Towards a More Efficient Stochastic Constraint Solver

Jimmy H.M. Lee, Ho-fung Leung and Hon-wing Won

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

Abstract. E-GENET shows certain success on extending GENET for non-binary CSP's. However, the generic constraint representation scheme of E-GENET induces the problem of storing too many penalty values in constraint nodes and the min-conflicts heuristic is not efficient enough on some problems. To overcome these two weaknesses and further improve the performance, we propose several modifications. All of them together can boost the efficiency of E-GENET without resorting to modifying the underlying network model or the convergence procedure in an *ad hoc* manner. The performance of modified E-GENET also compares well against that of CHIP.

1 Introduction

Many problems in artificial intelligence and computer science in general can be formulated as *constraint satisfaction problems* (CSP's). Efficient algorithms for solving CSP's are thus very useful. In 1992, Minton *et al.* published a paper on a new approach for solving CSP's. The approach is known as heuristic repair method or iterative repair method [8]. 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 SPARCstation1 [8].

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 [2], especially non-binary CSP's, since reassigning one variable at each step usually cannot reduce number of constraint violations.

In a previous paper [7], we present the E-GENET, which bases on iterative repair approach, features a generic representation scheme for general constraints and adopts the heuristic learning rule from GENET [2]. Constraints ranging from disjunctive constraints to non-linear constraints to symbolic constraints can now be handled. However, being a first step for solving non-binary CSP's, E-GENET has two insufficiencies. For a complicated constraint, there may be a large number of penalty values stored in the corresponding constraint node. Besides, the underlying principle of E-GENET, the min-conflicts heuristic in the iterative repair approach, cannot provide enough information to guide the search efficiently in some non-binary CSP's and hence the performance is not satisfactory.

In this paper, we describe several modifications to E-GENET such as a new type of nodes to deal with the problem of large constraint nodes and a novel assignment scheme of initial penalty values to improve the effectiveness of the min-conflicts heuristic on non-binary CSP's. We have also implemented an prototype to show the feasibility and efficiency of our proposal. The modified E-GENET compares well against CHIP [5] in most of the CSP's tested.

The rest of this paper is organized as follows. Section 2 briefly reviews E-GENET. In section 3, we explain the inadequacies of E-GENET. Section 4 describes the proposed modifications. Benchmarking results and related work are presented in section 5 and 6 respectively. Section 7 summarizes our contributions and sheds light on future work.

2 Brief Review of E-GENET

E-GENET extends GENET for general, binary and non-binary, constraint handling by a generic constraint representation scheme. It consists of a network model and a convergence procedure based on min-conflicts heuristic in iterative repair approach and the learning heuristic of GENET.

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 for $3y - z = 4$ is connected to relating 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 $z = 4$.

For any constraint $c(x_{i_1}, \ldots, x_{i_n})$, each combination (or tuple) $(v_1, \ldots, v_n)$ of possible values from domains of $x_{i_1}, \ldots, x_{i_n}$ is given a *penalty value* $\delta_{c(v_1,\ldots,v_n)}$. These penalty values, (conceptually) 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.

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

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

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

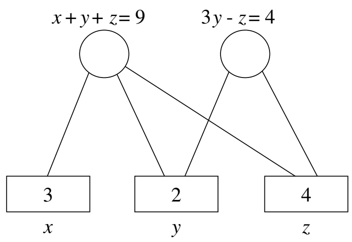


Fig. 1. An example network generated by E-GENET

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, 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 [2], we define a *repair* to be a variable update. Variables in the network are updated asynchronously 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. The *convergence procedure* is summarized in figure 2.

3 Inadequacies of E-GENET

3.1 Cumbrous Constraint Node

Consider the constraint $13x_1+4x_2+5x_3+9x_4 = 7y_1+8y_2$ where all variables have the domain $\{1, \ldots, 100\}$. Since there are 100^6 possible tuples, we have to store 10^{12} penalty values in the corresponding constraint node and this is impractical.

repeat
  update all variable nodes asynchronously until no repair
  if (sum of inputs of values in current assignment is zero)
    terminate with success
  else
    activate heuristic learning
until (time limit or maximum number of cycles is exceeded)

Fig. 2. The convergence procedure of E-GENET

If we break down the constraint into $13x_1 + 4x_2 + 5x_3 = T$ and $T + 9x_4 = 7y_1 + 8y_2$ by using a new variable T, the domain of T after pruning would be $\{22, \ldots, 1491\}$. The number of possible tuples for these two constraints is then $2 \times 1470 \times 100^3$, *i.e.* 2.94×10^9. Although we can successfully lower the storage requirement, the performance would be affected by the addition of a new variable of relatively large domain. Further decomposition of the two constraints can cut the space requirement to a greater extent but induces severe drawback in performance. So this is not a good method to solve the problem. What we need is something that has properties similar to T but does not increase the number of variables.

