Limited Assignments: A New Cutoff Strategy for Incomplete Depth-First Search

Roman Barták
Charles University, Faculty of Mathematics and Physics
Malostranské náměstí 2/25, Prague
Czech Republic
phone: +420 221 914 242
roman.bartak@mff.cuni.cz

Hana Rudová
Masaryk University, Faculty of Informatics
Botanická 68a, Brno
Czech Republic
phone: +420 549 496 345
hanka@fi.muni.cz

ABSTRACT
In this paper, we propose an extension of three incomplete depth-first search techniques, namely depth-bounded backtrack search, credit search, and iterative broadening, towards producing incomplete solutions. We also propose a new cutoff strategy for incomplete depth-first search motivated by a human style of problem solving. This technique, called limited assignment number (LAN) search, is based on limiting the number of attempts tried to assign a value to the variable. A linear worst-case time complexity of LAN Search leads to promising stable time behavior in all accomplished experiments. The techniques are studied in the context of constraint satisfaction problems.

Categories and Subject Descriptors
I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search – *backtracking, graph and tree search strategies.*

General Terms
Algorithms. Performance. Experimentation.

Keywords
Search. Constraint satisfaction.

1. INTRODUCTION
Search techniques are a popular problem solving approach in Artificial Intelligence and in particular in its sub-area called Constraint Satisfaction [5]. The goal of constraint satisfaction is finding an assignment of values to variables satisfying constraints between the variables. Sometimes, it is hard to find such a complete assignment (a Constraint Satisfaction Problem is NP-hard in general) or it is even impossible (the problem is called over-constrained). To handle both hard-to-solve and over-constrained problems, we proposed to consider partial assignments as an approximation of the solution in [2]. The idea was to instantiate the maximal number of variables at a limited time, for example to allocate the maximal number of lectures to available rooms [11]. Naturally, at least the constraints between the instantiated variables must be satisfied. We look for a maximal consistent assignment with the help of incomplete search techniques. They spread the search effort over the whole search space by skipping not promising parts of the search tree. Skipping is realized via a cutoff limit that stops complete exploration of some sub-space and forces the search algorithm to move to a different part. The variables that remain unassigned after the algorithm finishes can be seen as a hard part of the problem. Thus during the next run of the incomplete search algorithm (called restart), the algorithm can focus on assigning these hard variables primarily.

In this paper, we will focus on general cutoff techniques for incomplete depth-first search algorithms applied to constraint satisfaction problems. In particular, we will discuss limited depth [7], credit [3], and breadth [6] and we will add a new technique of a limited number of assignments per variable. This new technique, called LAN Search, follows up the human style of problem solving – we are trying to assign a value to some variable and if we do not succeed soon, we leave this variable unassigned and focus the attention to another variable. The hope behind this technique is making the runtime more stable in comparison to existing cutoff techniques while still obtaining a good solution (a quality is measured by the number of instantiated variables). In this paper, we will focus on the cutoff techniques only and the restart strategies will be a part of our follow-up research. We also do not cover the discrepancy-based incomplete search techniques like Limited Discrepancy Search [8], which depend on specific high-quality heuristics that are not always available.

To summarize, the main contribution of this paper is threefold. Firstly, an extension of three incomplete depth-first search algorithms towards producing incomplete assignments is proposed and all these algorithms are presented in the same uniform fashion. Secondly, a new cutoff strategy is introduced with the hope to improve the runtime stability of incomplete depth-first search. Thirdly, all the studied search techniques are compared on several benchmark domains, which, as far as we know, is the first such comparison in the literature.

SAC'05, March 13-17, 2005, Santa Fe, New Mexico, USA.

2. PRELIMINARIES

A *constraint satisfaction problem (CSP)* is a triple (V,D,C) where V is a finite set of variables, D is a set of possible values for variables (domain), and C is a finite set of constraints restricting the values of variables [5]. *Assignment* σ for a CSP (V,D,C) is a set of pairs v_i/d_i such that $v_i \in V$, $d_i \in D$ and each v_i appears at most once in σ. We call the assignment σ *complete* if each $v_i \in V$ appears exactly once in σ, otherwise the assignment is *incomplete*. We call the assignment τ an *extension* of the assignment σ if $\sigma \subset \tau$. A *solution* to a CSP is a complete assignment of the variables that satisfies all the constraints.

