2006 IEEE Congress on Evolutionary Computation
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

A Comparison of Evolutionary Protocols for Solving Distributed Constraint Satisfaction Problems

Winard R. Britt, Hurley D. Cunningham, and Gerry V. Dozier

Abstract— **Evolutionary Computation (EC) is the field of study devoted to problem solving using simulated evolution. In this paper, evolutionary operators are applied and a number of societies of hill-climbers (SoHCs), such as a genetic SoHC (GSoHC) and an evolutionary SoHC (ESoHC) are employed for solving randomly generated distributed asymmetric constraint satisfaction problems (DisACSPs). Further, we develop an Estimation of Distribution Algorithm SoHC (EDA-SoHC) variant using a uniform mutation operator. This variant produces offspring by drawing genetic material from a distribution of above-average individuals in the population.**

In this paper, we compare GSoHCs using distributed restricted forms of single-point, two-point, modified two-point, and uniform crossover. The GSoHCs are also compared with an ESoHC that uses a distributed restricted form of uniform mutation and a simple SoHC which does not use any evolutionary operators. Finally, we compare the SoHC, GSoHCs, and ESoHC to the EDA-SoHC.

I. Introduction

Evolutionary Computation [1]-[11] 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, machine learning, constrained optimization, and constraint satisfaction just to name a few [8]-[10],[12]-[15].

In the Evolutionary Constraint Satisfaction community, we have been a migration away from pure evolutionary computations for constraint satisfaction [16] towards the hybridization of ECs with traditional CSP techniques [17]-[23] and /or the incorporation of heuristics and problem specific knowledge [18]-[20] ,[24]–[28] 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) [29]-[33].

A DisCSP [29]-[37] 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 [34], [38]. 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 [34].

Many real world problems have been modeled and solved using DisCSPs [38]-[44]; 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 [42] and [44], little research has been done on distributed asymmetric CSPs (DisACSPs).

A simple example of a DisACSP might be attempting to schedule a meeting where a large number of the participants did not wish to reveal their personal time obligations (for security reasons, perhaps). Groups of individuals might try to resolve their time constraints with their coleagues (or "neighbors") until an actual meeting time could be determined. A similar idea is illustrated in [42].

In this paper, we demonstrate how distributed restricted forms of uniform, single-point and two-point crossover can be used for solving DisACSPs known as a society of hill-climbers (SoHC) [29]-[33]. We refer to these algorithms as genetic SoHCs (GSoHCs). We also demonstrate how distributed restricted form of mutation can be used for solving DisACSPs, we refer to this algorithm as evolutionary SoHCs (ESoHC). Further, we show how an Estimation of Distribution Algorithm based SoHC can be utilized, we refer to it as an EDA-SoHC. Our results show

Manuscript received January 31, 2006. This research was supported by the National Science Foundation under grant #IIS-9907377.

Winard R. Britt is with the Computer Science and Software Engineering Department of Auburn University, Auburn, AL 36849. Phone: 334-546-8273; e-mail: brittwr@auburn.edu).
Hurley D. Cunningham is with the Computer Science and Software Engineering Department of Auburn University, Auburn, AL 36849; e-mail: cunnihu@auburn.edu).
Gerry V. Dozier is with the Computer Science and Software Engineering Department of Auburn University, Auburn, AL 36849; e-mail: doziegv@auburn.edu).

the three forms of GSoHCs have similar performance. ESoHC and EDA-SoHC both have better performance than GSoHC and SoHC in our test suite.

The reminder of this paper is organized as follows. In section II, we present an overview of constraint processing which includes an introduction to the concept of asymmetric constraints and present a formula for predicting where the most difficult randomly generated asymmetric CSPs are located. In section III, we introduce the SoHC concept and explain how GSoHCs, ESoHCs, and EDA-SoHCs operate. In section IV, we present the results of applying SoHC, GSoHC, ESoHC, and EDA-SoHC on 400 randomly generated distributed DisACSPs. In this section, we also compare the three SoHCs on an additional 400 randomly generated distributed DisACSPs. In section V, we provide conclusions and some guidance for future work.

