

Forward Checking and Intelligent Backtracking*

D.A. Wolfram
University of Cambridge
Computer Laboratory
New Museums Site
Pembroke Street
Cambridge CB2 3QG
England.

Keywords: Forward checking; intelligent backtracking; search rearrangement; computational complexity.

1 Forward Checking

Forward checking (FC) [1] is similar to backtracking except that after a choice is made, the sets containing possible future choices are pushed on a stack and *filtered* [5] by deleting inconsistent choices from them. If ever a selected set is empty, the stack is popped to find a previously selected set which is not empty. FC has principally been used to solve finite combinatorial logic program formulations [2]. search problems [4], and their It can also be used to execute logic programs, instead of backtracking [8]. Candidate input clauses for atoms in a goal can be checked for consistency in a similar way to future choices in finite search problems. The refinement of *search rearrangement* (SR) [6] can be incorporated. This selects a set with the fewest consistent choices, rather than following a fixed ordering.

Even though forward checking with search rearrangement (FC+SR) can be faster than backtracking for some problems [1], it can make unnecessary consistency tests, and do unnecessary stacking. We shall consider two optimisations of FC to reduce these inefficiencies: *partial filtering*, and *partial stacking*. We also give a group of problems for which FC has exponential space and time overheads compared with backtracking.

The following distinct representatives problem will be used as a running example: given sets $S_1 = \{2, 8\}$, $S_2 = \{3, 5\}$, $S_3 = \{4, 5\}$, $S_4 = \{1, 3\}$, $S_5 = \{6, 7\}$, and $S_6 = \{1, 3\}$, find all sets with six elements of distinct $l_i \in S_i$, where $1 \leq i \leq 6$.

1.1 Partial Filtering

Partial filtering can reduce unnecessary consistency tests that FC+SR does to select the next set.

*In: *Information Processing Letters* 32:85-87, 1989

From the example, if 2 has been chosen from S_1 and 3 has been chosen from S_2 then 1 is the only consistent choice from S_4 . The consistency of every choice in the sets S_3 to S_6 is tested. However after filtering S_4 , it is unnecessary to find more than one consistent choice from each of the remaining sets. This is because SR will select S_4 or a set with fewer consistent choices than those in S_4 . In the example, finding at most one consistent choice from each of S_5 and S_6 reduces the number of consistency tests by two, compared with the unoptimised version, where there are four consistency tests.

In general, *partial filtering* finds at most n consistent choices from each unfiltered set, where n is the current minimum of the sizes of the filtered sets. Its initial value is the size of the smallest unfiltered set. A filtered set of minimum size among the filtered sets contains no inconsistent choices.

An implementation of FC with partial filtering for the eight queens problem required 11139 consistency tests compared to 18114 for FC+SR for the same problem.

1.2 Partial Stacking

It is unnecessary to push unselected sets on the stack when the currently selected set contains no more choices. When the next selected set also contains no more choices, the stack must be popped at least twice to find a previously selected set which is not empty. This first occurs in the example problem when 2 has been chosen from S_1 , 3 has been chosen from S_2 , and 1 has been chosen from S_4 . There are no more choices from S_4 , so it is unnecessary to push the sets $\{4, 5\}$, $\{6, 7\}$ and $\{1, 3\}$.

FC without this optimisation wastes time and space because it does unnecessary copying of unselected sets when the current set contains no more choices. Partial stacking is similar to the optimisation of tail recursion. SR and partial filtering can both be used with this modification.

1.3 Exponential Overheads

For some problems, FC can incur exponential space and time overheads compared with ordinary backtracking.

For example, to solve the distinct representatives problem $S_1 = \{1, 2\}$, $S_2 = \{3, 4\}$, $S_3 = \{5, 6\}$, sixteen choices must be copied. The sets $\{3, 4\}$ and $\{5, 6\}$ are pushed on the stack for each choice from S_1 , and the set $\{5, 6\}$ is pushed on the stack four times, once for each consistent combination of choices from S_1 and S_2 . In general, if the problem to be solved consists of n sets each of which contains two choices, and there are 2^n solutions, then $2^{n+2} - 4n - 4$ choices must be copied. This is found by solving the linear recurrence $f(n) = 2f(n-1) + 4n - 4$ where $f(0) = 0$. The term $4n - 4$ arises because after each choice from the first selected set, $n - 1$ sets must be pushed on the stack and each of these sets has size 2. The solution to the recurrence can be verified by induction. Even if partial filtering and partial stacking are used, such overheads can arise.

2 Intelligent Backtracking

The previous set is always the backtrack point in ordinary backtracking, but choosing an earlier set can prevent the repetition of predictable failures. There have been several approaches to finding early backtrack points for sequential or parallel implementations of logic programming. Compiled backtrack methods generally are modifications of implementation or process models, which aim to be faster than backtracking, while not pruning solutions [3].

