INCOMPLETE DEPTH-FIRST SEARCH TECHNIQUES: A SHORT SURVEY

ROMAN BARTÁK

Charles University, Faculty of Mathematics and Physics
Malostranské náměstí 2/25, 118 00 Praha 1, Czech Republic
e-mail: bartak@kti.mff.cuni.cz

Abstract: Constraint Satisfaction Problem (CSP) is a discrete combinatorial problem and hence search algorithms belong to the main constraint satisfaction techniques. There exist local search techniques exploring complete but inconsistent assignments and depth-first search techniques extending partial consistent assignments towards complete assignments. For most problems it is impossible to explore complete search space so incomplete techniques are used. In this paper, we survey incomplete depth-first search techniques, in particular, generic incomplete search techniques and discrepancy-based search techniques.

Keywords: constraint satisfaction, depth-first search, discrepancy search, incomplete solutions

1 INTRODUCTION

Incomplete search techniques are quite popular in solving real-life problems because the realistic search space cannot usually be explored exhaustively and parts of it must be skipped anyway. Instead of focusing on a small subpart of the search space that is explored completely, the incomplete search techniques spread available resources, like time, over the search space to increase chances of finding a solution[1]. This is usually done via a cutoff limit that stops complete exploration of some sub-space and forces the search algorithm to move somewhere else. In this paper, we will survey general cutoff techniques for incomplete depth-first search algorithms applied to constraint satisfaction problems. In particular, we will discuss limited depth [5], credit [4], breadth [6], and number of assignments [2].

For many problems there exist heuristics proposing values to be assigned to variables. For some problems the heuristics often lead to a solution but not always. So the question is what to do when the heuristics fail. The second group of incomplete algorithms discussed in this paper uses the number of violations of the heuristics – so called discrepancies – to guide the search procedure. These algorithms are based on observation that the number of discrepancies on the path leading to a solution is usually small. Thus, the algorithms limit the number of allowed discrepancies during search. In this paper, we will describe three representatives of discrepancy-based search algorithms, namely Limited Discrepancy Search [8], Improved Limited Discrepancy Search [9], and Depth-bounded Discrepancy Search [14].

We will present incomplete versions of the depth-first search algorithms so the algorithms do not guarantee finding a solution or proving that no solution exists. To obtain some valid assignment from incomplete search, in [1] we proposed to use incomplete assignments as an approximation of the solution. The idea is to return at least some partial assignment of variables at a given time, for example to allocate the maximal number of lectures to available rooms [12]. The variables, that are not assigned, can be seen as a hard part of the problem and they can be preferred for assignment during the next run of the search algorithm (called restart). Note also that the maximal assignment can be seen as a solution for over-constrained problems where no complete consistent assignment exists.

In this paper, we focus on the cutoff techniques only and the restart strategies will be a part of our follow-up research. Note finally, that by using the restart strategies that increase the limit, the presented algorithms can be completed.

[1] If no additional information is available, it is expected that the solutions are spread uniformly over the search space. Hence, instead of being stacked in one part of the search space, it seems better to spread the search effort over the search space as well.

2 PRELIMINARIES

A *constraint satisfaction problem (CSP)* is a triple $\Theta = (V,D,C)$, where

- $V = \{v_1, v_2,\ldots, v_n\}$ is a finite set of variables,
- $D = \{D_1, D_2,\ldots, D_n\}$ is a set of domains, where D_i is a set of possible values for the variable v_i,
- $C = \{c_1, c_2,\ldots, c_m\}$ is a finite set of constraints restricting the values that the variables can simultaneously take.

Assignment σ for a CSP $\Theta=(V,D,C)$ is a set of pairs v_i/d_i such that $v_i \in V$, $d_i \in D_i$ 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 Θ 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 consisten*t. 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 the solution[2] extending σ. In this paper, we focus on the search techniques and we expect that they are accompanied by a local consistency technique, typically generalized arc consistency.

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 in [1] we proposed a generalized view of the solution based on the notion of a *maximal consistent assignment*. 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 techniques discussed in this paper explore the locally maximal consistent assignments and remember 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 assignment we got the better approximation we have. Thus the quality of different incomplete search techniques can now be evaluated by the number of instantiated variables in the best locally maximal consistent assignment found. Note finally that such approach is also useful for solving hard-to-solve problems where incomplete assignments can be obtained.