II. Asymmetric Constraint Satisfaction

A CSP [6],[19],[30],[45]-[48] can be viewed as triple $\langle X,D,C\rangle$ where X is set of variables, D is set of domains where each xi$\in$X takes its value from the corresponding domain di$\in$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] $\langle X,D,C\rangle$ where X = {E, F, G}, D = {dE = {e1, e2,e3}, dF = {f1, f2,f3}, dG = {g1, g2, g3}}, and C = {cEF,cEG,cFG}. Suppose that the constraints cEF, cEG, cFG 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 di x dj. 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.

Constraints in a binary constraint network may also be represented as two directional constraints referred to as arcs [46],[48]. For example, the symmetric constraint c_{EF} can be represented as

$c_{EF}=\{c_{\vec{EF}},c_{\overleftarrow{EF}}\}$ where

$c_{\vec{EF}}=c_{\overleftarrow{EF}}=\{\langle e1,f2\rangle,\langle e1,f3\rangle,\langle e2,f2\rangle,\langle e3,f2\rangle\}$,

where $c_{\vec{EF}}$ 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 $c_{\vec{EF}}\neq c_{\overleftarrow{EF}}$.

[1] In this paper, we only consider binary constraint networks because any constraint involving more than one variable can be transformed into a set of binary constraints [45]

III. Society of Hill-Climbers

A society of hill-climbers (SoHC) [5],[6],[29]-[32], [34], [49] 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 [50]. Before presenting the SoHC, 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 [34] (mDBA) which is based on Morris' Breakout Algorithm [50]. 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 $C_{\vec{x_ix_j}}$ where $i \neq j$. The objective of agent a_i is to satisfy all of its constraints $C_{\vec{x_ix_j}}$. Each agent also maintains a breakout management mechanism (BMM) [29]-[32] 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.

A. The mDBA

The mDBA (shown in Figure 3) 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).