Some methods of intelligent backtracking which are based on finding maximal unifiable or minimal nonunifiable subsets with respect to a large defined class of unifiability procedures are NP-complete, NP-hard, or intractable and can be much slower in general than backtracking [7].

In this section, we consider the effects of combining FC+SR with intelligent backtracking. Intelligent backtracking here refers to modifications of backtracking which find early backtrack points without pruning solutions. Methods for logic programming are special cases.

The first failure for the distinct representatives problem using backtracking occurs for the branch of the search tree:

$$root \rightarrow (S_1, 2) \rightarrow (S_2, 3) \rightarrow (S_3, 4) \rightarrow (S_4, 1) \rightarrow (S_5, 6)$$

because there is no consistent choice from $S_6 = \{1, 3\}$.

An intelligent backtrack algorithm should choose S_4 rather than S_5 as the backtrack point. If the current set contains no consistent choices, let us define the *backtrack point* found by intelligent backtracking as the previous set which is most recent and whose choice produces an inconsistency with a choice from the current set. In general, we have the following result.

Lemma 1 *If the current set contains no consistent choices, then the backtrack point found by intelligent backtracking does not lose solutions.*

Proof. Changing choices in more recent sets cannot produce further solutions. There could be an untried choice at the backtrack point which leads to a solution. Thus, only if backtracking returns to an earlier backtrack point than this most recent previous set, could solutions be lost. \square

Remark. This set is the earliest backtrack point when there are untried choices in it.

It might be expected that FC+SR could produce solutions faster if it were combined with intelligent backtracking. However, when the current set contains no consistent choices, such a use of intelligent backtracking is redundant and inefficient.

The first failure for the distinct representatives problem using FC+SR occurs for the branch of the search tree:

$$root \rightarrow (S_1, 2) \rightarrow (S_2, 3) \rightarrow (S_4, 1)$$

because there is no consistent choice from $S_6 = \{1, 3\}$. The backtrack point is S_4 . This is the same backtrack point that intelligent backtracking would find.

Lemma 2 *If in solving a problem with FC+SR the current set contains no consistent choices, the previous set is the same backtrack point that intelligent backtracking would find.*

Proof. Suppose, on the contrary, that no choice in the previous set is inconsistent with any choice in the current set. Then all choices in the current set are inconsistent with choices made before the choice in the previous set.

This is a contradiction because SR would ensure that the current set would have been selected before the previous set because it would be smaller. By definition, the previous set is the same backtrack point that intelligent backtracking would find. \square

3 Conclusion

Even though our methods of partial checking and partial stacking can reduce the inefficiencies of unnecessary consistency tests and unnecessary stacking, FC is shown to have exponential space and time overheads for some problems.

There is no general time advantage for combining intelligent backtracking, with FC+SR; we have shown that the same backtrack point can be found more quickly without intelligent backtracking when the current set contains no consistent choices.

FC could be faster than intelligent backtracking for some problems, however every method of intelligent backtracking with tractable space and time overheads can avoid possible exponential overheads of FC and its refinements, and also find early backtrack points without losing solutions.

4 Acknowledgements

I thank Bill Clocksin and Larry Paulson for comments on this paper.

References

- [1] R.M. Haralick and G.L. Elliott, Increasing Tree Search Efficiency for Constraint Satisfaction Problems, *Artificial Intelligence* **14** (1980) 263–313.
- [2] P. Van Hentenryck and M. Dincbas, Forward Checking in Logic Programming, *Proceedings of the Fourth International Conference on Logic Programming*, Melbourne, 1987, Volume 1, MIT, 1987, 229–256.
- [3] Y.-J. Lin and V. Kumar, An Execution Model for Exploiting AND-Parallelism in Logic Programs, *New Generation Computing* **5** (1988) 393–425.
- [4] J.J. McGregor, Relational Consistency Algorithms and their Application in Finding Subgraph and Graph Isomorphisms, *Information Science* **19** (1979) 229–250.

- [5] A.K. Mackworth, Consistency in Networks of Relations, *Artificial Intelligence* **8** (1977) 99–118.
- [6] P.W. Purdom, Jr., Search Rearrangement Backtracking and Polynomial Average Time, *Artificial Intelligence* **21** (1983) 117–133.
- [7] D.A. Wolfram, Intractable Unifiability Problems and Backtracking, *Journal of Automated Reasoning* **5** (1989) 37–47.
- [8] D.A. Wolfram, *Adaptive Backtracking*, M.Sc. Thesis, Department of Computer Science, University of Melbourne, 1985.