Extending GENET for Non-Binary Constraint Satisfaction Problems

J. H. M. Lee H. F. Leung H. W. Won
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, New Territories
Hong Kong
{jlee, lhf, hwwon}@cs.cuhk.hk

Abstract

GENET has been shown to be efficient and effective on certain hard or large constraint satisfaction problems. Although GENET has been enhanced to handle also the `atmost` *and* `illegal` *constraints in addition to binary constraints, it is deficient in handling non-binary constraints in general. In this paper, we present E-GENET, an extended GENET. E-GENET features a convergence and learning procedure similar to that of GENET and a generic representation scheme for general constraints, which range from disjunctive constraints to non-linear constraints to symbolic constraints. We have implemented an efficient prototype of E-GENET for single-processor machines. Benchmarking results confirms the efficiency and flexibility of E-GENET. Our implementation also compares well against CHIP, PROCLANN, and GENET.*

1 Introduction

Many problems in artificial intelligence and computer science in general can be formulated as *constraint satisfaction problems* (CSP's). Some examples are computer vision [1], machine design and manufacturing [2], temporal reasoning [3] and scheduling [4]. A CSP can be stated as follows: given a set of variables, a finite and discrete domain of possible values for each variable and a conjunction of constraints limiting the combination of values that the variables can take, the goal is to find a *consistent* assignment of values to the variables so that all the constraints are satisfied simultaneously. CSP's are, in general, NP-complete and some are NP-hard [5].

Two main approaches have been developed for solving CSP's. One approach features various degrees of combinations of backtracking tree search and constraint propagation [6, 7]. This framework is realized in most constraint logic programming languages, such as CHIP [8] which has been applied successfully to real-life industrial applications [9, 10, 11]. Although various heuristics, such as variable and value orderings [12, 13], have been proposed to speed up tree search, this class of techniques are algorithmic in nature and cannot escape from the curse of NP-completeness, which is the main barrier in solving large-scale and hard CSP's.

Another approach is often known as the iterative repair method [14, 15], which consists of starting with a complete but possibly inconsistent variable assignment and incrementally repairing constraint violations by reassigning variables until a consistent variable assignment is achieved. Some standard problems such as N-queens and graph-coloring can be solved in orders of magnitude better than traditional backtracking techniques. The average solution time for the million-queens problem is reduced to less than one minute and a half on a SPARCstation [14]. A problem of this approach is that execution can easily be trapped in local minima (or maxima), a state in which no repair can be made but the current assignment is still inconsistent. When trapping occurs, execution has to be aborted. This situation is most likely to occur in highly constrained problems [16].

Wang and Tsang propose GENET [17], a generic neural network model for solving general CSP's with binary constraints. GENET is based on the iterative repair approach but a heuristic learning rule is used to help the network escaping from local minima and to avoid the network settling in the same local minima again. As a first step in handling general non-binary constraints, Davenport *et al.* [16] extend GENET with the `illegal` and the `atmost` constraints. The enhanced GENET [16] is shown to be effective and efficient in solving such hard CSP's as hard graph-coloring problems [18] and the car-sequencing problem [19].

A general scheme for handling non-binary constraints in GENET does not exist. Although non-binary constraints with finite domain variables can readily be transformed into binary constraints [20], the resulting GENET network of the transformed CSP is usually several orders of magnitude larger than that of the original CSP. Thus this transformational approach is infeasible both in terms of memory requirement and execution efficiency. In this paper, we present E-GENET for solving general constraints. We replace the notion of label nodes in GENET by *variable nodes*, while retaining the concept of constraint nodes. Variable and constraint nodes allow us to represent disjunctive, non-binary, non-linear and symbolic constraints in a homogeneous and flexible fashion. We have implemented an E-GENET simulator for single-processor systems and benchmarked our implementation on CSP's of various types. E-GENET compares favorably against GENET [16], CHIP [8], and PROCLANN [21], a constraint logic programming language based on GENET.

The rest of this paper is organized as follows. In section 2, we explain the deficiency of GENET in handling non-binary constraints, followed by an exposition of the E-GENET architecture and convergence procedure. Benchmarking results of E-GENET are presented in section 3. Section 4 summarizes our contributions and sheds light on future work.

2 An Extended GENET

Due to lack of space, we assume familiarity with GENET [17, 16] in the following presentation.