Depth-first search techniques typically extend a partial consistent assignment towards a complete consistent assignment, where by *consistent assignment* we understand the assignment satisfying at least the constraints over the instantiated variables (the variables from the assignment). There exist stronger consistency techniques that can check validity of constraints in advance by propagating information about the current assignment towards the non-instantiated variables. In particular, the values incompatible with the current partial assignment are removed from the domains (*domain filtering*), because these values cannot participate in any assignment extending the current assignment and satisfying all the constraints. If any domain becomes empty then the partial assignment is inconsistent; otherwise the partial assignment is *locally consistent*. Note, that in addition to a consistency check the domain filtering also prunes the search space to be explored when extending the partial assignment. On the other hand, local consistency of the assignment σ usually does not guarantee existence of a solution extending σ (there exist global consistency techniques but their complexity is often too large). In this paper, we focus on the depth-first search techniques and we expect that they are accompanied by a local consistency technique, typically generalized arc consistency [5].

For many constraint satisfaction problems, it is hard or even impossible to find a solution in the above sense that is to find a complete consistent assignment. For example, there is no complete assignment satisfying all the constraints in over-constrained problems. Therefore a generalized view of the solution based on the notion of a *maximal consistent assignment* was proposed in [2]. The basic idea is to assign as many variables as possible without getting (local) inconsistency. So, the maximal consistent assignment is defined as a consistent assignment of the largest cardinality. We can also define a weaker notion of *locally maximal consistent assignment* which is a consistent assignment that cannot be extended to a non-instantiated variable.

Note that if there is a solution to a CSP then it is a maximal consistent assignment and, vice versa, if the cardinality of a maximal consistent assignment equals to the number of variables in the problem then this assignment is a solution. The maximal consistent assignment can also be seen as a solution to over-constrained problems where no complete consistent assignment exists. Moreover looking for a maximal consistent assignment has the advantage that it is not necessary to know in advance whether the problem is over-constrained or not.

The search technique that we propose explores the locally maximal consistent assignments and remembers the assignment with the largest number of assigned variables. If the search space is explored completely then the maximal consistent assignment is obtained. For incomplete search techniques, we will get an *approximation* of the solution. The larger assignments we get the better is the approximation. Thus, we can now compare incomplete search techniques by the number of instantiated variables in the best locally maximal consistent assignment found. Note also that our approach is applicable to hard-to-solve problems where approximate solutions can be obtained with limited resources.

3. INCOMPLETE DEPTH-FIRST SEARCH

In this chapter, we will survey non-discrepancy based incomplete depth-first search algorithms, namely depth-bounded backtrack search [7], credit search [3], and iterative broadening [6]. Actually, the only missing algorithm from this category that we are aware of is bounded backtrack search [7] that limits the number of backtracks. We decided to omit this algorithm because it can be seen as an instance of the above algorithms by including a time limit (that is actually used in our experiments). Moreover bounded backtrack search does not follow the idea of spreading the search effort over the search space because all the explored branches are cumulated in the same area.

We will present the algorithms in a new uniform recursive code sharing the same structure and common procedures. The reason for this uniformity is an attempt to abstract from the particular implementation which helps us to compare better the core features of the algorithms. In particular, we will focus on the comparison of the cutoff schemes rather than on the restart strategies that will be studied next.

Another difference from the traditional formulation of these algorithms is using a specific, possibly non-binary, branching scheme called *enumeration* that selects a value for the variable in each step. Thus all the algorithms share the procedures for variable selection (select_free_variable/1) that determine the shape of the search tree and value selection (select_first_value/1, select_next_value/2) that determine the order in which the branches are explored. All the presented algorithms can use also the *step* branching scheme ($X=value \vee X \neq value$) but it leads to a complete depth-first search for iterative broadening (with the limit two or more). Finally, LAN search is currently designed for assignment-based search only (for example the *bisection* branching scheme with $X=<value \vee X>value$ is not studied). Thus, the enumeration branching scheme seems the best for the comparison of all the algorithms.