3.2 Inefficiency of the min-conflicts heuristic

In this section, we extend the model in [8] to investigate the efficiency of E-GENET. Consider a CSP with variables $x_1, \ldots, x_n$, where each x_i has k possible values and involves exactly in c m-ary constraints. Assume that there is only one solution $(v_1, \ldots, v_n)$ and the randomly generated initial assignment has d variables assigned different from the solution. Denote the set of these d variables as Var. For any constraint $c'(x_{i_1}, \ldots, x_{i_m})$ in the CSP, if $\exists j$ such that $S_{x_{i_j}} \neq v_{i_j}$, let the probability of the constraint being violated be p.

Randomly choose a variable $x_k \in Var$ for repairing (the case for $x_k \notin Var$ is similar). For any one of the $k-1$ incorrect values ($\neq v_k$) of x_k, since all c related constraints have a probability p of being violated, the expected number of constraint violations is pc and hence the expected input to these values is $-pc$.

Consider the correct value of x_k. For an arbitrary constraint depending on the variable, the probability that all other $m-1$ variables are not elements of Var is $\binom{n-d}{m-1}/\binom{n-1}{m-1}$. The probability of the constraint being violated is thus $\left(1 - \binom{n-d}{m-1}/\binom{n-1}{m-1}\right) p$ and the expected input to the correct value of x_k is given by $-\left(1 - \binom{n-d}{m-1}/\binom{n-1}{m-1}\right) pc$.

From the expected inputs, we can see that the probability of making an incorrect repair would be decreased if c increases or d shrinks. In other words, if number of constraints is small or d is very near to n ($d > n - m$) at the beginning, the efficiency of E-GENET may be very low.

This result can also be observed from the benchmarking results of E-GENET in our previous paper [7]. We find that in some non-binary CSP's like systems of linear equations and cryptarithmetic problems, the performance of E-GENET is worse than that of CHIP. Use the problem in figure 3 as an example. Here, $m = n = c = 4$ and p is equal to 1 approximately. If we try to repair a variable that is assigned incorrectly at start, in almost all cases, the input to any one of incorrect values of the variable is -4 and that to the correct value is -4 as well $\left(\binom{n-d}{m-1}/\binom{n-1}{m-1} = 0\right)$. Hence, variables are updated randomly and the network would wander over all possible states aimlessly until there are sufficient information provided by learning.

$$\begin{cases} 691x_1 + 81x_2 + 22x_3 + 629x_4 = 7007 \\ 519x_1 + 147x_2 - 971x_3 - 710x_4 = -8726 \\ 841x_1 - 527x_2 + 948x_3 - 589x_4 = -4357 \\ 899x_1 + 343x_2 - 877x_3 + 531x_4 = 4571 \\ 0 \leq x_1, x_2, x_3, x_4 \leq 10 \end{cases}$$

Fig. 3. A system of linear equations

4 Modifications

To overcome the two weaknesses of E-GENET, we propose four modifications. All of them together can boost the performance of E-GENET without resorting to modifying the underlying network model or the convergence procedure in an *ad hoc* manner.

4.1 Intermediate Node

The first modification is the introduction of a new type of nodes called *intermediate nodes* to address the problem of cumbrous constraint nodes. Figure 4 shows a general representation for a constraint with this modification.

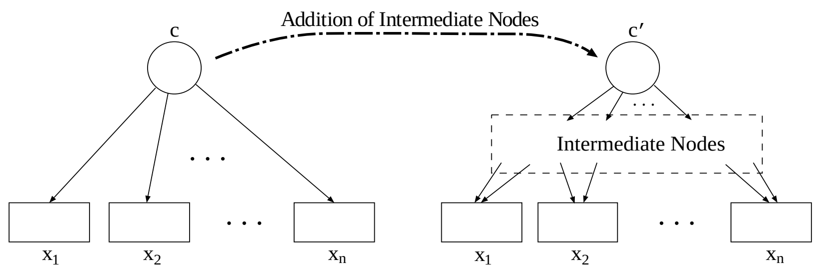


Fig. 4. General representation for a constraint after the addition of intermediate nodes