Theoretically, with finite domains, all non-binary constraints can readily be transformed into binary constraints. If variables x, y, and z have domain $\{1, 2, 3, 4\}$, then tuples satisfying "$x+y=z$" are listed as follows:

i	x	$+$	y	$=$	z
0	1		1		2
1	1		2		3
2	1		3		4
3	2		1		3
4	2		2		4
5	3		1		4

We observe that each tuple can be labeled with a unique index, which is shown in the first column of the table. We can decompose "$x + y = z$" into three constraints "$c_1(t, x) \wedge c_2(t, y) \wedge c_3(t, z)$," where t is a new variable with domain $\{0, 1, 2, 3, 4, 5\}$. We list the tuples permitted by c_1, c_2 and c_3 as follows.

$c_1($ t ,	x $)$
0	1
1	1
2	1
3	2
4	2
5	3

$c_2($ t ,	y $)$
0	1
1	2
2	3
3	1
4	2
5	1

$c_3($ t ,	z $)$
0	2
1	3
2	4
3	3
4	4
5	4

The GENET network for the decomposed constraints contains an extra variable of domain size 6 and 72 node connections. If we have to handle the constraint "$u + v = x + y$" with all variables having domain $\{1, \ldots, 100\}$, then we have to introduce a new variable of domain size 666,700 and 264,013,200 node connections. Hence it is impractical to implement non-binary constraints in GENET.

We propose E-GENET, an extended GENET for efficient general, binary and non-binary, constraint handling. E-GENET allows representation of arbitrary general constraints, thus avoiding the introduction of new variables and large number of node connections. E-GENET replaces the notion of label nodes by variable nodes, which are equivalent to clusters of label nodes in GENET. Thus a E-GENET network is in general smaller in size and has few connections than a corresponding GENET network.

2.1 Network Architecture

E-GENET has two types of nodes: *variable nodes* and *constraint nodes*. Each variable in a CSP is represented by a variable node, which contains the domain associated with the variable. The *state* S_x of a variable node x is defined to be the current variable assignment. A *constraint node* is created for each constraint in the CSP. A variable node x is connected to a constraint node c if x occurs in c[1]. Consider the CSP "$x + y + z = 9 \wedge 3y - z = 4$," where the domains of x, y, and z are $\{1, \ldots, 10\}$. Figure 1 shows the CSP's network in E-GENET. The constraint node

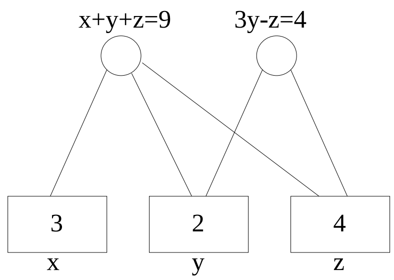

Figure 1: An example network of E-GENET

for $3y - z = 4$ is connected to related variable nodes y and z; and the constraint node for $x+y+z = 9$ is connected to all of x, y, and z. The current state of the network represents the following variable assignment: $x = 3$, $y = 2$, and $z = 4$.

Unlike GENET, node connections in E-GENET do not have weights. Consider the constraint $c(x_{i_1}, x_{i_2}, \ldots, x_{i_n})$. Each combination (or tuple) $(v_1, v_2, \ldots, v_n)$ of possible values from domains of $x_{i_1}, x_{i_2}, \ldots, x_{i_n}$ is given a *penalty value* $\delta_{c(v_1,v_2,\ldots,v_n)}$. These penalty values, stored in the corresponding constraint node, may be modified as a result of heuristic learning. Initially, penalty values of prohibited tuples are set to -1 and others to 0.

$$\delta_{c(v_1,v_2,\ldots,v_n)} = \begin{cases} 0 & c(v_1, v_2, \ldots, v_n) \text{ is true} \\ -1 & \text{otherwise} \end{cases}$$

Suppose x, y, and z have domain $\{1, 2\}$. The constraint node for $x + y = z$ contains the following

[1]We abuse terminology by naming a variable (a constraint) also by its variable (constraint) node name.

penalty values:

x	y	z	$\delta_{x+y=z}$
1	1	1	-1
1	1	2	0
1	2	1	-1
1	2	2	-1
2	1	1	-1
2	1	2	-1
2	2	1	-1
2	2	2	-1

Each value v in the domain of variable x has an input $I_{x=v}$ defined as:

$$I_{x=v} = \sum_{\text{all } c(x_{i_1},\ldots,x,\ldots,x_{i_k})} \delta_{c(S_{x_{i_1}},\ldots,v,\ldots,S_{x_{i_k}})}.$$

Thus input to each value of a variable node is a non-positive quantity. A negative input indicates that the value is involved in the violation of some constraints in the network. When the sum of input of values in the current variable assignment is zero:

$$\sum_{\text{all } x} I_{x=S_x} = 0$$

no constraints are violated and the network is in a *solution state*. The variable assignment associated with a solution state is a *solution* to the corresponding CSP.