Note also, that constraint propagation is integrated into the search algorithms in a MAC-like scheme [5]. It means that each time the algorithm attempts to assign a value to the variable (assign/2), constraint propagation is evoked and domains of non-instantiated variables are filtered. If any domain becomes empty then the assignment fails and a next value is tried. This so called shallow backtracking is not counted as a valid assignment in our algorithms. Otherwise search proceeds to the next variable. Upon backtracking, the current variable assignment must be revoked together with the changes in the variables' domains that have been done during constraint propagation (unassign/1).

Finally, we extended the algorithms to memorize the largest assignment found during search. For simplicity reasons, we evaluate the assignments before any backtrack (update_best/1). Recall that constraint propagation is integrated into the search

procedure so the actual number of instantiated variables can be higher than the actual depth where the partial assignment is being saved. If the algorithm succeeds (the value true is returned) then a complete consistent assignment has been found and all the variables have their own values. Otherwise (the value fail is returned) the largest assignment can be recovered from the saved assignment. This technique is going in the direction towards the locally maximal consistent assignment but it does not always give the locally maximal consistent assignment (when a value selection for a given variable failed it might be still possible to assign a value to another non-instantiated variable).

3.1 Depth-Bounded Backtrack Search

Depth-Bounded Backtrack Search (DBS) is based on the idea of *limiting the depth* of complete tree search where all alternatives are explored [7]. To be able to compute a complete solution, a single alternative still can be tried if the depth limit is exceeded (shallow backtracking ignored). In general, it is possible to use another incomplete tree search technique there. Note that DBS with the depth limit equal to the number of variables corresponds to standard chronological backtracking. Figure 1 shows an abstract code of Depth-Bounded Backtrack Search.

```
procedure DBS(Variables,DepthLimit)
  if all_instantiated(Variables) then
     return true
  Var ← select_free_variable(Variables)
  Tried ← false   // ignore shallow BT
  Value ← select_first_value(Var)
  repeat
     if assign(Var,Value) then
        Tried ← true
        if DBS(Variables,DepthLimit-1) then
             return true
        unassign(Var)
     Value ← select_next_value(Var,Value)
  until Value=nil or (Tried & DepthLimit<1)
  update_best(Variables)
  return fail
end DBS
```

Figure 1. An abstract code of Depth-Bounded Backtrack Search (DBS). The bold italics parts are specific for DBS.

Let d be the size of the domains of the variables, and h be the depth limit. The worst case time complexity of DBS is $O(d^h)$, which corresponds to the number of explored branches. Recall that some values are filtered from the domains due to constraint propagation and thus fewer branches are explored. In particular, some branches and sub-trees are eliminated using constraint propagation and some branches may be terminated earlier due to inconsistency detected during variable assignment. Actually, once a leaf is reached then the algorithm stops with success.

3.2 Credit Search

Credit Search (CS) uses a similar idea like DBS – they both restrict the number of alternatives tried in each node. In particular, fewer alternatives are explored in the bottom parts of the search tree [3]. However, CS uses a finer control over branching via a so called credit. *Credit* is a natural number describing the maximal number of alternative branches to be explored. The credit c is split to k child nodes (alternatives). Each child node is allocated a base credit *(c div k)* that is increased by one for the first *(c mod k)* child nodes (the ordering of nodes is given by the value selection heuristic). In particular, credit one implies that only one alternative is tried (shallow backtracking ignored). Figure 2 shows an abstract code of the Credit Search.

```
procedure CS(Variables,Credit)
  if all_instantiated(Variables) then
     return true
  Var ← select_free_variable(Variables)
  BaseCredit ← Credit div size(dom(Var))
  RestCredit ← Credit mod size(dom(Var))
  Value ← select_first_value(Var)
  repeat
     if assign(Var,Value) then
        if RestCredit>0 then
           ValueCredit ← BaseCredit+1
           RestCredit ← RestCredit-1
        else
           ValueCredit ← BaseCredit
        end if
        Credit ← Credit-ValueCredit
        if CS(Variables,ValueCredit) then
           return true
        unassign(Var)
     Value←select_next_value(Var,Value)
  until Value=nil or Credit=0
  update_best(Variables)
  return fail
end CS
```

Figure 2. An abstract code of Credit Search (CS). The bold italics parts are specific for CS.