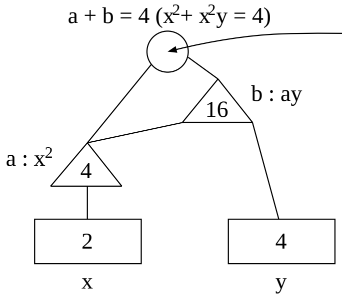
Formally, a representation for a constraint on n variables $\{x_1, \ldots, x_n\}$ is a directed acyclic graph (*dag*) with at least $n+1$ nodes. One of the nodes is called constraint node, denoted by c'. We have $indegree(c') = 0$ and $outdegree(c') \geq 1$. [2] n

[2] $indegree(Y)$ is the number of edges entering node Y and $outdegree(Y)$ is that leaving.

of the other nodes are called variable nodes. There exists a one-one mapping between variables and variable nodes. Thus we relax terminology and give a variable and its corresponding variable node the same name. For all x_i, $indegree(x_i) \geq 1$ and $outdegree(x_i) = 0$.

All other nodes, not in $\{c', x_1, \ldots, x_n\}$, are intermediate nodes. Let the set of intermediate nodes be $\{f_1, \ldots, f_m\}$. Each intermediate node f_i has the properties that $indegree(f_i) \geq 1$ and $outdegree(f_i) \geq 1$. Define $outbundle(f_i)$ as the set $\{a_1, \ldots, a_p\}$ such that for each a_j in the set, there is an edge from f_i to a_j. Intermediate node f_i is associated with a function defined on its outbundle, $F_i(a_1, \ldots, a_p)$. The state S_{f_i} of f_i is the value $F_i(S_{a_1}, \ldots, S_{a_p})$ and f_i's domain, $dom(f_i)$, is the range of the function F_i. In constraint node c', we store penalty values for combinations of values from domains of nodes in $outbundle(c')$ instead.

Consider the constraint $x^2+x^2y = 4$. Figure 5 shows a possible representation for this constraint. The intermediate node a is associated with the function x^2 and currently has the value (state) 4. If $x \in \{1, 2\}$ and $y \in \{3, 4\}$, after the addition of nodes a and b, the content of the constraint node is changed as shown in the figure.



x	y	$\delta_{x^2+x^2y=4}$
1	3	0
1	4	-1
2	3	-1
2	4	-1

$\Longrightarrow$

a	b	$\delta_{a+b=4}$
1	3	0
1	4	-1
4	12	-1
4	16	-1

Fig. 5. An example representation with intermediate nodes

Let the outbundle for constraint node c' of the constraint $c(x_1, \ldots, x_n)$ be $\{f_1, \ldots, f_p\}$. Since for each combination $(v_1, \ldots, v_n)$ of possible values from domains of $x_1, \ldots, x_n$, there is a corresponding tuple $(S_{f_1}, \ldots, S_{f_p})$, a mapping $H : x_1 \times \ldots \times x_n \to B$ where $B \subseteq dom(f_1) \times \ldots \times dom(f_p)$ exists and range of $H = B$ (surjective). Each tuple in B is given a penalty value and all these penalty values form the content of c'.

There is one restriction on usage of intermediate nodes. For each $(u_1, \ldots, u_p)$ in B, all tuples in $H^{-1}(u_1, \ldots, u_p)$ must be either satisfying or violating the constraint unanimously. If all these tuples satisfy the constraint, the initial penalty value for $(u_1, \ldots, u_p)$ is 0. Otherwise, it is set to -1. The heuristic learning is similar to that in the original E-GENET. For example, a plausible learning rule is $\delta_{c'(S_{f_1},\ldots,S_{f_p})} := \delta_{c'(S_{f_1},\ldots,S_{f_p})} - (\delta_{c'(S_{f_1},\ldots,S_{f_p})} < 0)$. [3]

The most essential usage of intermediate nodes is to reduce the number of

[3] $<$ is a boolean function returning 1 if the comparison is true and 0 otherwise.

nodes in outbundle of the constraint node and the size of domains of these nodes, i.e. the size of B. Although we allow any levels of intermediate nodes, one is sufficient to meet this objective.

Theorem Given a representation M for a constraint $c(x_1, \ldots, x_n)$, there exists another representation M' such that all paths in M' are of length ≤ 2 and the content of the constraint node in M' is the same as that in M.

The philosophy behind intermediate nodes is to divide tuples of a constraint into several groups (using the function H). Each group (a tuple in B) is given only one penalty value and during learning, we would treat a group as a unit and penalize a whole group.