2.2 Convergence Procedure

Dynamics of E-GENET concerns how the network changes states of variable nodes and penalty values in constraint nodes before settling in solution state(s). Initially, a complete but possibly inconsistent variable assignment is generated. The value of each variable node x is then updated to reduce constraint violation by choosing the value with maximum input:

$$x := v' \text{ if } I_{x=v'} = \max\{I_{x=v} | v \in \text{domain of } x\}$$

If there are several values with the maximum input, a value with maximum input that is currently assigned to x stays assigned. Otherwise, pick a value with maximum input randomly. Following Davenport *et al.* [16], we define a *repair* to be a variable update. Variables in the network are updated in parallel until variable assignment remains unchanged.

E-GENET terminates execution if a solution state is reached. Otherwise, the network is trapped in a local minima and *heuristic learning* is activated to help escape from the local minima. Heuristic learning in E-GENET amounts to decreasing the penalty values of tuples violating some constraints. We leave the heuristic learning rule unspecified intentionally since application domain knowledge can usually assist us in designing good learning rule for specific problems. We show two plausible heuristic learning rules for constraint c in the following:

1. $\delta_{c(S_{x_{i_1}},\ldots,S_{x_{i_k}})} :=$
   $\delta_{c(S_{x_{i_1}},\ldots,S_{x_{i_k}})} - (\delta_{c(S_{x_{i_1}},\ldots,S_{x_{i_k}})} < 0)$
2. For all possible tuples $(v_1,\ldots,v_k)$,
   $\delta_{c(v_1,\ldots,v_k)} := \delta_{c(v_1,\ldots,v_k)} - (\delta_{c(v_1,\ldots,v_k)} < 0)$

where $<$ is a boolean function returning 1 if the comparison is true and 0 otherwise. The two rules differ in that the former increases penalty on only the violating tuple made from current variable assignment, while the latter penalizes all violating tuples of the constraints. Both rules help to unstablize a network trapped in local minima, after which variable updating is repeated again.

The *convergence procedure* can be summarized as follows:

```
repeat
    update all variable nodes in parallel
    asynchronously until no repair

    if (sum of inputs of values in current
        assignment is zero) then
        terminate with success
    else
        activate heuristic learning
until (time limit or maximum number of
      cycles is exceeded)
```

3 Benchmarking Results

To illustrate the feasibility and effectiveness of our proposal, we have built an E-GENET simulator in about 1,200 lines of C code. We test our implementation on the graph-coloring problems, the N-queens problem, the car-sequencing problem, the cryptarithmetic problem, and the Hamiltonian path problem. We compare our result with that of CHIP version 4.0.1 [22], which uses traditional constraint propagation and backtracking tree search for constraint solving. Since we do not have access to a GENET implementation, we compare instead to PROCLANN [21], which is a constraint logic programming language with GENET as the constraint solver. Wherever possible, we quote results of GENET from the paper of Davenport *et al.* [16], timing of which is obtained on a SUN SPARCclassic. All benchmarking is performed on a SUN SPARCstation 10 model 30. Timing and number of repairs results for GENET, PROCLANN, and E-GENET are median of 10 runs. A "–" symbol means that the execution fails due to either execution time exceeded or memory exhaustion.

3.1 The Graph-Coloring Problem

The task of the graph-coloring problem is to color each vertex of a graph so that no two neighboring nodes share the same color. We use two sets of problems to test our implementations. The first set contains randomly generated simple graph 10- colorability problems; while the second set contains hard problems described in the paper of Johnson *et al.* [18]. Results of CHIP, PROCLANN, and E-GENET are shown in tables 1 and 2.

