Retroactive Ordering for Dynamic Backtracking

Roie Zivan, Uri Shapen, Moshe Zazone and Amnon Meisels,
{zivanr,moshezaz,shapenko,am}@cs.bgu.ac.il

Department of Computer Science,
Ben-Gurion University of the Negev,
Beer-Sheva, 84-105, Israel

Abstract. Dynamic Backtracking (DBT) is a well known algorithm for solving Constraint Satisfaction Problems. In DBT, variables are allowed to keep their assignment during backjump, if they are compatible with the set of eliminating explanations. A previous study has shown that when DBT is combined with variable ordering heuristics it performs poorly compared to standard Conflict-directed Backjumping (CBJ) [1]. The special feature of DBT, keeping valid elimination explanations during backtracking, can be used for generating a new class of ordering heuristics. In the proposed algorithm, the order of already assigned variables can be changed. Consequently, the new class of algorithms is termed *Retroactive DBT*.
In the proposed algorithm, the newly assigned variable can be moved to a position in front of assigned variables with larger domains and as a result prune the search space more effectively. The experimental results presented in this paper show an advantage of the new class of heuristics and algorithms over standard DBT and over CBJ. All algorithms tested were combined with forward-checking and used a *Min-Domain* heuristic.

1 Introduction

Conflict directed Backjumping (CBJ) is a technique which is known to improve the search of Constraint Satisfaction Problems (CSPs) by a large factor [4, 7]. Its efficiency increases when it is combined with forward checking [8]. The advantage of CBJ over standard backtracking algorithms lies in the use of conflict sets in order to prune unsolvable sub search spaces [8]. The down side of CBJ is that when such a backtrack (back-jump) is performed, assignments of variables which were assigned later than the culprit assignment are discarded.

Dynamic Backtracking (DBT) [5] improves on standard CBJ by preserving assignments of non conflicting variables during back-jumps. In the original form of DBT, the culprit variable which replaces its assignment is moved to be the last among the assigned variables. In other words, the new assignment of the culprit variable must be consistent with all former assignments. Although DBT saves unnecessary assignment attempts and therefore was proposed as an improvement to CBJ, a later study by Baker [1] has revealed a major drawback of DBT. According to Baker, when no specific ordering heuristic is used, DBT performs better than CBJ. However, when ordering heuristics which are known to improve the run-time of CSP search algorithms by a large factor are used [6, 2, 3], DBT is slower than CBJ. This phenomenon is easy to explain. Whenever the algorithm performs a back-jump it actually takes a variable which was placed according to the heuristic in a high position and moves it to a lower position. Thus, while in CBJ, the variables are ordered according to the specific heuristic, in DBT the order of variables becomes dependent on the algorithm's behavior [1].

In order to leave the assignments of non conflicting variables without a change on backjumps, DBT maintains a system of eliminating explanations ($Nogoods$) [5]. As a result, the DBT algorithm maintains dynamic domains for all variables and can potentially benefit from the *Min-Domain* (fail first) heuristic.

The present paper investigates a number of improvements to DBT that use radical versions of the *Min-Domain* heuristic. First, the algorithm avoids moving the culprit variable to the lowest position in the partial assignment. This alone can be enough to eliminate the phenomenon reported by Baker [1].

Second, the assigned variables which were originally ordered in a lower position than the culprit variable can be reordered according to their current domain size.

Third, a $retroactive$ ordering heuristic in which assigned variables are reordered is proposed. A *retroactive* heuristic allows an assigned variable to be moved upwards beyond assigned variables as far as the heuristic is justified.

If for example the variables are ordered according to the *Min-Domain* heuristic, the potential of each currently assigned variable to have a small domain is fully utilized. We note that although variables are chosen according to a *Min-Domain* heuristic, a newly assigned variable can have a smaller current domain than previously assigned variables. This can happen because of two reasons. First, as a result of *forward-checking* which might cause values from the current variables' domain to be eliminated due to conflicts with unassigned variables. Second, as a result of multiple backtracks to the same variable which eliminate at least one value each time. Therefore, the exploitation of the heuristic properties can be done, not only by choosing the next variable to be assigned, but by placing it in its *right* place among the assigned variables after it is assigned successfully.