Consider again the constraint $13x_1 + 4x_2 + 5x_3 + 9x_4 = 7y_1 + 8y_2$. It can be represented as shown in figure 6. Intermediate nodes a and b represent expressions at both sides of the constraint respectively. Each distinct combination (v_a, v_b) calculated with possible values from domains of $x_1, x_2, x_3, x_4, y_1, y_2$ is given one penalty value. When heuristic learning is activated, if $S_a \neq S_b$, we would decrease the penalty value $\delta_{(a=b)(S_a,S_b)}$ by 1. The pair $(60, 50)$ in the new constraint node, for example, represents the set of tuples $\{(1, 1, 5, 2, 6, 1),$ $(1, 2, 6, 1, 6, 1)$, $(1, 6, 1, 2, 6, 1)$,$(1, 7, 2, 1, 6, 1)$,$(2, 5, 1, 1, 6, 1)\}$ in the original node. Updating the penalty value of the pair is equivalent to performing the same operation on penalty values of all tuples in the set.

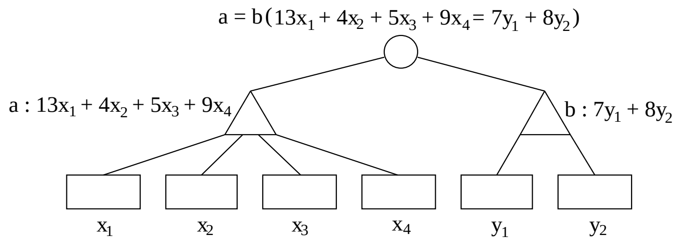


Fig. 6. A possible representation for the constraint $13x_1 + 4x_2 + 5x_3 + 9x_4 = 7y_1 + 8y_2$

Under this representation, the storage requirement for the constraint becomes 4×10^6 (compared to 10^{12} in the original one). If we divide all possible tuples into fewer groups, we can save more space. However, performance would be affected. Decreasing the penalty value of a tuple during learning is to reduce the probability that this tuple is assigned to corresponding variables again. With grouping, changing one penalty value would influence probabilities of all tuples in the group and this effect may not be desirable. To avoid any decrease in performance, we carefully design a suitable grouping for each type of constraints. Moreover, in practice, only penalty values of tuples that have been penalized during learning would be stored. The others would be computed on demand.

Besides reducing size of constraint node, intermediate node has two more advantages:

Eliminating Common Subexpression: In each constraint node, if we do not store all penalty values directly and, instead, derive some storage scheme to reduce storage requirement, we usually need to do constraint checks. Intermediate nodes can be used to store the results of common subexpressions and eliminate the need of re-calculation. The node a in figure 7 removes redundant work on both inter- and intra-constraint common subexpression.

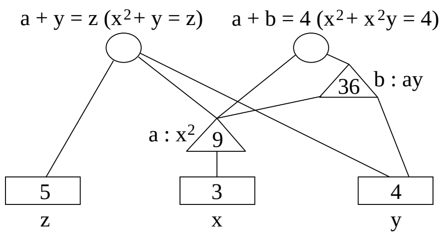


Fig. 7. An example network with the common subexpression x^2

Unifying Heuristic Learning: The heuristic learning rule is not fixed in original E-GENET. During learning, for any constraint $c(x_1, \ldots, x_n)$, if $\delta_{c(S_{x_1},\ldots,S_{x_n})} < 0$, users can choose a P where $\{(S_{x_1}, \ldots, S_{x_n})\} \subseteq P \subseteq \{(v_1, \ldots, v_k) | \delta_{c(v_1,\ldots,v_k)} < 0\}$ and decrease penalty values of all tuples in P by 1. As most advantages of this practice can be obtained by using intermediate nodes, heuristic learning rules can be unified with $P = \{(S_{x_1}, \ldots, S_{x_n})\}$.

Use the constraint $\texttt{atmost}(1, \{x_1, x_2, x_3, x_4\}, \{3\})$ as an example. This constraint states that at most one out of the four variables can take the value 3. We usually choose $P = \{(v_1, \ldots, v_k) | \delta_{c(v_1,\ldots,v_k)} < 0\}$. Constructing the representation as in figure 8, we can use the learning rule $\delta_{(a=0)(S_a)} := \delta_{(a=0)(S_a)} - (\delta_{(a=0)(S_a)} < 0)$ instead to get the same effect.

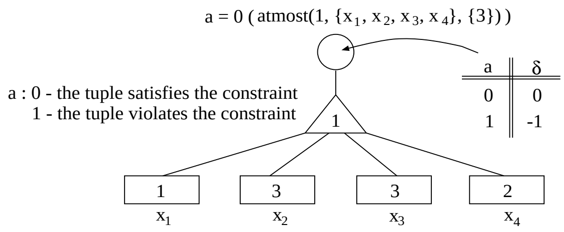


Fig. 8. The representation for the `atmost` constraint

4.2 New Assignment Scheme of Initial Penalty Values

E-GENET is based on the min-conflicts heuristic, which favors the value violating the minimum number of constraints. Before learning procedure is activated, the input to a value is $-n$ where n is number of constraints violated by assigning this value to the corresponding variable. E-GENET would then change the value of the variable to the one that has the maximum input.