CHIP is deficient in graph-coloring problems since little pruning is possible. Execution of CHIP basically reverts to backtracking tree search. Thus CHIP is not able to solve either of the two hard problems with 125 vertices within 48 hours and runs out of memory for

nodes	CHIP	PROCLANN	E-GENET
110	0.98 s	0.31 s	0.03 s
130	1.31 s	0.43 s	0.03 s
150	1.81 s	0.55 s	0.03 s
170	2.23 s	0.68 s	0.04 s
190	2.89 s	0.80 s	0.04 s
210	3.75 s	1.03 s	0.05 s
230	4.56 s	1.22 s	0.04 s
250	5.38 s	1.35 s	0.04 s

Table 1: Results on simple graph-coloring problems

hard problems with 250 nodes on our machine with 32M memory.

nodes	colors	CHIP	GENET
125	17	$>$ 48 hrs	2.6 hrs
125	18	$>$ 48 hrs	23 s
250	15	–	4.2 s
250	29	–	1.1 hrs
nodes	colors	PROCLANN	E-GENET
125	17	2.3 hrs	2.5 hrs
125	18	2.5 mins	2.6 mins
250	15	1.1 hrs	7.4 s
250	29	4.6 hrs	5 hrs

Table 2: Results on hard graph-coloring problems

For simple problems, both PROCLANN and E-GENET can converge in relatively few convergence cycles. The difference in performance is a result of the difference in network size and hence the difference in network construction time. This difference disappears in the hard problems since network construction accounts for only a negligible portion of the execution time.

Thus E-GENET is at least as efficient as GENET and PROCLANN in solving binary CSP's as expected since E-GENET and GENET are similar in handling binary constraints.

3.2 The N-queens Problem

The N-queens problem is to place N queens on a $N \times N$ chessboard so that no two queens attack each other. In this experiment, we set the time limit to 6000 second. PROCLANN cannot solve more than 140 queens in the limit. E-GENET can solve up to 500 queens. The CHIP program employs the first-fail principle in labeling. CHIP's performance fluctuates almost arbitrarily and is unpredictable since the CHIP approach is incredibly sensitive to data [23]. Any random change in the definition of constraints can result in drastic different runtime behavior. The timing results are summarized in table 3.

The GENET network for the N-queens problem is large: the number of label nodes, the number of constraints, and the number of connections are of order $O(N^2)$, $O(N^2)$, and $O(N^3)$ respectively. Most part of PROCLANN's execution is spent in network construction, where a form of consistency check is performed. The flexibility of E-GENET allows us to define a new constraint `noattack` for restricting queen positions. The new constraint enables us to further reduce the network construction time and memory requirement.

N	CHIP	PROCLANN	E-GENET
10	0.030 s	0.153 s	0.046 s
20	0.070 s	1.753 s	0.165 s
30	0.290 s	8.667 s	0.510 s
40	0.170 s	28.500 s	1.222 s
50	21.520 s	70.934 s	3.582 s
60	0.910 s	147.326 s	6.840 s
70	23.110 s	272.837 s	9.902 s
80	180.020 s	464.113 s	19.752 s
90	–	743.008 s	28.467 s
100	106.690 s	1232.717 s	37.582 s
110	1.270 s	1842.883 s	42.211 s
120	70.120 s	2408.887 s	61.672 s
130	1.980 s	4701.633 s	86.083 s
140	1.690 s	5634.042 s	94.377 s
150	–	–	152.001 s
160	–	–	188.033 s
170	2.750 s	–	219.317 s
180	–	–	264.543 s
190	3.750 s	–	316.562 s
200	–	–	439.952 s

Table 3: Results on N-queens problems

3.3 The Car-Sequencing Problem

The car-sequencing problem involves scheduling cars onto an assembly line so that different options can be installed on these cars satisfying various utilization constraints [9]. We compare the performance of GENET and E-GENET on problems with utilization percentage from 60% to 80%. 50 problems are tested, 10 for each percentage. The execution limit of GENET is set to one million repairs for each run [16] while that of E-GENET is set to 1000 repairs.

This benchmark compares the `atmost` constraint of E-GENET. The result is listed in table 4. The median number of repairs of E-GENET are less than 50% of those of GENET. E-GENET also achieves similar percentages of successful runs using a much smaller limit on the number of repairs. We are investigating this unexpected good performance of E-GENET.

All successful runs of E-GENET terminate in less than 2 minutes. CHIP can only manage to solve within one hour 6 out of the 50 problems.

3.4 Cryptarithmetic

One well-known example of cryptarithmetic is to assign each letter in $\{S, E, N, D, M, O, R, Y\}$ a different digit from $\{0, 1, \ldots, 9\}$ so that the equation $SEND + MORE = MONEY$ holds.