The combination of the three ideas above was found to be successful in the empirical study presented in the present paper.

2 Retroactive Dynamic Backtracking

We assume in our presentation that the reader is familiar with both DBT following [1] and CBJ [8].

The first step in enhancing the desired heuristic (*Min-Domain* in our case) for DBT is to avoid the moving forward variables that the algorithm backtracks to (i.e. culprit variables). One way to do this is to try to replace the assignment of the culprit variable and *leave the variable in the same position.*

The second step is to reorder the assigned variables that have a lower order than the culprit assignment which was replaced. This step takes into consideration the possibility that the replaced assignment of a variable that lies higher in the order has the potential to change the size of the current domains of the already assigned variables that are ordered after it. The simplest way to perform this step is to reassign these variables and order them using the desired heuristic.

The third step derives from the observation that in many cases the size of the current domain of a newly assigned variable is smaller than the current domains of variables which were assigned before it.

Allowing a reordering of assigned variables enables the use of heuristic information which was not available while the previous assignments have been performed. This takes ordering heuristics to a new level and generates a radical new approach. Variables can be moved up in the order, in front of assigned variables of the partial solution. As long as the new assignment is placed *after* the most recent assignment which is in conflict with one of the variable's values, the size of the domain of the assigned variable is not changed.

```
Retroactive FC_DBT
1.  var_list ← variables;
2.  assigned_list ← φ;
3.  pos ← 1;
4.  while (pos < N)
5.     next_var ← select_next_var(var_list);
6.     var_list.remove(next_var);
7.     assign(next_var);
8.  report solution;

procedure assign(var)
10. for each (value ∈ var.current_domain)
11.    var.assignment ← value;
12.    consistent ← true;
13.    forall (i ∈ var_list)
          and while consistent
14.       consistent ← check_forward(var, i);
15.    if not (consistent)
16.       nogood ← resolve_nogoods(i);
17.       store(var, nogood);
18.       undo_reductions(var, pos);
19.    else
20.       nogood ← resolve_nogoods(pos);
21.       lastVar ← nogood.RHS_variable;
21.       newPos ← select_new_pos(var, lastVar);
22.       assigned_list.insert(var, newPos);
23.       forall (var_1 ∈ assigned_list) and
             (pos_var_1 > newPos)
24.          check_forward(var, var_1);
25.          update_nogoods(var, var_1);
26.       forall (var_2 ∈ var_list)
27.          update_nogoods(var, var_2);
28.       pos ← pos+1;
29.       return;
30. var.assignment ← Nil;
31. backtrack(var);

procedure backtrack(var)
32. nogood ← resolve_nogoods(var);
33. if (nogood = φ)
34.    report no solution;
35.    stop;
36. culprit ← nogood.RHS_variable;
37. store(culprit, nogood);
38. culprit.assignment ← Nil;
39. undo_reductions(culprit, pos_culprit);
40. forall (var_1 ∈ assigned_list) and
          (pos_var_1 > newPos)
41.    undo_reductions(var_1, pos_var_1);
42.    var_1.assignment ← Nil;
43.    var_list.insert(var_1);
44.    assigned_list.remove(var_1);
45. pos ← pos_culprit;

procedure update_nogoods(var_1, var_2)
46. for each (val ∈
          {var_2.domain - var_2.current_domain})
47.    if not (check(var_2, val, var_1.assignment))
48.       nogood ← remove_eliminating_nogood
             (var_1, val);
49.       if not (∃var_3 ∈ nogood and
             pos_var_3 < pos_var_1)
50.          nogood ← ⟨var_1.assignment →
             var_2 ≠ val⟩;
51.       store(var_2, nogood);
```

