Improving Repair-based Constraint Satisfaction Methods by Value Propagation

Nobuhiro Yugami  Yuiko Ohta  Hirotaka Hara
FUJITSU LABORATORIES LTD.
1015, Kamikodanaka Nakahara-ku,
Kawasaki 211, Japan
yugami@flab.fujitsu.co.jp

Abstract
A constraint satisfaction problem (CSP) is a problem to find an assignment that satisfies given constraints. An interesting approach to CSP is a repair-based method that first generates an initial assignment, then repairs it by minimizing the number of conflicts. Min-conflicts hill climbing (MCHC) and GSAT are typical examples of this approach. A serious problem with this approach is that it is sometimes trapped by local minima. This makes it difficult to use repair-based methods for solving problems with many local minima.

We propose a new procedure, EFLOP, for escaping from local minima. EFLOP changes the values of mutually dependent variables by propagating changes through satisfied constraints. We can greatly improve the performance of repair-based methods by combining them with EFLOP.

We tested EFLOP with graph colorability problems, randomly generated binary CSPs and propositional satisfiability problems. EFLOP improved the performance of MCHC and GSAT for all experiments and was more efficient for large and difficult problems.

Introduction
A constraint satisfaction problem (CSP) is a problem to find an assignment that satisfies given constraints. Approaches to solving CSPs can be classified into constructive methods and repair-based methods [Minton et al. 92]. Constructive methods are based on a tree search and find a solution by incrementally extending a consistent partial assignment. Constraint directed search [Fox 87], arc and path consistency algorithms [Mohr & Henderson 86] and intelligent backtracking such as dependency directed backtracking [Doyle 79] fall into this group. Repair-based methods are based on a local search. They first generate an initial assignment with conflicts, then repair it by minimizing the number of conflicts. Min-conflicts hill climbing (MCHC) [Minton et al. 90] and GSAT [Selman, Levesque & Mitchell 92] are typical examples. For small- and medium-scale problems, constructive methods show good performance. However, repair-based methods are more practical for large-scale problems.

Repair-based methods use local search techniques such as hill climbing to minimize the number of conflicts. Local search techniques do not provide the capability to escape from local minima. It is, thus difficult to solve a CSP with many local minima by using a repair-based method. One solution is to use a more powerful local search technique such as simulated annealing (SA). Johnson et al. applied SA to graph colorability problems [Johnson et al. 91]. However, this has a major disadvantage that local search techniques capable of escaping from local minima such as SA, require a very long time for problem solving. Another solution is combining local search methods with a special procedure for escaping from local minima. [Selman & Kautz, 93] and [Morris, 93] proposed constraint weighting for escaping from local minima.

We propose a new procedure EFLOP (Escaping From Local Optima by Propagation) to resolve this problem. EFLOP changes the values of mutually dependent variables for escaping from local minima. Satisfied constraints are used for finding such a set of variables and their new values. Repair-based methods call EFLOP when they are trapped by local minima and restart search from an output assignment of EFLOP.
Repair-based methods and their limitations

Repair-based methods solve a minimization problem of the number of conflicts. If an original CSP is solvable, then the minimization problem has an optimum assignment with no conflict and the optimum assignment is a solution of the original CSP. Min-conflicts hill climbing (MCHC) [Minton et al. 90] solves this minimization problem by repairing an assignment with a following simple value selection heuristic.

**Min-conflicts heuristic**

Select a variable in conflict randomly. Assign it a value that minimizes the number of conflicts. Break ties randomly.

This heuristic guarantees that the number of conflicts of the new assignment is fewer than or equal to that of the old assignment because it selects the present value if all other values increase the number of conflicts. The number of conflicts, thus decreases monotonically.

MCHC could solve certain classes of CSPs, e.g., n-queens problems and graph colorability problems for dense graphs but could not solve other classes of CSPs such as graph colorability problems for sparse graphs [Minton et al. 92]. This is because MCHC has few ability to escape from local minima of the minimization problem. We explain this with an example of a graph colorability problem. Figure 1 shows a local minimum assignment for a 3-colorability problem used by Selman [Selman & Kautz 93] as an example that the basic GSAT could not escape from. Each node (variable) should be colored with one of the three colors but the color must be different from colors of neighboring nodes. In Figure 1, only one constraint, x=#y, is violated. Min-conflicts heuristic selects x (or y) and tries to assign a new value to x. However, x=green and x=blue cause two conflicts respectively and x=red, the present value, minimizes the number of conflicts. Min-conflicts heuristic, thus selects red for x and MCHC can not escape from this local minimum assignment.