Let c be the credit then the worst case time complexity of CS is $O(c)$. Like in DBS, some branches may be terminated earlier due to inconsistency of the partial assignment.

3.3 Iterative Broadening

Iterative Broadening (IB) restricts the number of alternatives tried in each node by a *breadth limit* [6]. Opposite to DBS and CS, Iterative Broadening may explore a restricted number of alternatives in each node which leads to an exponential time complexity. Figure 3 shows an abstract code of IB. Again, shallow backtracking is ignored and alternatives that fail immediately are not counted in the breadth limit.

```
procedure IB(Variables,BreadthLimit)
  if all_instantiated(Variables) then
     return true
  Var ← select_free_variable(Variables)
  AvailableBreadth ← BreadthLimit
  Value ← select_first_value(Var)
  repeat
     if assign(Var,Value) then
        AvailableBreadth ← AvailableBreadth-1
        if IB(Variables,BreadthLimit) then
           return true
        unassign(Var)
     Value ← select_next_value(Var,Value)
  until Value=nil or AvailableBreadth=0
  update_best(Variables)
  return fail
end IB
```

Figure 3. An abstract code of Iterative Broadening (IB). The bold italics parts are specific for IB.

Let b be the breadth limit and n be the number of variables. Then the worst case time complexity of IB is $O(b^n)$. Thus, for any $b>1$ the algorithm has an exponential time complexity which makes it different from the above described incomplete search techniques.

3.4 Limited Assignment Number Search

Iterative Broadening allows to explore values of all variables equally while Depth-Bounded Backtrack Search and Credit Search prefer the variables selected earlier (more values are tried for these variables). We propose to modify the main idea of Iterative Broadening in such a way that the total number of attempts to assign a value to the variable is restricted. We introduce a so called LAN (Limited Assignment Number) limit indicating how many times a variable can be assigned a value. For simplicity reasons, we use an identical LAN limit for all the variables. A possible future extension could be to use different LAN limits for different variables, for example to derive the LAN limit from the domain size. When the number of assignments to the variable reaches the LAN limit, we say that the *variable expired.* The search procedure does not try to assign a value to the expired variables (shallow backtracking ignored). These variables are removed from the list of variables before a variable is selected for assignment (filter_expired/1). We call the resulting algorithm Limited Assignment Number (LAN) Search and Figure 4 shows its abstract code. Note finally, that a value can still be assigned to the expired variable via constraint propagation.

```
procedure LAN(Variables, LANlimit)
// counters were initialized to zero
// before overall search started
  FreeVariables ← filter_expired(Variables)
  if all_instantiated(FreeVariables) then
     return true
  Var ← select_free_variable(FreeVariables)
  Value ← select_first_value(Var)
  repeat
     if assign(Var,Value) then
        counter(Var) ← counter(Var)+1
        if LAN(Variables, LANlimit) then
             return true
        unassign(Var)
     Value ← select_next_value(Var,Value)
  until Value=nil or counter(Var)=LANlimit
  update_best(Variables)
  return fail
end LAN
```

Figure 4. An abstract code of Limited Assignment Number Search (LAN). The bold italics parts are specific for LAN.

Let l be the LAN limit and n be the number of variables. The number of assignments to the variable is accumulated during search so at most l times a value is tried for each variable. Thus, the worst case time complexity of LAN Search is $O(ln)$.