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

```
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 +
        breakout_elements($x_i = v_i$, $x_j = j_i$) 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 +
        breakout_elements($x_i = v_i$, $x_j = v_j$) 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 $f = \Sigma conflicts_k$;
    Step 8: if (max($r_k$) == 0)
    for each conflict, ($x_i = v_i$, $x_j = w_j$)
    update_breakout_elements($BMM_i$,
    ($x_i = v_i$, $x_j = w_j$));
    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.3. mDBA Agent Protocol

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 [34]. 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 *conflicts$_i$*.

Once the number of conflicts, *conflicts$_i$*, 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 *conflicts$_i$* and r_i have been computed, agent a_i broadcasts these values to each of its neighbors.

When an agent receives the *conflicts$_j$* and r_j values from each of its neighbors, it sums up all *conflicts$_j$* including *conflicts$_i$* 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.

B. *The Genetic and Evolutionary Protocols*

The SoHCs reported in this paper are based on the mDBA. Each SoHC runs *s* mDBA hill-climbers in parallel, where *s* is the society size. Each of the *s* hill-climbers communicates with each other indirectly throught a distributed BMM [6].

A GSoHC [31], [33] is a SoHC that applies a genetic operator [1], [5], [7]-[11], [33], [51]-[54] on each cycle. The genetic operator could be any kind of crossover, such as single-point, two-point, multi-point and uniform crossover.

1) *Distributed Restricted Uniform Crossover*

The operator of the GSoHC that Dozier employed in [31] is a distributed restricted uniform crossover. We denote a GSoHC that uses this kind of crossover as GSoHC$_{ux}$. On each cycle, GSoHC$_{ux}$ applies a distributed restricted uniform crossover operator, as follows. Each distributed candidate solution with an above average number of conflicts, dCS$_j$, is replaced with an offspring that is a recombination of the best individual, dCS$_{best}$, and dCS$_j$. An agent a_i will assign v_{ij} the value from v_{iq} with probability (1-μ)/2 and will leave v_{ij} unchanged with probability (1-μ)/2, with μ being a mutation probability. With probability μ agent a_i will randomly assign v_{ij} a value from d$_i$, the domain of values for variable x_i.

[3] In this paper, $Neighbor_i = A - \{a_i\}$. This effectively implies that there are effective *constraints among all agents.*

[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 to change its current value assignment. We refer to this as the deterministic tie-breaker (DTB) method

2) *Distributed Restricted Two-Point Crossover and Distributed Restricted Modified Two-Point Crossover*

Similarly, when a distributed restricted two-point crossover [33] is applied on each cycle, each dCS$_i$ that has an above average number of conflicts is also replaced with an offspring. To create the offspring, the first part is the same as single point crossover. With probability μ, the offspring will randomly select a value from the domain of the values. Then, with probability 1-μ, two cut points are randomly chosen with duplicates from 1 to N-1, where N is the number of agents of an individual, and the cut points are sorted. The dCS$_i$ and dCS$_{best}$ are cut into parts at these two points. With probability (1-μ)/2, agents with addresses less than or equal to the first cut point or greater than the second one will take the values of dCS$_{best}$, others will keep their values. With probability of the other (1-μ)/2, agents with addresses greater than the first cut point and less than or equal to the second one will take the values of dCS$_{best}$, while others will remain unchanged. We call this kind of recombination a distributed restricted two-point crossover and refer it as GSoHC$_{tpx}$ [33].

In order to improve the performance of two-point crossover, we increase the probability for taking genes from the dCS$_{best}$. The difference between the GSoHC using the improved two-point crossover with GSoHC$_{tpx}$ is as follows. With probability 1-μ, agents with addresses less than or equal to the first cut point or greater than the second cut point will take the values of dCS$_{best}$, others will keep their values. This way approximately 67% of an offspring's genes will come from dCS$_{best}$. The distributed restricted version of two-point crossover takes approximately 50% of the genes from dCS$_{best}$. We denote the GSoHC that uses this modifed two-point crossover GSoHC$_{mtpx}$.