**EFLOP procedure**

We first discuss the property of local minimum assignments of CSPs. The example in Figure 1 can be divided into two sub-problems where all constraints within each sub-problem are satisfied but a constraint between sub-problems is violated. In general, if an assignment is local minimum, a problem can be divided into consistent sub-problems where conflicts occur only between different sub-problems. In such a case, changing the value of the conflicting variable causes new conflicts in the sub-problem that the variable belongs to. This increases the total number of conflicts and makes it impossible to escape from local minima with hill climbing like methods such as MCHC and GSAT.

This property of local minima leads directly to the following procedure for escaping from them.

**Step 1:** Find a consistent sub-problem that involves a variable in conflict.

**Step 2:** Change the values of variables in the sub-problem so that the new values satisfy all constraints in the sub-problem.

However, this is not practical because Step 2 requires to solve the sub-problem and takes a long time if the sub-problem is not small. EFLOP avoids this difficulty by combining Step 1 and Step 2, extending the sub-problem incrementally by propagating the changes of values of variables through satisfied constraints.

Figure 2 shows the procedure of EFLOP. EFLOP first selects a variable in conflict randomly and changes the value of it. If this change causes a satisfied constraint to become unsatisfied, EFLOP tries to resolve it by changing a value of another variable in the constraint. If
procedure EFLOP
Input : a local minimum assignment;
Output: an assignment that is not local minimum;

begin
select a variable \( v \) in conflict randomly;
change \( v \)'s value randomly;
\( V := \{ v \} \);
while possible
select a constraint \( c \) that satisfies following conditions;
(c1) \( c \) is satisfied before EFLOP is called;
(c2) \( c \) is not satisfied now;
(c3) there is a variable \( v \) in \( c \) and its value \( a \) such that
\( (c3-1) \ v \notin V \);
\( (c3-2) v \)'s present value is consistent with old values of variables in \( V \);
\( (c3-3) v=a \) makes \( c \) satisfied;
\( (c3-4) v=a \) is consistent with new values of variables in \( V \);
change \( v \)'s value to \( a \);
add \( v \) to \( V \);
end_of_while;
end;

Figure 2: EFLOP procedure

this second change causes a new constraint violation, EFLOP tries to resolve it by changing the value of the third variable. This is continued until no new conflict can be resolved by changing values of variables that aren't changed yet.

If more than one variable-value pairs satisfy the condition (c3), EFLOP selects one of them with a following heuristic so as not to propagate value changes to too many variables.

**EFLOP heuristic**
Select a pair of a variable and its value that minimizes the number of constraints which are satisfied before EFLOP is called but will be violated after the change. Break ties randomly.

The sub-problem defined by the set of changed variables \( V \) and constraints between variables in \( V \), is consistent before EFLOP is called because of the condition (c3-2). The sub-problem is also consistent after EFLOP terminates because of the condition (c3-4).

We explain EFLOP procedure in detail using the example in Figure 1. EFLOP first selects a variable in conflict and its new value (new value must be different from the present value) randomly. Let \( x \) and \( green \) be selected. EFLOP changes the value of \( x \) to \( green \) and initializes the set of changed variables \( V = \{ x \} \). This new value, \( green \), violates two constraints, \( xfz \) and \( xfw \). Both of these constraints satisfy conditions (c1)-(c3), EFLOP thus selects one of them. Let \( xfz \) be selected. EFLOP tries to satisfy it by changing a value of a variable in it, i.e., \( x \) or \( z \) but EFLOP doesn't change \( x \)'s value because the condition (c3-1) requires that each variable is changed at most once. Two variable-value pairs, \( z=red \) and \( z=blue \), satisfy condition (c3). EFLOP selects \( z=blue \) because \( z=red \) causes one new conflict but \( z=blue \) causes no new conflict. EFLOP changes \( z \)'s value to \( blue \) and adds \( z \) to \( V \). Next, EFLOP tries to satisfy \( xfw \). Because of the same reason in \( xfz \), EFLOP changes \( w \)'s value to \( blue \). EFLOP terminates because there is no constraint that satisfies (c1)-(c3). EFLOP then assigns \( green \) to \( x \), \( blue \) to \( z \) and \( blue \) to \( w \), i.e., EffLOP changes the values of variables in sub-problem 1 (Figure 1) to other consistent values between them and the new assignment generated by EFLOP satisfies all constraints.