4. EXPERIMENTS

We compared the algorithms using random, structured, and real-life problems. In particular, we used Random Constraint Satisfaction Problems, Quasigroups with Holes, and Random Placement Problems. Random Constraint Satisfaction Problem (RCSP) is characterized by a tuple $\langle n,m,p_1,p_2\rangle$, where n is a number of variables, m is a uniform domain size, p_1 is a measure of the density of the constraint graph, and p_2 is a measure of the tightness of the binary constraints [9]. We use a so called model B of RCSP. Quasigroups With Holes (QWH) are characterized by the order (size) of a Latin square and by the number of holes [1]. The task is to fill the holes in the partial Latin square to complete it. Random Placement Problem (RPP) is characterized by the size of a rectangular area and by the number of rectangles that should be placed in this area without any overlap [10]. There are additional attributes of RPP that allow simulation of timetabling problems. We used the attributes derived from the Purdue timetabling problem [11] to obtain problems close to real world. We selected the problems from the phase-transition area where the hard problems settle (we obtained similar results for the area of over-constrained problems). We generated a hundred instances of each problem and we run the algorithms with the fail-first variable selection [5], the smallest value selection, and with the different cutoff limits (from 1 to 10 for DBS, IB, and LAN, and from 10 to 100 for CS) for each instance. The algorithms were implemented in the `clpfd` library of SICStus Prolog 3.11.2 [4] and the experiments run on 1.8 GHz Pentium under Windows XP.

Figures 5-7 show the relation between the runtime and the number of instantiated variables in the obtained solution for each studied algorithm. Each curve shows the results for a particular algorithm with an increasing cutoff limit. We use this type of comparison because it is independent of the cutoff limit. So it characterizes better the particular algorithm. A curve closer to bottom or to right borders indicates a better algorithm (either a better time for the same quality or a better quality for the same time).

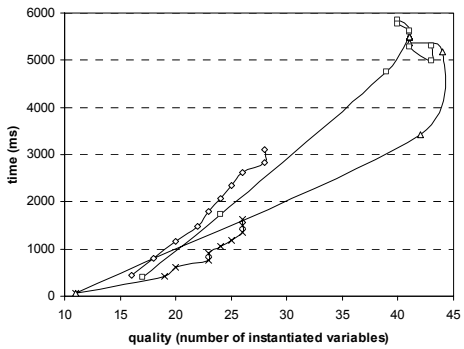

Figure 5. Relation between runtime and quality of solution for RCSP ⟨50,12, 250/1225,0.35⟩, ◇-CS, □-DBS, Δ-IB, ×-LAN.

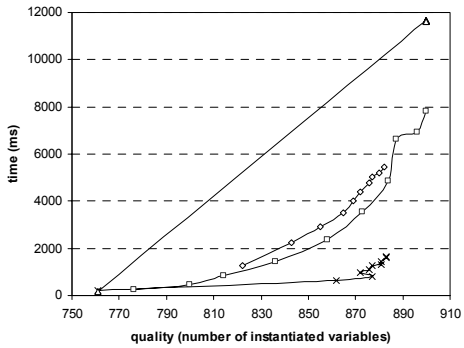

Figure 6. Relation between runtime and quality of solution for QWH, order 30 and 40% holes, ◇-CS, □-DBS, Δ-IB, ×-LAN.

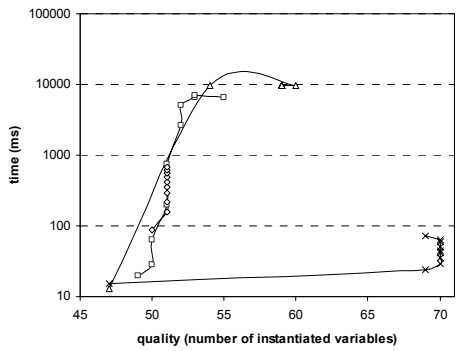

Figure 7. Relation between runtime and quality of solution for RPP, size 16 and 40 rectangles, ◇-CS, □-DBS, Δ-IB, ×-LAN.

In the experiments, LAN Search achieved the best runtime for a given solution quality so it has the best quality/performance ratio. However, the quality of the best obtained solution for LAN Search in the artificial benchmarks (RCSP and QWH) is not as good as the best quality obtained by the other algorithms. But LAN Search is a hands-down winner for RPP derived from the real-life problem both in terms of solution quality and runtime (notice the logarithmic scale for the runtime in Figure 7).

