Performance of a Comprehensive and Efficient Constraint Library Based on Local Search*

J. H. M. Lee, H. F. Leung, and H. W. Won

Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
{jlee, lhf, hwwon}@cse.cuhk.edu.hk

Abstract. Constraint satisfaction is one of the major areas in AI that has important real-life applications. Lee *et al.* propose E-GENET, a stochastic solver for general constraint solving based on iterative repair. Performance figures show that E-GENET compares favorably against tree-search based solvers in many hard problems. On the other hand, global constraints have been shown to be very effective in modeling complicated CSP's. They have also improved substantially the efficiency of tree-search based solvers in solving real-life problems. In this paper, we present a comprehensive and efficient library of elementary and global constraints for E-GENET. Such a library is essential for applying E-GENET to complex real-life applications. We first present the improved performance of some constraints that appear in our previous papers, followed by the implementation details of three additional global constraints available in the CHIP constraint language. Experimental results, using standard benchmarks and a real-life problem, confirm empirically that the E-GENET architecture is comparable to, if not better than, state of the art in constraint solver technology.

1 Introduction

The research into constraint satisfaction problems (CSP) [14] is one of the major topics in artificial intelligence that have important real-life applications. A CSP can be stated as follows. We are given a finite set V of variables. Each variable x ranges over its domain D_x, a finite set of discrete constants. There is a finite set C of constraints over these variables, which limits the allowed combination of values that can be taken by the variables. The goal is to find a consistent assignment of values to the variables so that all constraints are satisfied.

Two main approaches to tackle CSP's are backtracking tree-search, probably enhanced with consistency algorithms [10], and iterative repair methods [15]. An extension of GENET [5], E-GENET [11, 12] is a stochastic solver for general constraint solving based on iterative repair. Performance figures show that E-GENET compares favorably against tree-search based solvers (such as CHIP [7])

* This project is supported in part by a CUHK Direct Grant.

in many hard problems. E-GENET encapsulates both an architecture and an algorithm. The architecture encodes a compact representation scheme for binary and non-binary constraints, whereas the algorithm governs how solution is searched using an iterative repair procedure coupled with learning for escaping from local minima. The Breakout Method of Morris [16] emphasizes more on the algorithmic aspect of the method, without explaining how constraints are represented. Walser [25] and Walser *et al.* [26] restrict attention to linear pseudo-Boolean and integer constraints in the WSAT algorithm. Thornton and Sattar [23] applies weighted iterative repair to the solution of real-life over-constrainted nurse rostering but no details of the solver architecture is given.

In order for E-GENET to be applicable to real-life applications, we design and implement a comprehensive library of constraints for E-GENET. The library consists of elementary and global constraints [3], the latter of which have been shown to be effective in modeling and solving a large class of difficult sequence, scheduling, geometrical placement and vehicle routing problems [19, 18, 20]. Algorithmic implementation of global constraints is nontrivial, requiring sophisticated and specific algorithms. We demonstrate how simplistic representation of these constraints in E-GENET can result in performance superior to state of the art algorithmic implementation.

While the design details of some constraints in the library were described previously [11, 12], we give improved performance figures of these constraints, resulting from fine tunning of our implementation. We further describe how three global constraints, namely `among`, `diffn`, and `cycle`, can be effectively implemented in E-GENET. Benchmarking results, including the Mystery Shopper Problem, confirm the representational and algorithmic efficiency of the E-GENET architecture and algorithm respectively.

The rest of the paper is organized as follows. We briefly review E-GENET in section 2. Performance results of the library and design of the new global constraints are reported in section 3. Section 4 summarizes and remarks on the results of the paper.

2 An Overview of E-GENET

An extension of GENET [5], E-GENET [11, 12] is a network architecture that solves general binary and non-binary constraint satisfaction problems [14] (CSP's) using local search. There are two types of nodes in E-GENET. Each *variable node* represents a variable in a CSP and contains the domain of the variable. Each *constraint node* represents a constraint in the CSP. A variable node x is connected to a constraint node c if x occurs in c.[1] For a constraint $c(x_1, \ldots, x_n)$, each possible combination $(v_1, \ldots, v_n)$ of values from domains of $(x_1, \ldots, x_n)$ is given an *initial penalty value* $\delta_{c(v_1,\ldots,v_n)}$, which is 0 if $c(v_1, \ldots, v_n)$ is true, or -1 otherwise.