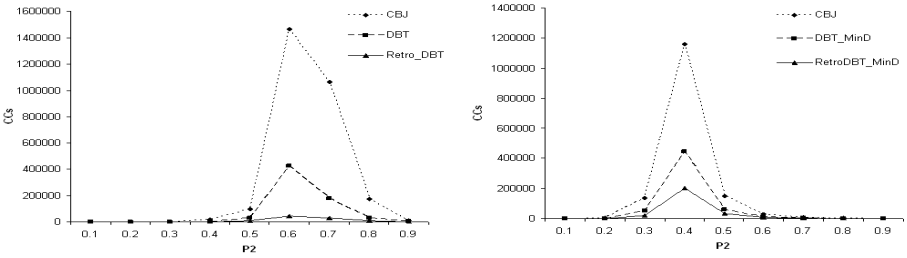
Fig. 1. The Retroactive FC_DBT algorithm

In the best ordering heuristic proposed by the present paper, the new position of the assigned variable in the order of the $partial\_solution$ is dependent on the size of its current domain. The heuristic checks all assignments from the last up to the first assignment which is included in the union of the newly assigned variable's eliminating Nogoods. The new assignment will be placed right after the first assigned variable with a smaller current domain.

Figures 1 presents the code of *Retroactive Forward Checking Dynamic Backtracking* ($Retro\_FC\_DBT$). For lack of space we leave out the detailed description of the algorithm and its correctness proof.

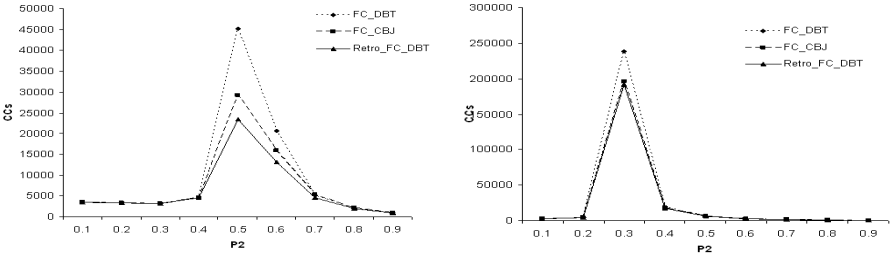
3 Experimental Evaluation

The common approach in evaluating the performance of *CSP* algorithms is to measure time in logical steps to eliminate implementation and technical parameters from affecting the results. The number of constraints checks serves as the measure in our experiments [9, 7].



(a) (b)

Fig. 2. CCs performed by *DBT*, *CBJ* and *Retroactive DBT* (a) $p_1 = 0.3$, (b) $p_1 = 0.7$.



(a) (b)

Fig. 3. CCs performed by *FC_DBT*, *FC_CBJ* and *FC_Retroactive DBT* (a) $p_1 = 0.3$, (b) $p_1 = 0.7$.

Experiments were conducted on random *CSPs* of n variables, k values in each domain, a constraints density of p_1 and tightness p_2 (which are commonly used in experimental evaluations of CSP algorithms [10]). Two sets of experiments were performed. In the first set the *CSPs* included 15 variables ($n = 15$) and in the second set the *CSPs* included 20 variables ($n = 20$). In all of our experiments the number of values for each variable was 10 ($k = 10$). Two values of constraints density were used, $p_1 = 0.3$ and $p_1 = 0.7$. The tightness value p_2, was varied between 0.1 and 0.9, in order to cover all ranges of problem difficulty. For each of the pairs of fixed density and tightness (p_1, p_2), 50 different random problems were solved by each algorithm and the results presented are an average of these 50 runs.

Three algorithms were compared, Conflict Based Backjumping (CBJ), Dynamic Backtracking (DBT) and Retroactive Dynamic Backtracking (*Retro* DBT). In all of our experiments all the algorithms use a *Min-Domain* heuristic for choosing the next variable to be assigned. In the first set of experiments, the three algorithms were implemented without forward-checking.