If the network is trapped in a local minima, heuristic learning is invoked. This amounts to decreasing the penalty values of tuples violating some constraints. Assume that in a particular learning process, the tuple $(v_1, \ldots, v_n)$ of a constraint $c(x_1, \ldots, x_n)$ is penalized. This can be considered as adding a redundant constraint $\texttt{illegal}((x_1, \ldots, x_n), (v_1, \ldots, v_n))$, which specifies that $(x_1, \ldots, x_n) \neq (v_1, \ldots, v_n)$: decreasing the penalty value $\delta_{c(v_1,\ldots,v_n)}$ is effectively the same as combining the content of the constraint node with that of the `illegal` constraint.

$x_1 \ldots x_n$	new $\delta_{c(x_1,\ldots,x_n)}$ =	old $\delta_{c(x_1,\ldots,x_n)}$ +	$\delta_{\texttt{illegal}((x_1,\ldots,x_n),(v_1,\ldots,v_n))}$
$\vdots$	$\vdots$	$\vdots$	**0**
$v_1 \ldots v_n$	-2	-1	-1
$\vdots$	$\vdots$	$\vdots$	**0**

Hence decreasing penalty values can be interpreted as the accumulation of knowledge by introducing redundant constraints. It is well known that domain knowledge or some other information can be utilized to guide the search by adding appropriate redundant constraints. According to the the above discussion, we need not create new constraint nodes for these redundant constraints in the E-GENET model. What we need is simply a new assignment scheme of initial penalty values: for any constraint, initial penalty values of tuples satisfying the constraint are 0's, but those for prohibited tuples may be any negative integers determined by the user using his domain knowledge. [4] This resembles the fitness in "Evolutionary Model" of GENET [11].

4.3 Concept of Contribution

Consider a constraint $c(x_1, \ldots, x_n)$. If the current assignments of $x_1, \ldots, x_n$ are $v_1, \ldots, v_n$ respectively, $\delta_{c(v_1,\ldots,v_n)}$ is always used in computing any input $I_{x_i=v_i}$, where $1 \leq i \leq n$. However, this method is inadequate in some cases. Take the constraint $\texttt{atmost}(1, \{x_1, x_2, x_3, x_4\}, \{3\})$ and the tuple $(3, 4, 3, 3)$ as an example. That x_2 being assigned the value 4 does not contribute to the violation of the constraint. The penalty value of the tuple should not be added to the input $I_{x_2=4}$ to decrease the probability that x_2 takes the value 4. Thus, the definition of an input is given as:

$$I_{x=v} = \sum_{\forall c(x_{i_1},\ldots,x,\ldots,x_{i_k})} contribute_c(j, (S_{x_{i_1}}, \ldots, v, \ldots, S_{x_{i_k}})) \times \delta_{c(S_{x_{i_1}},\ldots,v,\ldots,S_{x_{i_k}})}$$

where the j^{th} argument of $(S_{x_{i_1}}, \ldots, v, \ldots, S_{x_{i_k}})$ is v; $contribute_c(i, (v_1, \ldots, v_n))$ is a function returning a value between 0 and 1. The magnitude of the value shows the contribution of v_i to the tuple $(v_1, \ldots, v_n)$ with respect to the constraint c.

[4] It should be noted that any finite domain constraints can be expressed as a conjunction of `illegal` constraints.

4.4 Learning Heuristic

As now the initial penalty value of a tuple can be much smaller than -1, the landscape of the search space becomes comparably rough. A penalty amount of -1 may not be sufficient to unstable the network. So if the same tuple of a constraint is penalized consecutively, penalty amount is increased exponentially at each time of learning $(-1, -1, -2, -4, -8, \ldots)$.

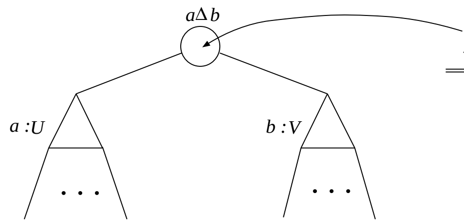
5 Benchmarking Results

To illustrate the feasibility and effectiveness of the modifications, we have implemented several types of constraints and test each of them on different problems. We compare our result with that of a constraint logic programming language Cosytec CHIP version 4.1.0 [9], which uses traditional constraint propagation and backtracking tree search for constraint solving.