[1] We abuse terminology by naming a variable node (constraint node) by the corresponding variable (constraint).

The *convergence procedure* of E-GENET can be summarized as follows. Initially, a complete, but possibly inconsistent, variable assignment is generated. Let S_x denote the current value assigned to variable x. We first update each variable x in parallel asynchronously by choosing the value with the maximum *input*, where the input of a value v is defined by $I_{x=v} = \sum_{\text{all } c(x_1,\ldots,x,\ldots,x_k)} \delta_{c(S_{x_1},\ldots,v,\ldots,S_{x_k})}$. If there are several values with the maximum input, the one that is currently assigned to x stays assigned. Otherwise, an arbitrary one of the values with the maximum input is picked. If a state is reached where the variables cannot be further updated, there are two possibilities. If $\sum_{\text{all } x} I_{x=S_x} = 0$, then E-GENET is in a global maximum. The current assignment is a solution and the procedure terminates. Otherwise, E-GENET is trapped in a local maximum. We update the penalty values and restart the asynchronous updating of the variables. One *heuristic rule* to update the penalty values follows:

$$\delta_{c(S_{x_1},\ldots,S_{x_n})} := \delta_{(S_{x_1},\ldots,S_{x_n})} - (\delta_{(S_{x_1},\ldots,S_{x_n})} < 0)$$

where '$<$' is a function returning 1 if the comparison is true and 0 otherwise.

The introduction of *intermediate nodes* is shown to improve the efficiency of E-GENET significantly in many types of problems [12]. Several variables nodes are first connected to an intermediate node. The latter is then connected to the constraint node(s) as the "general representative" of these variable nodes: it computes a single value from the current values of the variable nodes and sends this single value to constraint node(s).

3 A Comprehensive Constraint Library

The constraint library for E-GENET consists of two kinds of constraints. *Elementary constraints* are simple constraints whereas *global constraints* are complex constraints found in such commercial systems as CHIP [22] and ILOG [9]. Benchmarking results in this section are median of 10 runs obtained on a SUN SPARCstation 10 model 30. Statistical averages with the best and worst runs removed are given in brackets.

3.1 Elementary Constraints

The design and implementation of elementary constraints, including linear arithmetic constraints, the `atmost` constraint and disjunctive constraints, have been previously reported in [12]. We report the improved performance of these constraints in the following.

Linear Arithmetic Constraints. A linear arithmetic constraint is a constraint of the form $U \triangle V$, where U and V are linear terms and $\triangle \in \{=, \neq, <, \leq, >, \geq\}$. Table 1 summarizes the performance figures of the improved implementation on five benchmark programs [4]: `send`, `donald` and `crypta` are cryptarithmetic problems of different sizes, while `eq10` and `eq20` are systems of 10 and 20 linear equations respectively.

Problem	CHIP CPU Time (sec)	E-GENET CPU Time (sec)
`send`	0.010	0.015 (0.020)
`donald`	0.050	0.040 (0.045)
`crypta`	0.050	0.055 (0.055)
`eq10`	0.140	0.020 (0.025)
`eq20`	0.210	0.010 (0.025)

Table 1. Results on cryptarithmetic problems and systems of linear equations

The `atmost` Constraint. The `atmost`(N, Var, Val) constraint specifies that no more than N variables in the set Var are assigned values in the value set Val.

A set of 50 different instances of the car-sequencing problem [6] with utilization percentage ranging from 60% to 80% is used as benchmarks. The new performance figures of E-GENET are summarized in table 2. The execution limit is set to 1000 repairs. CHIP is only able to solve within one hour 6 out of the 50 problems.

utiliza-	Original E-GENET		Optimized E-GENET	
tion %	% succ. runs	median repairs	% succ. runs	median repairs
60	74	223.5	100	282.5
65	80	223.5	99	262
70	81	241	100	280.5
75	84	339	97	331
80	53	576	73	537