3) *Evolutionary SoHC*

The Evolutionary SoHC [32] is similar to GSoHC mentioned earlier except it relies on mutation exclusively. We denote this as ESoHC. On each cycle, the distributed restricted mutation 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 mutated version of the best individual, dCS$_{best}$, as follows. With probability μ, agent a_i will randomly assign v_{ij} a value from d$_i$ and with probability 1 - μ agent a_i will set $v_{ij} = v_{iq}$.

4) *Estimation of Distribution Algorithm SoHC*

The EDA-SoHC uses a distributed restricted estimation of distribution operator (dREDO) which is commonly used in Estimation of Distribution Algorithms (EDA)[55]. In order to create a gene dCS$_i$, the dREDO first applies a distributed uniform mutation operator with probability μ to a random parent. Alternatively, with probability (1-μ)ε (where a higher ε variation operator value produces a higher variation rate) an agent will select a value for dCS$_i$ from a probability distribution function based on the above average individuals (dCS$_s$). The generated offspring, dCS$_i$, will possess attributes that most often appear in above-average individuals. With probability (1-μ)(1-ε) the agent leaves the associated value of dCS$_i$ unchanged. In this manner, all of the below-average individuals in the population are replaced with offspring.

IV. Results and Discussion

A. *Experiment Settings*

In this experiment, we compared five forms of SoHCs (SoHC, GSoHC$_{ux}$, GSoHC$_{mtpx}$, ESoHC, EDA-SoHC). For the SoHC (running the Society of Hill-Climbers described in Section III without any genetic protocols), experimentation showed that the population size 32 yielded the best results (with respect to success rate and communication cycles) out of {1,2,4,8,16, 32}. We chose our mutation rate μ to be 0.06, as it proved to be the best performer from {0.03, 0.06, 0.12, 0.25}. The above best values were obtained from the experiments described in [6]. For our experiments, we tested the algorithms once each on a test suite of 400 instances of randomly generated DisACSPs of the form n = 30 variables, number of domain values m = 6, arc density $p1_\alpha$ = 1.0, and varying tightness p2 from {0.03, 0.04, 0.05, 0.06}. For the EDA-SoHC, we explored values of ε between 0 and .936. For the purposes of comparison, a value of ε = 0.125 was utilized; this value proved to have the best performance at the phase transition (p2=0.06), which is the most difficult class of problems [30]. For problems beyond the phase transition (more tightly constrainted), various techniques such as arc revision and arc consistency can be used to demonstrate the lack of a solution [23].

B. *Results*

In each of the tables 1-4, the first column represents the algorithms (Alg.); the second is Success Rate for the algorithm when given a maximum of 2000 cycles to solve each problem in the class (SR); the third is Average Cycles needed to discover a solution, and the last column shows the results of a single point ANOVA test between the cycles of the algorithm and the cycles of the EDA-SoHC (where a p value less than or equal to 0.05 will be considered statistically significant difference).

As shown in Table I with p2 = 0.03, all of the algorithms had a 100% success rate and the EDA-SoHC slightly outperforms all of the algorithms with respect to average number of cycles. There is statistically significant distinction between the cycles of the EDA-SoHC and the other

Table I
Comparison of SoHC, GSoHC, ESoHC, and EDA-SoHC for μ = .06, <30,6,1.0,0.03>

Alg.	SR	Cycles	p
SoHC	1.00	17.93	6.16E-28
GSoHC$_{tpx}$	1.00	14.68	1.162E-10
GSoHC$_{mtpx}$	1.00	13.15	0.00278
ESoHC	1.00	12.78	0.04272
EDA-SoHC	1.00	11.98	--

algorithms. The average number of cycles per run of all the algorithms is 14.104, an indication of the relative ease of these particular problems.

In Table II with p2 = 0.04, all of the algorithms had a 100% success rate and EDA-SoHC performs slightly worse than ESoHC, but better than SoHC and the GSoHCs. Only the comparisons with SoHC and $GSoHC_{upx}$ show statistically significant differences. The average number of cycles per run of all the algorithms is 33.406, which is still fairly low. Of particular interest is the disparity between the SoHC and the ESoHC and EDA-SoHC. The number of cycles required to reach the same success rate is nearly halved in the later two.

TABLE II
