior to SoHC-32. Our future work will include the development of other

distributed forms procreation that may increase the performance of ESoHC as well as the study of effect that different reallocation strategies have on the performance of ESoHC.

Acknowledgement. The author would like to thank the National Science Foundation for the support of this research under grant #IIS-9907377.

References

1. Bejar, R., Krishnamachari, B., Gomes, C., and Selman, B. (2001). "Distributed Constraint Satisfaction in a Wireless Sensor Tracking System", *Proc. of the IJCAI Workshop on Distributed Constraint Reasoning*, pp. 81–90.
2. Calisti, M., and Faltings, B. (2000). "Agent-Based Negotiations for Multi-Provider Interactions", *Proc. of the Intl. Sym. on Agent Sys. and Applications*, pp. 235–248.
3. Calisti, M., and Faltings, B. (2000). "Distributed Constrained Agents for Allocating Service Demands in Multi-Provider Networks", *Journal of the Italian Operational Society*, Special Issue on Constraint Problem Solving, vol. XXIX, no. 91.
4. Dozier, G. and Rupela, V. (2002). "Solving Distributed Asymmetric CSPs via a Society of Hill-Climbers", *Proc. of IC-AI'02*, pp. 949–953, CSREA Press.
5. Freuder, E. C., Minca, M., and Wallace, R. J. (2001). "Privacy/Efficiency Tradeoffs in Distributed Meeting Scheduling by Constraint-Based Agents", *Proc. of the IJCAI Workshop on Distributed Constraint Reasoning*, pp. 63–71.
6. Krishnamachari, B., Bejar, R., and Wicker, S. (2002). "Distributed Problem Solving and the Boundaries of Self-Configuration in Multi-hop Wireless Networks", *Proc. of the Hawaii Intl. Conference on System Sciences*, HICSS-35.
7. MacIntyre, E., Prosser, P., Smith, B., and Walsh, T. (1998). "Random Constraint Satisfaction: Theory Meets Practice," *The Proc. of CP-98*, pp. 325–339.
8. Mackworth, A. K. (1977). "Consistency in networks of relations". *Artificial Intelligence*, 8 (1), pp. 99–118.
9. Modi, P. J., Jung, H., Tambe, M., Shen, W.-M., and Kulkarni, S. (2001). "Dynamic Distributed Resource Allocation: A Distributed Constraint Satisfaction Approach" *Proc. of the IJCAI Workshop on Distributed Constraint Reasoning*, pp. 73–79.
10. Morris, P. (1993). "The Breakout Method for Escaping From Local Minima," *Proc. of AAAI'93*, pp. 40–45.
11. Sebag, M. and Shoenauer, M. (1997). "A Society of Hill-Climbers," *The Proc. of ICEC-97*, pp. 319–324, IEEE Press.
12. Silaghi, M.-C., Sam-Haroud, D., Calisti, M., and Faltings, B. (2001). "Generalized English Auctions by Relaxation in Dynamic Distributed CSPs with Private Constraints", *Proc. of the IJCAI Workshop on Distributed Constraint Reasoning*, pp. 45–54.
13. Smith, B. (1994). "Phase Transition and the Mushy Region in Constraint Satisfaction Problems," *Proc. of ECAI-94*, pp. 100–104, John Wiley & Sons, Ltd.
14. Solnon, C. (2002). "Ants can solve constraint satisfaction problems", to appear in: *IEEE Transactions on Evolutionary Computation*, IEEE Press.
15. Tsang, E. (1993). *Foundations of Constraint Satisfaction*, Academic Press, Ltd.
16. Walrand, J. (1998). *Communication Networks: A First Course*, 2nd Edition, WCB/McGraw-Hill.
17. Yokoo, M. (2001). *Distributed Constraint Satisfaction*, Springer-Verlag.