Table 2. Results on car sequencing problems

Disjunctive Constraints. A disjunctive constraint in E-GENET is of the form $C_1 \vee C_2 \vee \ldots \vee C_n$. The new performance figures of E-GENET on the Hamiltonian path problem are listed in table 3. CHIP uses choices [24] to model disjunctive constraints and its performance is worse than that of E-GENET.

graph node	CHIP CPU Time (sec)	E-GENET CPU Time (sec)
30	581.640	2.200 (2.545)
40	4023.500	8.865 (9.575)
50	$>$ 10 hours	110.235 (301.710)

Table 3. Results on Hamiltonian path problems

3.2 Global Constraints

In this section, we describe methods to implement global constraints in E-GENET. We first report the new performance figures for the `cumulative` constraint previously presented in [12]. Then we describe the implementations of the three global constraints, namely the `among`, `diffn` and `cycle` constraints, in E-GENET and present their performance figures.

task no.	utiliza-tion %	CHIP CPU Time (sec)			E-GENET CPU Time (sec)		
7	80	0.020	0.010	0.010	$0.005_{(0.008)}$	$0.000_{(0.000)}$	$0.000_{(0.000)}$
	85	0.025	0.010	0.010	$0.005_{(0.010)}$	$0.000_{(0.005)}$	$0.000_{(0.001)}$
	90	0.015	0.010	0.010	$0.005_{(0.030)}$	$0.000_{(0.005)}$	$0.000_{(0.000)}$
	95	0.015	0.010	0.010	$0.015_{(0.025)}$	$0.010_{(0.015)}$	$0.000_{(0.000)}$
	98	0.015	0.010	0.010	$0.010_{(0.015)}$	$0.010_{(0.010)}$	$0.000_{(0.003)}$
	100	0.020	0.010	0.010	$0.010_{(0.020)}$	$0.005_{(0.005)}$	$0.000_{(0.004)}$
14	80	0.205	0.025	0.015	$0.010_{(0.030)}$	$0.005_{(0.010)}$	$0.005_{(0.005)}$
	85	0.030	0.030	0.020	$0.020_{(0.025)}$	$0.015_{(0.020)}$	$0.010_{(0.011)}$
	90	4.630	0.020	0.070	$0.165_{(0.340)}$	$0.050_{(0.054)}$	$0.000_{(0.000)}$
	95	43.920	6.940	5.330	$2.580_{(3.468)}$	$0.490_{(1.086)}$	$0.015_{(0.018)}$
	98	> 3600	2.430	2.800	$2.780_{(5.919)}$	$1.455_{(12.488)}$	$0.020_{(0.054)}$
	100	55.280	0.020	1.740	$22.210_{(45.874)}$	$2.345_{(10.689)}$	$0.075_{(0.126)}$
21	80	0.030	0.030	0.035	$0.020_{(0.040)}$	$0.010_{(0.014)}$	$0.010_{(0.010)}$
	85	0.055	0.045	0.030	$0.045_{(0.243)}$	$0.045_{(0.049)}$	$0.010_{(0.010)}$
	90	1269.560	0.125	0.030	$0.140_{(0.135)}$	$0.220_{(0.251)}$	$0.020_{(0.019)}$
	95	> 3600	> 3600	4.190	$13.565_{(20.385)}$	$1.895_{(2.043)}$	$0.035_{(0.039)}$
	98	> 3600	> 3600	> 3600	$263.675_{(289.125)}$	$89.815_{(111.614)}$	$0.100_{(0.127)}$
	100	> 3600	38.180	> 3600	> 3600	$279.840_{(462.799)}$	$0.465_{(0.845)}$
28	80	0.060	0.050	0.055	$0.015_{(0.033)}$	$0.020_{(0.035)}$	$0.010_{(0.010)}$
	85	0.065	0.050	0.050	$0.095_{(0.125)}$	$0.100_{(0.100)}$	$0.010_{(0.018)}$
	90	> 3600	0.050	0.060	$0.355_{(0.455)}$	$0.255_{(0.298)}$	$0.025_{(0.040)}$
	95	> 3600	> 3600	0.060	$10.335_{(13.965)}$	$1.920_{(2.071)}$	$0.060_{(0.060)}$
	98	> 3600	> 3600	> 3600	$1228.465_{(1285.085)}$	$91.230_{(504.219)}$	$0.200_{(0.238)}$
	100	> 3600	> 3600	> 3600	> 3600	> 3600	$1.060_{(1.225)}$
35	80	0.085	0.080	0.085	$0.055_{(0.149)}$	$0.015_{(0.020)}$	$0.010_{(0.020)}$
	85	0.100	0.085	0.090	$0.110_{(0.250)}$	$0.060_{(0.085)}$	$0.020_{(0.034)}$
	90	0.145	0.090	0.090	$0.580_{(0.789)}$	$0.260_{(0.495)}$	$0.030_{(0.039)}$
	95	> 3600	> 3600	0.090	$10.570_{(18.685)}$	$2.140_{(2.617)}$	$0.115_{(0.144)}$
	98	> 3600	> 3600	> 3600	> 3600	$3028.865_{(3114.055)}$	$0.365_{(0.449)}$
	100	> 3600	> 3600	> 3600	> 3600	> 3600	$0.735_{(1.336)}$