COMPARISON OF SOHC, GSOHC, ESOHC, AND EDA-SOHC FOR
μ = 0.06, <30,6,1.0, 0.04>

Alg.	SR	Cycles	p
SoHC	1.00	54.28	2.445E-9
$GSoHC_{upx}$	1.00	31.11	0.029125
$GSoHC_{mtpx}$	1.00	28.32	0.5207
ESoHC	1.00	26.09	0.509
EDA-SoHC	1.00	27.23	--

In Table III with p2 = 0.05, the SoHC discovered a relatively low SR of 60%, the GSoHCs and ESoCH had a 98% success rate, but EDA-SoHC falls to a 97% SR. Requiring 377.48 cycles, EDA-SoHC performs noticeably worse than all algorithms, excepting the SoHC which takes 1180.34. Based on the results of the ANOVA test, only the comparison with SoHC shows statistical distinction.

Table III
Comparison of SoHC, GSoHC, ESoHC, and EDA-SoHC for
μ = 0.06, <30,6,1.0, 0.05>

Alg.	SR	Cycles	p
SoHC	0.60	1180.34	5.7179E-20
$GSoHC_{upx}$	0.98	270.39	0.0799
$GSoHC_{mtpx}$	0.98	273.62	0.10172
ESoHC	0.98	263.2	0.068946
EDA-SoHC	0.97	377.48	--

Table IV shows the performance at the phase transition (p2=0.06). The SoHC manages only a 1% success rate, the $GSoHC_{upx}$ discovers a SR of 8%, while all the other algorithms achieve an 11% SR. With respect to cycles, EDA-SoHC outperforms the other three algorithms (although only slightly). It is worth noting that since many of the problems could not be solved, the average cycle values tend towards 2000 (the number averaged in if no solution is found by that point). Only the SoHC shows statistical distinction, but ESoHC and EDA-SoHC show extremely high statistical correlation.

TABLE IV
COMPARISON OF SOHC, GSOHC, ESOHC, AND EDA-SOHC FOR
μ = .06, <30,6,1.0, 0.06>

Alg.	SR	Cycles	p
SoHC	0.01	1981.29	0.0063
$GSoHC_{upx}$	0.08	1884.99	0.501404
$GSoHC_{mtpx}$	0.11	1901.72	0.309
ESoHC	0.11	1849.02	0.919
EDA-SoHC	0.11	1842.17	--

V. CONCLUSIONS AND FUTURE WORK

The applied evolutionary operators, known as crossover and mutation, on randomly generated distributed asymmetric CSPs are studied in the work presented in this paper. The SoHCs, GSoHCs, EDA-SoHC and ESoHC are described and a series of experimental results and comparisons are presented. In particular, GSoHCs using distributed restricted forms uniform crossover, and modified two-point crossover, are studied with respect to SR, Cycles and their statistical correlation to the EDA-SoHC (ε =0.125).

From experimental results and a series of comparisons, we can draw the following conclusions:

In general, we note that SoHC performs noticeably worse than all the other algorithms, save for the easiest instances when p2 = 0.03.

Generally speaking, GSoHCmtpx has better performance than GSoHCupx due to the fact that it takes more genes from the best dCS.

The GSoHCs and ESoHC have better performance than the simple SoHC. The EDA-SoHC protocol performs roughly on par with ESoHC under most conditions. Further, the differences between the number of cycles for EDA-SoHC and for the other algorithms are generally not statistically significant, except for the easier instances (p2 = 0.03 and 0.04). In general, we attribute the success of the EDA-SoHC to its application of selection pressure through taking genetic material from the above average members of the population. At the same time, the algorithm preserves a level of population diversity by granting the opportunity for a child to obtain attributes from any of the above-average individuals (not merely the best).

Comparing the results between GSoHC, ESoHC, and EDA-SoHC suggests that the variation of the above-average dCS's plays the most important role in the genetic and evolutionary SoHC search. The specific type of operator creating this variation is only of secondary importance [32]. Our future work will be devoted to discovering a form of recombination that is has a performance equal to or better than the distributed restricted uniform mutation operator and the EDA-SoHC.

REFERENCES

[1] Bäck, T., Evolutionary Algorithms in Theory and Practice, New York: Oxford Univ. Press, 1996.

[2] Bäck, T., Hammel, U., and Schwefel, H. P., "Evolutionary Computation: Comments on the History and Current State", IEEE Trans. on Evolutionary Computation, Vol. 1, No, 1, pp. 3-17, 1997.

[3] De Jong, K., and Spears. K., “On the State of Evolutionary Computation”, Proc. of the Fifth Int. Conf. on Genetic Algorithms, pp. 618-623, Morgan Kaufmann Publishers, San Francisco, CA, 1993. 6.

[4] Dorigo, M., and Gambardella, L. M., “Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem”, IEEE Trans. on Evolutionary Computation, Vol. 1, No. 1, pp. 53-66, 1997.