All benchmarking is performed on a SUN SPARCstation 10 model 30. Timing (including network construction) and number of repairs results for both versions of E-GENET are median of 10 runs. For each problem, median number of penalty values stored in the modified E-GENET is also shown. A "–" symbol means that the execution fails due to either execution time exceeded or memory exhaustion.

5.1 Linear Arithmetic Constraint

A linear arithmetic constraint is a constraint of the form $U \triangle V$, where U and V are linear arithmetic expressions and $\triangle \in \{=, \neq, <, \leq, >, \geq\}$. The representation of the constraint is shown in figure 9. Intermediate nodes a and b are used to hold the current values of U and V. The initial penalty values for violating tuples are mainly based on the difference between values of a and b.



Δ	penalty value assignment scheme
$=$	$-\lvert a - b \rvert$
$\neq$	-1
$<$	$b - a - 1$
$\leq$	$b - a$
$>$	$a - b - 1$
$\geq$	$a - b$

Fig. 9. The representation for a linear arithmetic constraint

Five traditional benchmark programs [3] have been used to show the ability of the modified E-GENET on solving linear equation problems. `send`, `donald` and `crypta` are cryptarithmetic problems of different size. `eq10` and `eq20` are systems of 10 and 20 linear equations respectively.

Since constraint propagation alone can solve the cryptarithmetic problems with little backtracking, CHIP outperforms both versions of E-GENET significantly in most cases. However, we can observe that the proposed modifications

Problem	CHIP CPU Time (sec)	Original E-GENET CPU Time (sec)	Modified E-GENET CPU Time (sec)	penalty values stored
`send`	0.010	0.300	0.085	53
`donald`	0.050	2.895	0.380	166
`crypta`	0.050	3.400	1.040	629
`eq10`	0.140	–	0.045	0
`eq20`	0.210	–	0.040	0

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

can improve the performance of E-GENET. The modified E-GENET is more efficient than CHIP in the problems `eq10` and `eq20`, which cannot be solved within 10 minutes by the original version.

5.2 Atmost Constraint

The $\texttt{atmost}(N, Var, Val)$ constraint specifies that no more than N variables taken from the variable set Var are assigned values in value set Val. Assume n is the number of variables currently having values in Var. If $n > N$, we would prefer a smaller n. Take the constraint $\texttt{atmost}(1, \{x_1, x_2, x_3, x_4\}, \{3\})$ as an example. The assignment $(1, 3, 3, 2)$ for variable tuple (x_1, x_2, x_3, x_4) is better than $(3, 4, 3, 3)$ as we only need to change the value of one variable to get the constraint satisfied.

To utilize this information, we use the representation in figure 8 with the intermediate node a changed for storing the number of variables currently assigned values in Val. The initial penalty value for each violating tuple is given by the formula $N - a$ as follows:

a	$\delta'_{\texttt{atmost}}$
0	0
$\vdots$	$\vdots$
N	0
$N+1$	-1
$\vdots$	$\vdots$
$\lvert Var\rvert$	$N - \lvert Var\rvert$

where $|Var|$ is the cardinality of Var. This can be regarded as adding the set of redundant constraints $\{\texttt{atmost}(N+1, Var, Val), \texttt{atmost}(N+2, Var, Val), \ldots, \texttt{atmost}(|Var|-1, Var, Val)\}$. Here we set $contribute_{atmost}(i, (v_1, \ldots, v_n)) = (v_i \in Val)$, where $\in$ is a function returning 1 if value v_i is in Val and 0 otherwise.

We compare our implementations with CHIP on the car-sequencing problem which involves scheduling cars onto an assembly line so that different options can be installed on these cars satisfying various utilization constraints [4]. 50 problems [5] are tested, 10 for each utilization percentage in the range 60% to 80%. Using the method described in [4], CHIP can only manage to solve 6 out of

[5] We thank Andrew Davenport for supplying the car-sequencing benchmarks.

50 problems within one hour. We have also tested the performance of CHIP with the new method in [1]. There is not much improvement. For the two versions of E-GENET, the execution limit is set to 1000 repairs and the results are summerized in table 2. The modified E-GENET can terminate in less than 10 minutes for all 500 runs, requiring no more than 1800 repairs (except in one run). In general, there are increases of 15% to 20% in percentages of successful runs.

utiliza-	Original E-GENET		Modified E-GENET		penalty values
tion %	% succ. runs	median repair	% succ. runs	median repair	stored
60	74	223.5	100	282.5	29
65	80	223.5	99	262	20
70	81	241	100	280.5	30
75	84	339	97	331	74
80	53	576	73	537	187