Table 4. Results on simple scheduling problems

The `cumulative` Constraint. The `cumulative` constraint has the form

$$\texttt{cumulative}([O_1, \ldots, O_m], [D_1, \ldots, D_m], [R_1, \ldots, R_m], L)$$

which solves a simple scheduling problem of m tasks and one kind of resources [2].

The new performance figures of E-GENET on problems with different resources utilization percentages (3 problems for each percentage) are shown in table 4.

Its performance on 8 job-shop problems [17, 13] of scheduling n jobs on m machines are listed in table 5. Since E-GENET solves *satisfaction* problems but not *optimization* problems, we fix the time period T and try to find a feasible solution. Problems `swv01` — `swv05` are from [21] and problems `abz7`— `abz9` are from [1].

problem	T	E-GENET CPU Time (sec)
`swv01`	1418	89.545 (105.123)
`swv02`	1491	91.865 (92.623)
`swv03`	1398	142.065 (154.669)
`swv04`	1493	91.765 (111.470)
`swv05`	1448	89.965 (128.378)
`abz7`	665	848.910 (1150.691)
`abz8`	670	3216.785 (3300.865)
`abz9`	686	15135.747 (15291.765)

Table 5. Results on job-shop problems

The `among` Constraint. The `among` constraint has five variants. The first and second variants are

$$\texttt{among}(N, [X_1, \ldots, X_s], [C_1, \ldots, C_s], [V_1, \ldots, V_m])$$

$$\texttt{among}([Low, Up], [X_1, \ldots, X_s], [C_1, \ldots, C_s], [V_1, \ldots, V_m])$$

where $N, X_1, \ldots, X_s$ are variables and Low, $Up, C_1, \ldots, C_s$, $V_1, \ldots, V_m$ are natural numbers. They specify that exactly N (for the first variant) or at least Low and at most Up (for the second variant) terms among $X_1 + C_1, \ldots, X_s + C_s$ having values in the value list $[V_1, \ldots, V_m]$. Figures 1 and 2 show the representations in E-GENET. In both cases, intermediate node a stores the number of terms currently taking their values in the value list. The constraint is then reduced into an equality constraint and the assignment scheme [12] of initial penalty values is $-|a - N|$.

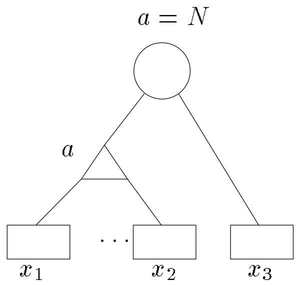
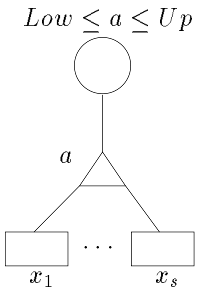


Fig. 1. The representation for the first variant of the `among` constraint

The other three variants are more complicated versions of `among`, but they can all be transformed into a number of `among` constraints of variants 1 and 2. This is exactly how they are handled in the E-GENET implementation.