[5] Dozier, G., Homaifar, A., Tunstel, E., and Battle, D., “An Introduction to Evolutionary Computation”, pp. 365-379, Chapter 17, Intelligent Control Systems Using Soft Computing Methodologies, CRC Press.

[6] Dozier, G., Cunningham, H., Britt, W., and Zhang, F., “Distributed Constraint Satisfaction, Restricted Recombination, and Hybrid Genetic Search.” GECCO(1) 2004: 1078-1087.

[7] Dumitrescu, D., Lazzerini, B., Jain, L. C., and Dumitrescu, A., Evolutionary Computation, International Series on Computational Intelligence, CRC Press, 2000.

[8] Fogel, D. B., Evolutionary Computation: Toward a New Philosophy of Machine Intelligence, 2nd Edition, IEEE Press, 2000.

[9] Goldberg, D. E., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing Company, Inc., 1989.

[10] Michalewicz, Z., Genetic Algorithms + Data Structures = Evolution Programs, 3rd Edition, Artificial Intelligence series, Springer-Verlag, Berlin, 1996.

[11] Spears, W. M., De Jong, K. A., Bäck, T., Fogel, D. B., and De Garis, H., “An overview of Evolutionary Computation”, Proc. of the 1993 European Conf. on Machine Learning, pp. 442-459, 1993.

[12] Carlisle, A., and Dozier, G. (2001), “An off-the-shelf PSO”, Proceedings of the Workshop on Particle Swarm Optimization. Indianapolis, IN: Purdue School of Engineering and Technology, IUPUI (in press).

[13] Minton, S., Johnston, M. D., Philips, A. B., and Laird, P., “Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling problems”, Artificial Intelligence 58(1-3), 161-205, 1992.

[14] Prosser, P., “Binary constraint satisfaction problems: some are harder than others”, in Proc. ECAI-94, Cohn, A. G. (ed.), John Wiley & Sons, Ltd. 1994.

[15] Selman, B., Levesque, H., and Mitchell, D., “A New Method for Solving Hard Satisfaction Problems”, In Proceedings of AAAI-92, San Jose, California, 1992.

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

[17] Bowen, J., and Dozier, G., "Constraint Satisfaction Using A Hybrid Evolutionary Hill-Climbing Algorithm That Performs Opportunistic Arc and Path Revision", Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference, AAAI Press/The MIT Press, August 1996, Portland, Oregon, pp.326-331.

[18] Dozier, G., Bowen, J., and Bahler, D., “Solving Small and Large Scale Constraint Satisfaction Problems Using a Heuristic-Based Microgenetic Algorithm”, Proceedings of the 1994 IEEE International Conference on Evolutionary Computation, pp. 306-311, 1994.

[19] Dozier, G., Bowen, J., and Bahler, D., “Solving Randomly Generated Constraint Satisfaction Problems Using a Hybrid Microgenetic Search Technique That Evolves a Population of Hill-Climbers”, IEEE Transactions on Evolutionary Computation, IEEE, Perth, Australia, November 1995, pp. 614-619.

[20] Dozier, G., Bowen, J., and Homaifar, A. (1998). “Solving Constraint Satisfaction Problems Using Hybrid Evolutionary Search”, Proceedings of the 1995 IEEE International Conference on Evolutionary Computation, Vol. 2, No. 1, pp. 23-33, April 1998, Institute of Electrical & Electronics Engineers.

[21] Kowalczyk, R., “Constraint Consistent Genetic Algorithms”, Proceedings of the 1997 IEEE International Conference on Evolutionary Computation, IEEE, Indianapolis, IN, April 1997, pp.343-348.

[22] Marchiori, E., “Combining Constraint Processing and Genetic Algorithms for Constraint Satisfaction Problems”, Proceedings of the 7th International Conference on Genetic Algorithms, Bäck , T., Ed., Morgan Kaufmann, East Lansing, MI, July 1997, pp. 330-337.

[23] Solnon, C. (2002). “Ants Can Solve Constraint Satisfaction Problems”, IEEE Transactions on Evolutionary Computation. 6(4):347-357, 2002

[24] Craenen, B. G. W., Eiben, A. E., and E. Marchiori, “Solving Constraint Satisfaction Problems with Heuristic-based Evolutionary Algorithms”, Proceedings of the Congress on Evolutionary Computation, CEC 2000, pp. 1571-1577, 2000.

[25] Eiben, A. E., van der Hauw, J. K., and van Hemert, J. I., “Graph Coloring with Adaptive Evolutionary Algorithms”, Journal of Heuristics, Vol. 4, pp. 25-46.

[26] Paredis, J., “Co-Evolutionary Constraint Satisfaction”, Proceedings of the 3th International Conference on Parallel Problem Solving from Nature, pp. 46-56, Eds. Y. Davidor, H. P. Schwefel and R. Männer, No. 866 Lecture Notes in Computer Science, Springer-Verlag, 1994.

[27] Riff-Rojas, M.C, “Using the Knowledge of the Constraint Network to Design an Evolutionary Algorithm that Solves CSP”, Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, IEEE, Nagoya, Japan, May 1996, pp. 279-284.