Table 2. Results on car sequencing problems

5.3 Disjunctive Constraint

To handle a disjunctive constraint $C_1 \vee C_2 \vee \ldots \vee C_n$ in the modified E-GENET, we can use the representation in figure 10. For each constraint C_i, there is a corresponding intermediate node a_i which holds the initial penalty value for the associated tuple under the assignment scheme of C_i. Actually, a_i can be regarded as the degree of violation of C_i. Since we only need one of them satisfied, we add one more intermediate node b to store the maximum value among all a_i's. The assignment scheme of the disjunctive constraint would then be based on the value of b. Use the constraint $x = y + z \vee x = 2y - z$ as an example. The representation is given in figure 11. It is constructed with also techniques described in section 5.1.

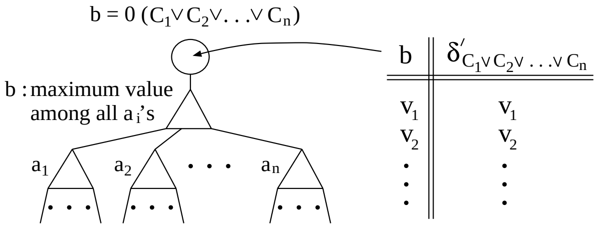


Fig. 10. The representation for a disjunctive constraint

The Hamiltonian path problem is used to test this handling method of disjunctive constraint. The problem can be summarized 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 formulate the problem as in [7]. In CHIP and the original E-GENET, one kind of redundant constraints is used to speed up the execution. We omit these constraints in the modified E-GENET intentionally to test the efficiency of our proposal.

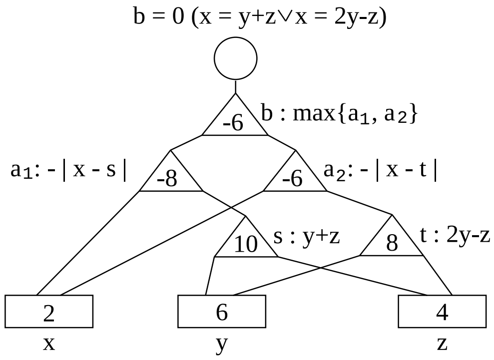


Fig. 11. The representation for the constraint $x = y + z \vee x = 2y - z$

The benchmarking results are listed in table 3. CHIP uses choices [10] to model disjunctive constraints. This approach induces a large number of backtracking and fails to solve a graph with 50 vertices in 10 hours. With more information provided by penalty values, the modified E-GENET is several times faster than the previous version.

graph node	CHIP CPU Time (sec)	Original E-GENET CPU Time (sec)	Modified E-GENET CPU Time (sec)	penalty values stored
30	581.640	54.000	6.550	215
40	4023.500	587.158	29.465	350
50	–	4897.092	783.790	2943

Table 3. Results on Hamiltonian path problems

5.4 Cumulative Constraint

The `cumulative` constraint [1] in CHIP is found to be useful as some real world problems like scheduling and placement problems can be stated more easily and directly. Thus, we implement it in modified E-GENET. The constraint has the form $\texttt{cumulative}([O_1, \ldots, O_m], [D_1, \ldots, D_m], [R_1, \ldots, R_m], L)$ and can be explained with a simple scheduling problem of m tasks and one kind of resources. We can treat $O_1, \ldots, O_m$ as starting times for the tasks. Each task i uses R_i amount of resources and lasts for D_i units of time. The constraint holds if at any time unit t, resources required are less than L, $\sum_{i|O_i \leq t \leq O_i+D_i-1} R_i \leq L$.

To handle the constraint, we can break it down into several inequality constraints and apply the techniques described in section 5.1. Assume $\alpha(V)$ be the minimum value in the domain of variable V and $\beta(V)$ the maximum. Let $t' = min\{\alpha(O_1), \ldots, \alpha(O_m)\}$ and $t'' = max\{\beta(O_1) + \beta(D_1), \ldots, \beta(O_m) + \beta(D_m)\}$. For each time unit t between t' and t'', we use an intermediate node to hold the resources required at that unit. In figure 12, the intermediate node a_1 is for t', a_2 for $t' + 1$, and so on. Then we need to solve n constraints of $a_i \leq L$, where $n = t'' - t' + 1$.