We conduct experiments on two sets of benchmark problems: the N-queens problems and the channel assignment problems. For the N-queens problems, we use a variable with domain $\{0, 1\}$ to indicate if there is a queen on a square on the chessboard. For each column or row of the chessboard, we have

$$\texttt{among}(1, [\text{variables of the column or row}], [0, \ldots, 0], [1])$$



a	$\delta_{Low \leq a \leq Up}$
$\vdots$	$\vdots$
$Low - 2$	-2
$Low - 1$	-1
Low	0
$\vdots$	$\vdots$
Up	0
$Up + 1$	-1
$Up + 2$	-2
$\vdots$	$\vdots$

Fig. 2. The representation for the second variant of the `among` constraint

and for each diagonal of the chessboard,

$$\texttt{among}([0,1], [\text{variables of the diagonal}], [0,\ldots,0], [1]).$$

Table 6(a) shows results of problems with 10 to 100 queens.

No. of queens	CHIP CPU Time (sec)	E-GENET CPU Time (sec)
10	0.050	0.010 (0.009)
20	56.410	0.125 (0.129)
30	> 3600	0.270 (0.255)
40	> 3600	0.460 (0.493)
50	> 3600	1.540 (1.580)
60	> 3600	1.125 (1.256)
70	> 3600	2.788 (2.792)
80	> 3600	2.454 (2.209)
90	> 3600	4.053 (4.136)
100	> 3600	4.905 (4.660)

(a)

Problem	$n \times m$	E-GENET CPU Time (sec)
1	4×11	0.000 (0.000)
2	25×73	8.280 (8.309)
3	21×381	5.280 (5.412)
4	21×533	126.865 (130.493)
5	21×533	180.595 (179.053)
6	21×221	42.425 (44.905)
7	21×309	396.805 (399.256)
8	21×309	288.665 (285.009)

(b)

Table 6. Results on (a) binary N-queens and (b) channel assignment problems

The channel assignment problem [8] involves assigning channels or frequencies to radio cells in a cellular radio network. A problem of n-cell network consists of an demand vector $D = (d_1, \ldots, d_n)$ and an $n \times n$ symmetric compatibility matrix C. Each d_i is the number of frequencies required by cell i and each c_{ij} in C represents the minimum separation distance between any two frequencies assigned to cell i and j. In each problem, we use $n \times m$ Boolean variables. A variable v_{ij} equals 1 if frequency f_j is assigned to cell i and 0 otherwise. For each c_{ii} in C, we have

$$\texttt{among}([0, 1, c_{ii}, d_i, d_i], [v_{i1}, \ldots, v_{im}], [0, \ldots, 0], [1])$$

and for each c_{ij} where $i < j$, for all $k \in \{0, \ldots, m - c_{ij}\}$,

$$\texttt{among}([0,1], [v_{i,k+1}, v_{j,k+1}, v_{i,k+2}, v_{j,k+2}, \ldots, v_{i,k+c_{ij}}, v_{j,k+c_{ij}}], [0, \ldots, 0], [1]).$$

Based on the problems in [8], we randomly generate 8 testing examples and the results are shown in table 6(b). CHIP can solve only the smallest problem (4×11) in 0.08 seconds.

The `diffn` Constraint. The `diffn` constraint specifies that there is no overlapping between m objects, a set of n-dimensional rectangles. There are five variants of the constraint and the basic form is

$$\texttt{diffn}([O_{11}, \ldots, O_{1n}, L_{11}, \ldots, L_{1n}], \ldots, [O_{m1}, \ldots, O_{mn}, L_{m1}, \ldots, L_{mn}]).$$

The point $(O_{i1}, \ldots, O_{in})$ is the origin of the object i and the length of the object in each dimension j is given by L_{ij}.

The `diffn` constraint is handled as follows. For each unit space in the n-dimensional space, there is an intermediate node that holds the current number of objects occupying this unit space. Hence the constraint reduces to a set of inequality constraints $a_i \leq 1$.

We test our implementation with the Mystery Shopper problem, a real-life scheduling problem posted by the first author in the newsgroup comp.constraint in September 1996. A well-known cosmetic company wants to evaluate the performance of her 35 salesladies, who station at the company's counters at 19 department stores in 4 different geographical regions. For this purpose, the company has hired 40 secret agents of different categories (age, professions, *etc*)to disguise as shoppers to visit the salesladies. Each shopper has to visit 3 to 4 salesladies and each saleslady has to be visited by 4 shoppers. Two of these visits must be in the first two weeks. The visits must be scheduled in such a way that each saleslady must be visited by shoppers of different varieties, and that shoppers should visit salesladies in different geographic regions. The assessment period is 4 weeks.