Figure 2 (a) presents the number of constraints checks performed by the three algorithms on low density CSPs ($p_1 = 0.3$). The CBJ algorithm does not benefit from the heuristic when it is not combined with forward-checking. The advantage of both versions of DBT over CBJ is therefore large. *Retroactive DBT* improves on standard *DBT* by a large factor as well. Figure 2 (b) present the results for high density CSPs ($p_1 = 0.7$). Although the results are similar, the differences between the algorithms are smaller for the case of higher density CSPs..

In our second set of experiments, each algorithm was combined with *Forward-Checking* [8]. This improvement enabled testing the algorithms on larger CSPs with 20 variables

Figure 3 (a) presents the number of constraints checks performed by each of the algorithms. It is very clear that the combination of *CBJ* with forward-checking im-

proves the algorithm and makes it compatible with the others. This is easy to explain since the pruned domains as a result of forward-checking enable an effective use of the Min-Domain heuristic. Both *FC_CBJ* and *Retroactive FC_DBT* outperform *FC_DBT*. *Retroactive FC_DBT* performs better than *FC_CBJ*. Figures 3 (b) presents similar results for higher density CSPs. As before, the differences between the algorithms are smaller when solving CSPs with higher densities.

4 Discussion

Variable ordering heuristics such as *Min-Domain* are known to improve the performance of CSP algorithms [6, 2, 3]. This improvement results from a reduction in the search space explored by the algorithm. Previous studies have shown that DBT does not preserve the properties of variable ordering heuristics and as a result performs poorly compared to CBJ [1]. The *Retroactive* DBT algorithm, presented in this paper, combines the advantages of both previous algorithms by preventing the placing of variables in a position which does not support the heuristic and allowing the reordering (or reassigning) of assigned variables with lower priority than the culprit assignment after a backtrack operation.

We have used the mechanism of *Dynamic Backtracking* which by maintaining eliminating $Nogoods$, allows variables with higher priority to be reassigned while lower priority variables keep their assignment. These dynamically maintained domains enable to take the *Min-Domain* heuristic to a new level. Standard backtracking algorithms use ordering heuristics only to decide on which variable is to be assigned next. *Retroactive* DBT enables the use of heuristics which reorder assigned variables. Since the sizes of the current domains of variables are dynamic during search, the flexibility of the heuristics which are possible in *Retroactive* DBT enables a dynamic enforcement of the *Min-Domain* property over assigned and unassigned variables.

The ordering of assigned variables requires some overhead in computation when the algorithm maintains consistency by using *Forward Checking*. This overhead was found by the experiments presented in this paper to be worth the effort since the overall computation effort is reduced.

References

[1] Andrew B. Baker. The hazards of fancy backtracking. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI '94), Volume 1*, pages 288–293, Seattle, WA, USA, July 31 - August 4 1994. AAAI Press.

[2] C. Bessiere and J.C. Regin. Using bidirectionality to speed up arc-consistency processing. *Constraint Processing (LNCS 923)*, pages 157–169, 1995.

[3] R. Dechter and D. Frost. Backjump-based backtracking for constraint satisfaction problems. *Artificial Intelligence*, 136:2:147–188, April 2002.

[4] Rina Dechter. *Constraint Processing*. Morgan Kaufman, 2003.

[5] M. L. Ginsberg. Dynamic backtracking. *J. of Artificial Intelligence Research*, 1:25–46, 1993.

[6] R. M. Haralick and G. L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[7] G. Kondrak and P. van Beek. A theoretical evaluation of selected backtracking algorithms. *Artificial Intelligence*, 21:365–387, 1997.

[8] P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9:268–299, 1993.

[9] P. Prosser. An empirical study of phase transitions in binary constraint satisfaction problems. *Artificial Intelligence*, 81:81–109, 1996.

[10] B. M. Smith. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81:155 – 181, 1996.