[28] Riff-Rojas, M.C, “Evolutionary Search Guided by the Constraint Network to Solve CSP”, Proceedings of the 1997 IEEE International Conference on Evolutionary Computation, IEEE, Indianapolis, IN, April 1997, pp. 337-342, 1997.

[29] Dozier, G., “Distributed Constraint Satisfaction via a Society of Hill-Climbers”, The Proceedings of the 2002 World Automation Congress (Soft Computing, Multimedia, Biomedicine, Image Processing, and Financial Engineering), pp. 313-318, ISSCI-025, Orlando, FL, TSI Press, 2002.

[30] Dozier, G., “Solving Distributed Asymmetric CSPs via a Society of Hill-Climbers”, The Proceedings of the International Conference on Artificial Intelligence, pp. 949-953, 2002

[31] Dozier, G., “Distributed Recurrent Constraint Satisfaction via a Genetic Society of Hill-Climbers”, The Proceedings of the International Conference on Artificial Intelligence, pp. 1400-1403, 2002.

[32] Dozier, G., “Solving Distributed Asymmetric Constraint Satisfaction Problems Using genetic and Evolutionary Society of Hill-Climbers”, Submitted to IEEE Trans. on Evolutionary Computation, 2003.

[33] Zhang, F., “A Comparison of Distributed Restricted Recombination Operators for Genetic Societies of Hill-Climbers: A DisACSP Perspective”, Master Thesis, Auburn University Department of Computer Science and Software Engineering, December, 2003.

[34] Yokoo, M., Distributed Constraint Satisfaction, Springer-Verlag, 2001.

[35] Zhang, W. and Wittenburg, L., “Distributed Breakout Revisited”, In Proc. AAAI-02, to appear, 2002.

[36] Zhang, W. and Xing, Z., “Distributed Breakout vs. Distributed Stochastic: A Comparative Evaluation on Scan Scheduling”, In Proc. AAMAS-02 Workshop on Distributed Constraint Reasoning, to appear, 2002.

[37] Zhang, W., Wang, G. and Wittenburg, L., “Distributed stochastic search for distributed constraint satisfaction and optimization: Parallelism, Phase Transitions and Performance”, In Proc. AAAI-02 Workshop on Probabilistic Approaches in Search, to appear, 2002

[38] Calisti, M., Frei, C., and Faltings, B. (1999), “A Distributed Approach for QoS-Based Multi-Domain Routing”, Proceedings of the AAAI Workshop on Artificial Intelligence for Distributed Information Networking (AiDIN’99).

[39] 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.

[40] Calisti, M., and Faltings, B. (2000) "Distributed Constrained Agents for Allocating Service Demands in Multi-Provider Networks", Journal of the Italian Operational Society, Special issur on Constraint Problem Solving, vol. XXIX, no. 91.

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

[42] 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.

[43] Krishnamachari, B., Bejar, R., and Wicker, S. (2002), "Distributed Problem Solving and the Boundaries of Self-Configuration in Multi-hop Wireless Networks", Proceedings of the Hawaii International Conference on System Sciences, HICSS-35.

[44] Silaghi, M. C., Sam-Haroud, D., Calisti, M., and Falting, 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.

[45] Dechter, R, "Constraint networks", Encyclopedia of Artificial Intelligence, pages 276-285, 1992.

[46] Kumar, V., "Algorithms for Constraints Satisfaction Problems: A Survey", AI Magazine, vol. 13, pp. 32-44, 1992.

[47] Smith, B. M, "Phase Transition and the Mushy Region in Constraint Satisfaction Problems", The Proceedings of the 11th European Conference on Artificial Intelligence, A.G. Cohn (ed.) pp.100-104, John Wiley & Sons, Ltd, 1994.

[48] Tsang, E. (1993). Foundations of Constraint Satisfaction, Academic Press, Ltd.

[49] Sebag, M., and Shoenauer, M., "A Society of Hill-Climbers", The Proceedings of the 1997 International Conference on Evolutionary Computation, pp. 319-324, IEEE Press, 1997.

[50] Morris, P., "The Breakout Method for Escaping from Local Minima", Proceedings of AAAI-93, pp. 40-45, 1993.

[51] Jones, T., "Crossover, Macromutation, and Population-based Search", Proceedings of the Sixth International Conference (ICGA) 73-80, Morgan Kaufmann, 1995.

[52] Mühlenbein, H., and Schlierkamp-Voosen, D, "Analysis of Selection, Mutation and Recombination in Genetic Algorithms". Technical Report 93-24, GMD, 1993.

[53] Spears, W. M., "Crossover or Mutation?" pp. 221-237 of: Whitley, L. D., editor, Foundations of Genetic Algorithm, vol. 2, San Mateo, CA: Morgan Kanfmann, 1993.

[54] Spears, W. M., De Jong, K. A., "An Analysis of Multi-Point Crossover", Proceedings of the Foundations of Genetic Algorithms Workshop, G. Rawlins (ed.), Morgan Kaufmann Publishers.

[55] Larranaga, P., and Lozano, J.A., Estimation of Distribution Algorithms. Kluwer Academic Publishers, 2001.