We model each visit of a shopper by a unit square. Square s_{ij} denotes the j-th visit of shopper s_i. We then need to place 140 squares into a 35×4 table. That square s_{ij} is placed at position (p, q) of the table means that the saleslady p is visited in the q-th week by shopper s_i as his j-th visit. A `diffn` constraint is used to enforce that the squares do not overlap. We also impose some other constraints, such as `element` and `alldifferent`, to meet the rest of the requirements.

Using this modeling, the median execution time and the mean execution time with the best and worst runs removed for E-GENET are 0.663 second and 0.725 second respectively while CHIP does not terminate within 10 hours. H. Simonis (private communication) later developed a CHIP program for solving the problem using another modeling with only the `among` constraint. This program is also able to solve the problem almost instantly. The efficiency of Simonis's program is derived from a domain specific value ordering heuristics tailor-made for this problem.

An advantage of using E-GENET is that we need only to concentrate on modeling by providing a purely *declarative* statement of the problem, without having to worry about variable and value ordering heuristics, which are algorithmic aspects of constraint programming. This exercise confirms the suitability of embedding E-GENET in a constraint programming language and brings us one step towards pure declarative constraint programming.

To increase the efficiency for placement problems in which objects can be flipped or rotated, we provide one more variant of the `diffn` constraint:

$$\texttt{diffn}([O_{11}, \ldots, O_{1n}, R_1, L_{11}, \ldots, L_{1n}], \ldots, [O_{m1}, \ldots, O_{mn}, R_m, L_{m1}, \ldots, L_{mn}])$$

where $R_i \in \{1, \ldots, n!\}$ indicates the orientation of the object i. The lengths of the object i in different dimensions are given by a permutation of $[L_{i1}, \ldots, L_{in}]$ determined by R_i.

We use this variant to solve three-dimensional packing problems in E-GENET. Each problem consists of packing a number of parallelepipeds into a large cube, which may be partially filled. Three problems are generated for each setting. In the CHIP program in [3], besides a `diffn` constraint, three redundant `cumulative` constraints are added to improve the performance. However, CHIP cannot solve any of these problems within one hour. The E-GENET version is composed of a single `diffn` constraint. The results are shown in table 7.

The `cycle` Constraint. The `cycle` constraint is a very complex but powerful constraint. It has six variants but we would describe just the first one,

$$\texttt{cycle}(N, [S_1, \ldots, S_m])$$

where $N, S_1, \ldots, S_m$ are all variables. The constraint holds if three conditions are true. First, for all S_i, $S_i \in \{1, \ldots, m\}$. Second, $S_i \neq S_j$ if $i \neq j$. Third, C_i is the set of integers defined by (1) $i \in C_i$ and (2) if $j \in C_i$ then $S_j \in C_i$. The constraint can be seen as finding N distinct cycles in a directed graph of m nodes where each node is totally visited once. Initially, the list $[S_1, \ldots, S_m]$ specifies the topology of the directed graph. The domain of S_i is the outbundle of node i. $S_i = j$ if edge (i, j) is in one of the cycles.

In E-GENET, the first condition is warranted by setting up domains for the variables appropriately. An `alldifferent` constraint is used to enforce the second condition. An intermediate node a stores the number of distinct sets among all C_i's. Thus the constraint $a = N$ can be used to cover the last condition.

Benchmark problems for the `cycle` constraint are a number of Hamiltonian cycle problems. Each problem consists of finding a cycle in an undirected graph where all nodes are visited exactly once. We randomly generate the graphs with different number of nodes and average degree of a node. Three graphs are generated for each setting and the results are enlisted in table 8.