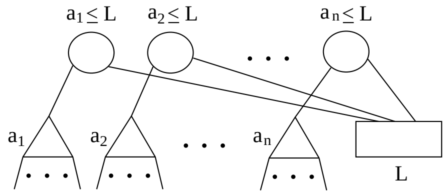


Fig. 12. The representation for the `cumulative` constraint

Simple scheduling problems of different number of tasks and resources utilization percentage (three problems for each setting) are used as testing examples and the results are shown in table 4. CHIP cannot solve two problems with 21 tasks within 1 hour. The performance of the original E-GENET is extremely poor as there is only one constraint in each problem. However, with the modifications, the new version of E-GENET can solve all problems efficiently.

task no.	utilization %	CHIP CPU Time (sec)			Modified E-GENET CPU Time (sec)			penalty values stored		
7	80	0.020	0.010	0.010	0.010	0.000	0.000	0	0	0
	85	0.025	0.010	0.010	0.010	0.040	0.000	0	0	0
	90	0.015	0.010	0.010	0.095	0.010	0.000	49	2	0
	95	0.015	0.010	0.010	0.050	0.010	0.010	37	3	5
14	80	0.205	0.025	0.015	0.090	0.095	0.020	3	2	1
	85	0.030	0.030	0.020	0.180	0.085	0.030	12	6	2
	90	4.630	0.020	0.070	0.410	0.375	0.030	38	43	2
	95	43.920	6.940	5.330	11.185	13.950	0.125	714	727	25
21	80	0.030	0.030	0.035	0.115	0.140	0.045	1	3	0
	85	0.055	0.045	0.030	0.435	0.265	0.080	13	4	2
	90	1269.560	0.125	0.030	2.420	0.915	0.135	57	28	9
	95	–	–	4.190	69.460	16.780	0.380	1501	407	23

Table 4. Results on simple scheduling problems

6 Related Work

Davenport *et al.* [2] proposed an extension to GENET for non-binary constraints, in which a hyper-edge is used to link up incompatible labels in an n-ary constraint. Our approach is different from theirs in three aspects:

1. In E-GENET, all constraints, binary or non-binary, have the same status and are represented homogeneously, while they are much smaller in size than their counterparts in the scheme of Davenport *et al*. For example, for the constraint $x+y=u+v$ where all variables have the domain $\{1,\ldots,100\}$, their scheme needs around 100^4 constraint nodes whereas only **3** nodes are needed in E-GENET and penalty values can be computed on demand.
2. The cost of network construction in E-GENET is low compared with that for the extension of Davenport *et al* as huge amount of work on building connections is removed. For the above example, 4×100^4 connections are required in their scheme but we can just use 6 in E-GENET.

3. E-GENET also provides much greater flexibility. New types of constraints and domain knowledge can be incorporated easily, not the case for their scheme, by using intermediate nodes and suitable assignment schemes of initial penalty values. As a result, better performance can be obtained.

7 Conclusion

In this paper, several modifications to E-GENET are proposed. The new version of E-GENET has three advantages over the original one. First, memory requirement is largely reduced. Second, there is a great increase in performance, especially on highly constrained problems. Third, some very complex constraints can now be handled. Both versions of E-GENET outperform CHIP on CSP's requiring much backtracking tree search. However, more experiments have to be done to compare these two approaches in different regions of the problem space.

Interesting future work includes applying the `cumulative` and the `diffn` constraint [1] of the modified E-GENET on real world problems and further enhancement of E-GENET to solve partial constraint satisfaction problems [6] and optimization problems.

References

1. N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Journal of Mathematical and Computer Modelling*, 17(7):57–73, 1994.
2. A. Davenport, E. Tsang, C. J. Wang, and K. Zhu. GENET: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. In *Proc. 12th National Conference on Artificial Intelligence*, 1994.
3. D. Diaz and P. Codognet. A minimal extension of the WAM for `clp(FD)`. In *Proc. 10th International Conference on Logic Programming*, pages 774–790, 1993.
4. M. Dincbas, H. Simonis, and P. Van Hentenryck. Solving car sequencing problem in contraint logic programming. In *Proc. European Conference on AI*, pages 290–295, 1988.
5. M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *Proc. International Conference on Fifth Generation Computer Systems*, pages 693–702, December 1988.
6. E. C. Freuder. Partial constraint satisfaction. In *Proc. 11th International Joint Conference on AI*, pages 278–283, 1989.
7. J.H.M. Lee, H.F. Leung, and H.W. Won. Extending GENET for non-binary CSP's. In *Proc. 7th International Conference on Tools with Artificial Intelligence*, pages 338–343, 1995.
8. 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.
9. The COSYTEC Team. *CHIP V4.1 User Manuals*, 1994.
10. P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. The MIT Press, 1989.
11. T. Warwick and E.P.K. Tsang. Tackling car sequencing problems using a generic genetic algorithm. *Evolutionary Computation*, 3(3):267–298, 1995. (to appear).