**Experiments**

In this section, we show the effect of EFLOP by using graph colorability problems, randomly generated binary CSPs and propositional satisfiability problems. EFLOP is not a method for solving CSP, but is a method for improving the performance of repair-based methods. We combined EFLOP with MCHC for binary CSPs and for colorability problems, and combined with GSAT for SAT. We compared their performance with and without EFLOP. MCHC (GSAT) with EFLOP calls EFLOP whenever it trapped by local minima and restarts from an output assignment of EFLOP. MCHC (GSAT) without EFLOP restarts from a randomly generated initial assignment when it is trapped by local minima. We used C language on a SPARCstation2 for all experiments.

**Graph Colorability Problems**
We generated 3-colorability problems of sparse graphs by the way in [Minton et al. 92]. We first divided N nodes
Table 1: Average numbers of hill climbing steps for graph 3-colorability problems

<table>
<thead>
<tr>
<th>nodes</th>
<th>edges</th>
<th>MCHC</th>
<th>MCHC+EFLOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>60</td>
<td>304</td>
<td>85</td>
</tr>
<tr>
<td>60</td>
<td>120</td>
<td>3,320</td>
<td>341</td>
</tr>
<tr>
<td>90</td>
<td>180</td>
<td>13,700</td>
<td>924</td>
</tr>
<tr>
<td>120</td>
<td>240</td>
<td>71,000</td>
<td>1,880</td>
</tr>
<tr>
<td>150</td>
<td>300</td>
<td>334,000</td>
<td>3,990</td>
</tr>
</tbody>
</table>

Table 2: 50% solvable strength and average number of hill climbing steps for binary CSPs

<table>
<thead>
<tr>
<th>variables</th>
<th>strength</th>
<th>MCHC</th>
<th>MCHC+EFLOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0.64</td>
<td>9,890</td>
<td>3,570</td>
</tr>
<tr>
<td>40</td>
<td>0.64</td>
<td>203,000</td>
<td>71,100</td>
</tr>
<tr>
<td>60</td>
<td>0.62</td>
<td>3,320,000</td>
<td>334,000</td>
</tr>
</tbody>
</table>

Figure 3: Probability of solvability of random CSPs with 20 variables, 10 values for each variable and 40 constraints

Figure 4: Average numbers of hill climbing steps for 20 variables CSPs

to 3 groups with N/3 nodes and randomly created edges between nodes in different groups. If the generated graph had unconnected components, we rejected it. This generation guarantees the solvability of generated problems.

We used the graphs with 2N edges because [Minton et al. 92] reported the poor performance of MCHC for such problems. Table 1 shows the average numbers of hill climbing steps of MCHC (average of 100 problems for each N) with and without EFLOP for 3-colorability problems with 30–150 nodes. EFLOP could improved MCHC drastically and its effect was larger for large problems than that of small problems.

**Binary CSPs**

We generated binary CSPs with 4 parameters, the number of variables, N, the domain size of each variable, D, the number of constraints, M and the strength of the constraint, S. The strength of the constraint is the ratio of the number of forbidden value pairs to the number of all value pairs of two variables in the constraint. This means that \((1-S)D^2\) value pairs satisfy the constraint and \(SD^2\) value pairs violate the constraint. When generating a problem, we first selected M variable pairs as constraints and for each constraint (variable pair), we selected \((1-S)D^2\) permitted value pairs randomly. We did all selections randomly, so a generated problem may not have a solution. This way of generating CSPs was based on [Freuder & Wallace 92].

We first tested EFLOP's effect on CSPs with 20 variables, 10 possible values for each variable and 40 constraints.
constraints (N=20, D=10, M=40). Figure 3 shows the probability of solvability, the ratio of solvable problems to generated problems and figure 4 shows the average numbers of hill climbing steps (average of 100 solvable problems for each strength). MCHC with EFLOP was faster than without at all strength and the difference was biggest at the "50% solvable" strength, the strength at which a half of randomly generated problems were solvable. This is because that when S was near this value, the ratio of the numbers of local minima to the number of solutions was large, so MCHC trapped by local minima with high probability.

Table 2 shows 50% solvable strengths and average numbers of hill climbing steps (average of 100 solvable problems) at the strengths of CSPs with 20, 40 and 60 variables. For each case, we set M=2N. It is interesting to note that the strength was almost constant. This suggests that the 50% solvable strength depends only on the constraints-to-variables ratio and domain size. The result was that EFLOP could improve the performance of MCHC for all cases and the improvement was greater for large problems. MCHC with EFLOP was about 3 times faster than MCHC without EFLOP for problems with 20 variables and the improvement was more than 10 times for problems with 60 variables.