4 Concluding Remarks

In this paper we present a comprehensive library of constraints for E-GENET. We show that these constraints, elementary and global, can be effectively and effi-

Size of the cube	No. of parallel-epipeds	Percentage of volume occupied	E-GENET CPU Time (sec)		
5^3	14	80	0.070(0.068)	0.050(0.053)	0.040(0.049)
		85	0.090(0.083)	0.085(0.085)	0.040(0.045)
		90	0.285(0.383)	0.235(0.260)	0.210(0.233)
		95	1.555(3.045)	1.410(5.174)	0.405(0.904)
		98	> 3600	> 3600	> 3600
		100	> 3600	> 3600	> 3600
	16	80	0.045(0.059)	0.045(0.054)	0.040(0.050)
		85	0.095(0.090)	0.085(0.130)	0.075(0.095)
		90	0.410(0.475)	0.335(0.398)	0.190(0.188)
		95	5.435(9.411)	3.610(13.871)	1.660(4.245)
		98	> 3600	> 3600	103.545(119.373)
		100	> 3600	> 3600	> 3600
	18	80	0.065(0.065)	0.055(0.059)	0.035(0.043)
		85	0.115(0.130)	0.105(0.111)	0.065(0.074)
		90	0.200(0.235)	0.195(0.231)	0.155(0.174)
		95	3.960(5.356)	1.470(1.601)	0.930(1.465)
		98	> 3600	29.950(70.121)	6.725(27.143)
		100	> 3600	> 3600	> 3600
10^3	112	80	1.635(1.649)	1.500(1.514)	1.315(1.324)
		85	3.040(2.954)	2.060(2.044)	1.945(2.145)
		90	13.135(15.149)	10.935(11.464)	9.565(9.930)
		95	> 3600	789.395(1035.041)	148.810(274.539)
		98	> 3600	> 3600	> 3600
		100	> 3600	> 3600	> 3600
	128	80	1.570(1.898)	1.425(1.550)	1.115(1.495)
		85	3.805(3.761)	3.605(4.219)	2.565(3.279)
		90	9.515(10.938)	8.205(8.163)	6.110(5.941)
		95	737.340(862.386)	340.750(459.226)	151.495(258.745)
		98	> 3600	> 3600	2393.955(1527.629)
		100	> 3600	> 3600	> 3600
	144	80	1.720(1.765)	1.440(1.504)	1.085(1.299)
		85	3.310(3.171)	2.805(3.326)	2.765(3.331)
		90	8.960(8.996)	7.040(7.308)	5.990(6.720)
		95	175.090(335.349)	106.295(131.776)	37.505(52.293)
		98	> 3600	3384.175(1993.694)	503.015(645.385)
		100	> 3600	> 3600	> 3600

Table 7. Results on three-dimensional packing problems

node no.	average degree	CHIP CPU Time (sec)			E-GENET CPU Time (sec)		
100	11	122.530	0.640	0.630	0.045(0.053)	0.075(0.074)	0.065(0.064)
200	12	4.490	4.480	4.430	0.310(0.339)	0.335(0.313)	0.295(0.328)
300	13	14.240	14.140	14.080	0.835(0.963)	1.080(1.521)	0.965(0.913)
400	14	> 3600	> 3600	41.810	1.120(1.181)	1.555(1.620)	1.645(1.830)
500	15	69.940	63.080	62.780	2.810(2.924)	2.435(3.302)	2.300(2.549)
600	16	> 3600	128.870	108.120	4.415(4.651)	3.105(4.754)	3.885(6.179)
700	17	> 3600	284.770	174.400	5.880(11.361)	4.880(7.006)	6.455(7.421)
800	18	> 3600	> 3600	252.220	7.860(8.256)	7.385(11.385)	9.250(12.059)
900	19	> 3600	> 3600	362.270	12.210(17.938)	9.145(15.991)	8.575(13.946)
1000	20	> 3600	> 3600	497.040	9.030(16.190)	10.085(11.735)	10.875(15.455)

Table 8. Results on Hamiltonian cycle problems

ciently implemented in E-GENET under a uniform framework. The performance figures in our benchmarks confirm empirically that the E-GENET constraints compare favorably against those in commercial implementations.

E-GENET is based on *incomplete* local search, while commercial constraint systems are based on *complete* tree search. Comparison between the two approaches is worthwhile and useful, provided that the difference is taken into account. In many real-life large scale and difficult applications, completeness is unimportant. What is important is that a (any) solution is generated in a timely fashion. Our results confirm that local search is a competitive approach for this kind of problems.