5. CONCLUSIONS

In this paper, we presented a new uniform view of existing cutoff strategies for incomplete depth-first search and we described their extension towards producing incomplete solutions. This extension is useful for solving over-constrained problems as well as hard-to-solve problems where complete assignments do not exist or it is hard to find them. We also introduced a new cutoff strategy, a so called LAN Search, based on a limited number of assignments per variable. The accomplished experiments confirmed the runtime stability of this new strategy that achieved the best quality/performance ratio. The experiments also showed that for structured problems (QWH and RPP) LAN Search produces good solutions even for small cutoff limits. Therefore, it is possible to use a small cutoff limit (2-5) for the structured problems and to achieve a good runtime. Both these features make LAN Search different from the other incomplete depth-first search techniques where finding a suitable cutoff limit is an open research question. Nevertheless, a further study of why LAN search works well, especially why it has so exceptional performance in solving a RPP, is an interesting direction for future research.

The paper focused on a single iteration of each algorithm. Once the iteration is finished, the incomplete algorithms usually restart with modified attributes to improve further the solution. The results in [11] showed that restarts can significantly improve solution quality for timetabling problems. So we believe that adding a restart strategy is a promising addition to the proposed cutoff strategy. Recall also, that a single iteration of LAN Search already achieved the far best results for this type of problem in comparison to other incomplete depth-first search techniques. Still, the quality of a single iteration for other types of problems could be improved, for example by using variable locking [2] that ensures finding locally maximal consistent assignments.

The other interesting area for further studies is a generalization of the proposed cutoff scheme. While DBS, CS, and IB are tree oriented search algorithms that can be applied to any branching scheme, LAN Search is oriented primarily to variable assignments. It is not clear whether and how it can be extended to a general tree search. Such extension would be valuable because the current results of LAN Search are quite promising and in some areas, like scheduling, different branching schemes are preferred over the variable assignments.

6. ACKNOWLEDGMENTS

The research is supported by the Czech Science Foundation under the contract No. 201/04/1102 and by Purdue University. We would like to thank anonymous reviewers for useful comments.

7. REFERENCES

[1] Dimitris Achlioptas, Carla Gomes, Henry Kautz, and Bart Selman. Generating Satisfiable Problem Instances. In *Proceedings of National Conference on Artificial Intelligence (AAAI-2000)*. AAAI Press, 2000, 256-261.

[2] Roman Barták, Tomáš Müller, Hana Rudová. A New Approach to Modelling and Solving Minimal Perturbation Problems. In *Recent Advances in Constraints, 2003*. Springer-Verlag LNAI 3010, 2004, 223-249.

[3] N. Beldiceanu, E. Bourreau, P. Chan, D. Rivreau, D. Partial Search Strategy in CHIP. In *2nd International Conference on Metaheuristics (MIC 97)*, 1997.

[4] M. Carlsson, G. Ottosson, and B. Carlson. An open-ended finite domain constraint solver. In *Programming Languages: Implementations, Logics, and Programming*. Springer-Verlag LNCS 1292, 1997.

[5] R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.

[6] Matthew L. Ginsberg and William D. Harvey. Iterative Broadening. In *Proceedings of National Conference on Artificial Intelligence*. AAAI Press, 1990, 216-220.

[7] William D. Harvey. *Nonsystematic backtracking search*. PhD thesis, Stanford University, 1995.

[8] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995, 607-615.

[9] Ewan MacIntyre, Patrick Prosser, Barbara Smith, and Toby Walsh. Random Constraint Satisfaction: theory meets practice. In Michael Maher and Jean-Francois Puget (eds.): *Principles and Practice of Constraint Programming - CP98*. Springer-Verlag LNCS 1520, 1998, 325-339.

[10] Hana Rudová. *Random Placement Problem*. http://www.fi.muni.cz/~hanka/rpp/, 2002.

[11] Hana Rudová and Keith Murray. University Course Timetabling with Soft Constraints. In Edmund Burke and Patrick De Causmaecker (eds.): *Practice And Theory of Automated Timetabling IV*. Springer-Verlag LNCS 2740, 2003, 310-328.