Satisfiability Problems

A propositional satisfiability problem (SAT) is a problem to determine whether a given logical formula can be satisfied or not, and to find a truth assignment that satisfies the formula if it is satisfiable. The formula is given in conjunctive normal form and a clause is a constraint that at least one literal in the clause should be true. Selman et al. proposed an efficient repair-based method for SAT, GSAT [Selman, Levesque & Mitchell 92] and extended GSAT with clause weighting and random walk to overcome the inability to escape from local minima [Selman & Kautz 93]. We used basic GSAT and didn't use these extensions for our experiments because we wanted to know the effect of EFLOP alone.

We generated SAT with three parameters, the number of variables, N, and the number of clauses, M and the length of clause, K. For each clause, we first selected K variables randomly and negated each variable with probability 0.5. This generation was based on [Mitchell, Selman & Levesque 92].

We first tested EFLOP with 3-SAT (K=3) problems. Mitchell et al. [Mitchell, Selman & Levesque 92] reported the hardness of 3-SAT. Their conclusion was that 3-SAT was most difficult when the ratio of clauses to variables was 50% satisfiable ratio, the ratio at which a randomly generated problem was satisfiable with probability 0.5, and this ratio was 4.3 for 3-SAT. Table 3 shows the average number of flips (average of 100 satisfiable problems) for randomly generated 3-SAT with M=4.3N. GSAT with EFLOP was about twice as fast as GSAT without EFLOP. The difference between them didn't depend on the number of variables.

We also examined the dependency of the EFLOP's effect on clause length. Table 4 shows the 50% satisfiable clauses-to-variables ratio for 3-, 4- and 5-SAT and the average number of flips (average of 100 satisfiable problems) at the ratio. EFLOP was more effective for problems with longer clauses and GSAT with EFLOP was about 10 times faster than GSAT without EFLOP for 5-SAT.

<table>
<thead>
<tr>
<th>variables</th>
<th>clauses</th>
<th>GSAT</th>
<th>GSAT+EFLOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>215</td>
<td>1,390</td>
<td>562</td>
</tr>
<tr>
<td>100</td>
<td>430</td>
<td>11,300</td>
<td>5,760</td>
</tr>
<tr>
<td>150</td>
<td>637</td>
<td>87,800</td>
<td>35,600</td>
</tr>
</tbody>
</table>

Table 3: Average number of flips for 3-SAT

<table>
<thead>
<tr>
<th>clause length</th>
<th>clauses</th>
<th>GSAT</th>
<th>GSAT+EFLOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>215</td>
<td>1,390</td>
<td>562</td>
</tr>
<tr>
<td>4</td>
<td>495</td>
<td>7,950</td>
<td>2,230</td>
</tr>
<tr>
<td>5</td>
<td>1,075</td>
<td>68,000</td>
<td>4,760</td>
</tr>
</tbody>
</table>

Table 4: Average number of flips for 3-, 4- and 5-SAT with 50 variables

Conclusions

We have proposed a new procedure, EFLOP, for improving the performance of repair-based constraint satisfaction methods such as min-conflicts hill climbing (MCHC) and GSAT. Repair-based methods solve a CSP by minimizes the number of conflicts. The most serious problem of these methods is that they can't escape from...
local minima. EFLOP propagates changes of values through satisfied constraints and changes the values of mutually dependent variables at once. This enables to escape from local minima and thus improves the performance of repair-based methods. EFLOP is independent on problem domains and can be used with any repair-based method.

We tested EFLOP's effect by using graph colorability problems, randomly generated binary CSPs and satisfiability problems. EFLOP improved the performance of MCHC and GSAT, and the improvement was greater for larger and more difficult problems.

An interesting question is how effective EFLOP is for structured problems such as planning problems. In structured problems, some variables strongly depend on each other and a value change for one variable causes many conflicts between such variables. In a planning problem, a selection of an operator is strongly dependent on previous operators and affects selections for following operators through its preconditions and post conditions. This property causes many local minima and makes it very difficult to solve structured problems by repair-based methods [Kautz & Mitchell 92]. EFLOP changes values of such mutually dependent variables at once, and the ability to do this will improve the performance of repair-based methods for structured problems.

References