We test the performance of PROCLANN and E-GENET on larger problems adopted from Van Hentenryck's book [24]. The result is listed in table 5.

utilization %	GENET % succ. runs	GENET median repair	E-GENET % succ. runs	E-GENET median repair
60	84	463	74	223.5
65	87	426	80	223.5
70	83	456	81	241
75	85	730	84	339
80	50	4529	53	576

Table 4: Results on car sequencing problems

	PROCLANN	E-GENET
problem 1	77.49 s	4.28 s
problem 2	74416.64 s	8.00 s

Table 5: Results on cryptarithmetic problems

For PROCLANN, we have to decompose the constraints using the method described in section 2. The results of PROCLANN do not include the time for decomposing the non-binary constraints. E-GENET can represent the constraints in the problems almost directly; thus the good performance. This test demonstrates the non-binary constraint handling of E-GENET. On contrary to the graph-coloring problem, constraint propagation alone can almost solve the cryptarithmetic problems entirely, requiring little backtracking. This explains why CHIP can solve both problems within 1 second.

3.5 Hamiltonian Path

The Hamiltonian path problem is stated as follows: given a graph of n vertices, we have to find an ordering of these n vertices $\langle v_1, v_2, \ldots, v_n \rangle$ so that for all i, $1 \leq i < n$, there is an edge between v_i and v_{i+1}.

We specify the topology of a graph and its associated Hamiltonian path problem using constraints as follows. For a graph of n vertices, we use n variables with domain $\{1, \ldots, n\}$. Each variable denotes the ordering of the corresponding vertex in the graph. Thus variables must be *all different*. This can be modeled conveniently using the `alldifferent` constraint. If a vertex v has only one neighbor v', then v and v' must be consecutive numbers, described by the following *disjunctive* constraint:

$$v = v' + 1 \text{ or } v' = v + 1$$

If a vertex v has more than one neighbor $v_a, v_b, \ldots, v_k$, then the associated constraints are described by the following two disjunctive constraints:

$$\begin{cases} v = 1 \text{ or } v = v_a + 1 \text{ or } \ldots \text{ or } v = v_k + 1 \\ v = n \text{ or } v_a = v + 1 \text{ or } \ldots \text{ or } v_k = v + 1 \end{cases}$$

While CHIP uses choices [24] to model disjunctive constraints, E-GENET can represent disjunctive constraints directly. The CHIP approach induces a large number of backtracking and fails to solve a graph with 50 vertices in 10 hours. The Hamiltonian path problem exceeds the capability of GENET since it is impractical to decompose the constraints using the technique in section 2. The benchmarking results are listed in table 6.

nodes	CHIP	E-GENET
30	581.64 s	54.00 s
40	4023.50 s	587.16 s
50	–	4897.09 s

Table 6: Results on Hamiltonian path problems

4 Concluding Remarks

In this paper, an extended GENET (E-GENET) is proposed. E-GENET has four advantages over GENET. First, binary and non-binary constraints have the same status and are represented homogeneously. The representation scheme allows users to define arbitrary new constraints conveniently without resorting to modifying the underlying network model in an *ad hoc* manner. Second, E-GENET networks are in general smaller in size than their GENET counterparts. In most cases, the penalty values, which correspond to the product of weight and output in GENET, are computed on demand. This is not the case for weights and output for GENET. Thus the cost for network construction is greatly reduced. Third, smaller network size also implies less memory requirement by E-GENET. Solving the 1000-queens problem with GENET requires around 1 giga connection weights, which makes it impractical to solve the problem on current workstations. We have successfully used E-GENET to solve problems with more than 1000 queens. Fourth, the method to update penalty values in heuristic learning for each constraint node is left unspecified intentionally. Users can define suitable heuristic learning rule using problem domain knowledge so that the performance of each constraint can be tuned according to the characteristics of the constraint.

We have implemented a prototype of E-GENET. We show that E-GENET is at least as efficient as GENET on binary CSP's and outperforms GENET on non-binary CSP's. On CSP's requiring much backtracking tree search, GENET and E-GENET have better performance than that of CHIP, which excels in problems requiring almost only constraint propagation and little backtracking.

Interesting future work includes parallel implementation of E-GENET and further enhancement of E-GENET to solve partial constraint satisfaction problems (PCSP's) [25], which find applications in over-constrained scheduling problems and some expert systems. On the theoretical side, it is worthwhile to investigate the effect of allowing *positive* penalty values on the efficiency of E-GENET.