Global constraints in CHIP not only make it easier to model complicated problems but also much improve the efficiency of problem solving. Learning from the success of CHIP and by including global constraints in the E-GENET library, we bridge the gap between constraint solving and constraint programming for E-GENET so that it is ready for deployment in real-life applications.

References

1. J. Adams, E. Balas, and D. Zawack. The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science*, 34:391–401, 1988.
2. A. Aggoun and N. Beldiceanu. Extending CHIP in order to Solve Complex Scheduling and Placement Problems. *Journal of Mathematical and Computer Modelling*, 17(7):57–73, 1993.
3. N. Beldiceanu and E. Contejean. Introducing Global Constraints in CHIP. *Journal of Mathematical and Computer Modeling*, 20(12):97–123, 1994.
4. P. Codognet and D. Diaz. Compiling Constraints in `clp(FD)`. *Journal of Logic Programming*, 27(3):185–226, 1996.
5. 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 Twelfth National Conference on Artificial Intelligence*, pages 325–330, 1994.
6. M. Dincbas, H. Simonis, and P. Van Hentenryck. Solving Car Sequencing Problem in Constraint Logic Programming. In *Proceedings of Eighth European Conference on Artificial Intelligence*, pages 290–295, 1988.

7. M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *Proceedings of International Conference on Fifth Generation Computer Systems*, pages 693–702, December 1988.
8. N. Funabiki and Y. Takefuji. A Neural Network Parallel Algorithm for Channel Assignment Problems in Cellular Radio Network. *IEEE Transactions on Vehicular Technology*, 41(4), 1992.
9. ILOG. *ILOG SOLVER: Reference Manual Version 4.0*, 1997.
10. V. Kumar. Algorithms for Constraint-Satisfaction Problems: A Survey. *AI Magazine*, 13(1):32–44, 1992.
11. J.H.M. Lee, H.F. Leung, and H.W. Won. Extending GENET for Non-Binary CSP's. In *Proceedings of Seventh International Conference on Tools with Artificial Intelligence*, pages 338–343, Washington DC, USA, November 1995. IEEE Computer Society Press.
12. J.H.M. Lee, H.F. Leung, and H.W. Won. Towards a More Efficient Stochastic Constraint Solver. In *Proceedings of Second International Conference on Principles and Practice of Constraint Programming*, LNCS 1118, pages 338–352, Cambridge, Massachusetts, USA, August 1996. Springer–Verlag.
13. J.K. Lenstra and A.H.G. Rinnooy Kan. Computational Complexity of Discrete Optimization Problems. *Annals of Discrete Mathematics*, 4:121–140, 1979.
14. A.K. Mackworth. Consistency in Networks of Relations. *AI Journal*, 8(1):99–118, 1977.
15. 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*, 58:161–205, 1992.
16. P. Morris. The breakout method for escaping from local minima. In *Proceeding of the 11th National Conference on Artificial Intelligence*, pages 40–45, 1993.
17. J.F. Muth and G.L. Thompson. *Industrial Scheduling*. Prentice Hall, Engleclood Cliffs, N.J., 1963.
18. H. Simonis. Modelling Machine Set-up Times in CHIP. Technical Report COSY/TR/95-2, COSYTEC SA, 1995.
19. H. Simonis, N. Beldiceanu, P. Charlier, and P. Kay. A Model of TACT. Technical Report COSY/TR/95-5, COSYTEC SA, 1995.
20. H. Simonis and T. Corenlissens. Modelling Producer/Consumer Constraints. Technical Report COSY/TR/94-2, COSYTEC SA, 1994.
21. R.H. Storer, S.D. Wu, and R. Vaccari. New Search Spaces for Sequencing Instances with Application to Job Shop Scheduling. *Management Science*, 38:1495–1509, 1992.
22. The COSYTEC Team. *CHIP V4.1 User Manuals*, 1994.
23. J. Thornton and A. Sattar. Applied partial constraint satisfaction using weighted iterative repair. In *Proceedings of the Tenth Australian Joint Conference on Artificial Intelligence*, pages 57–66, 1997.
24. P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. The MIT Press, 1989.
25. J.P. Walser. Solving linear pseudo-boolean constraint problems with local search. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 269–274, 1997.
26. J.P. Walser, R. Iyer, and N. Venkatasubramanyan. An integer local search method with application to capacitated production planning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 1998.