Acknowledgements

We thank Andrew Davenport for supplying the car-sequencing benchmarks. This research was supported in part by a RGC Earmarked Grant (CUHK 70/93E) of the Hong Kong University Grants Committee and a Direct Grant of The Chinese University of Hong Kong.

References

[1] A.L. Davis and A. Rosenfeld, "Cooperating processes for low-level vision: A survey", *Artificial Intelligence*, vol. 17, pp. 245–263, 1981.

[2] B. Nadel and J. Lin, "Automobile transmission design as a constraint-satifaction problem: Modeling and kinetic level", *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, vol. 5, no. 2, 1991.

[3] R. Dechter, I. Meiri, and J. Pearl, "Temporal constraint networks", *Artificial Intelligence*, vol. 49, pp. 61–95, 1991.

[4] V. Dhar and N. Ranganathan, "Integer programming versus expert systems: An experimental comparison", *Communications of the ACM*, vol. 33, pp. 323–336, 1990.

[5] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, The MIT Press, 1990.

[6] B.A. Nadel, "Constraint satisfaction algorithms", *Computational Intelligence*, vol. 5, pp. 188–224, 1989.

[7] V. Kumar, "Algorithms for constraint-satisfaction problems: A survey", *AI Magazine*, vol. 13, no. 1, pp. 32–44, 1992.

[8] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier, "The constraint logic programming language CHIP", in *Proceedings of the International Conference on Fifth Generation Computer Systems*, December 1988, pp. 693–702.

[9] M. Dincbas, H. Simonis, and P. Van Hentenryck, "Solving car sequencing problem in contraint logic programming", in *Proceedings, European Conference on AI*, 1988, pp. 290–295.

[10] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, and T. Graf, "Applications of CHIP to industrial and engineering problems", in *First International Conference on Industrial and Engineering Applications of AI and Expert Systems*, June 1988.

[11] M. Perett, "Using constraint logic programming techniques in container port planning", *ICL Technical Journal*, May 1991.

[12] J. Bitner and E.M. Reingold, "Backtrack programming techniques", *Communications of the ACM*, vol. 18, pp. 651–655, 1985.

[13] R.M. Haralick and G.L. Elliot, "Increasing tree search efficiency for constraint satisfaction problems", *Artificial Intelligence*, vol. 14, pp. 263–313, 1980.

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

[15] J. Liu and K. Sycara, "Emergent constraint satisfaction through multi-agent coordinated interaction", in *Proceedings of MAAMAW'93*, Neuchatel, Switzerland, 1993.

[16] A. Davenport, E. Tsang, C.J. Wang, and K. Zhu, "Genet: A connectionist architecture for solving constraint satisfaction problems by iterative improvement", in *Proceedings of the Twelth National Conference on Artificial Intelligence*, 1994.

[17] E.P.K. Tsang and C.J. Wang, "A generic neural network approach for constraint satisfaction problems", in *Neural Network Applications*, G. Taylor, Ed., pp. 12–22. Springer-Verlag, 1992.

[18] D. Johnson, C. Aragon, L. McGeoch, and C. Schevon, "Optimization by simulated annealing: an experimental evaluation; part 2 graph coloring and number partition", *Operations Research*, vol. 39, no. 3, pp. 378–406, 1991.

[19] B.D. Parrello, W.C. Kabat, and L. Wos, "Job-shop scheduling using automated reasoning: a case study of the car sequencing problem", *Journal of Automated Reasoning*, vol. 2, pp. 1–42, 1986.

[20] E.P.K. Tsang, *Foundations of Constraint Satisfaction*, Academic Press, 1993.

[21] J.H.M. Lee and V.W.L. Tam, "A framework for integrating artificial neural networks and logic programming", *International Journal on Artificial Intelligence Tools*, 1995, (to appear).

[22] The COSYTEC Team, *CHIP V4 User Manuals*, 1993.

[23] Y. Cras, "Using constraint logic programming in services: A few short tales", in *Logic Programming: Proceedings of the 1994 International Symposium*, 1994, pp. 3–16.

[24] P. Van Hentenryck, *Constraint Satisfaction in Logic Programming*, The MIT Press, 1989.

[25] E.C. Freuder, "Partial constraint satisfaction", in *Proceedings, 11th International Joint Conference on AI*, 1989, pp. 278–283.