3 GENERIC INCOMPLETE SEARCH

In this section, we will survey non-discrepancy based incomplete depth-first search algorithms, namely depth-bounded backtrack search, credit search, iterative broadening, and limited-assignment number search. 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.

Another difference from the traditional formulation of these algorithms is using a specific, possibly non-binary, branching scheme based on the selection of a value for the variable. Thus all the algorithms share the procedures for variable selection (`select_free_variable/1`) determining the shape of the search tree and value selection (`select_first_value/1`, `select_next_value/2`) determining the order in which the branches are explored. Note also, that constraint propagation is integrated into the search algorithms in a MAC-like

[2] There exist global consistency techniques that guarantee existence of the solution but the time complexity of these techniques is comparable to much simpler search algorithms.

scheme. 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. The partial assignment is stored if it is better (has a higher number of instantiated variables) than the so-far best stored assignment. 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). In [1], we proposed a technique of *variable locking* to get the locally maximal consistent assignments. It produces better assignments in terms of the number of instantiated variables but it is also more time consuming. For simplicity reasons, we do not use variable locking here and a deeper study of this technique is still required.

3.1 Depth-Bounded Backtrack Search (DBS)

Depth-Bounded Backtrack Search (DBS) is based on the idea of *limiting the depth* of complete tree search where all alternatives are explored [5]. To have a chance of obtaining a complete solution we use the technique that if the depth limit is exceeded then only a single alternative is tried (shallow backtracking ignored). In general, it is possible to use another incomplete tree search technique there. Note that if the depth limit equals to the number of variables then standard chronological backtracking is obtained. 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
```

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

Let d be the size of the domains of the variables, and h be the depth limit. Then the worst case time complexity of the algorithm is $O(d^h)$, which corresponds to the number of explored branches. Note that some values are filtered from the domains due to constraint propagation and thus fewer branches are explored. Figure 2 illustrates the branches possibly explored by the algorithm. Notice that some branches and sub-trees are eliminated using constraint propagation (dotted lines in Figure 2) and some branches may be terminated earlier due to inconsistency detected during variable assignment (not included in Figure 2). Actually, when a leaf is reached then the algorithm stops with success.

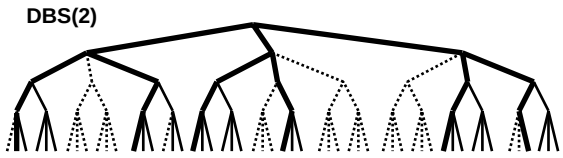

Fig. 2. Branches explored by Depth-Bounded Backtrack Search with the depth 2 (bold). The branches are explored from left to right. Dotted sub-trees are pruned via constraint propagation.

3.2 Credit Search (CS)

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 [4]. 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) in the following way. Each child node has allocated a credit *(c div k)* that is increased by one for the first *(c mod k)* child nodes (the order of the nodes is given by the value

selection heuristic). In particular, credit one implies that only one alternative is tried (shallow backtracking ignored). Figure 3 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
```

Fig. 3. An abstract code of Credit Search (CS). The bold parts are specific for CS.

Let c be the credit then the worst case time complexity of the algorithm is $O(c)$. Figure 4 illustrates the branches possibly explored by the algorithm. Again, some branches may be terminated earlier due to inconsistency of the partial assignment.

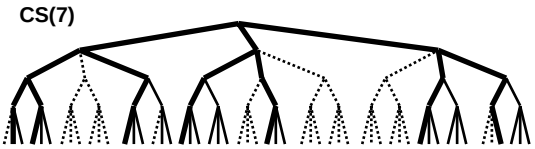

Fig. 4. Branches explored by Credit Search with the credit 7 (bold). The branches are explored from left to right. Dotted sub-trees are pruned via constraint propagation.

3.3 Iterative Broadening (IB)

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 5 shows an abstract code of IB. Again, shallow backtracking is ignored so immediately failing alternatives 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
```

Fig. 5. An abstract code of Iterative Broadening (IB). The bold 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 the algorithm 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. Figure 6 illustrates the branches explored by IB.

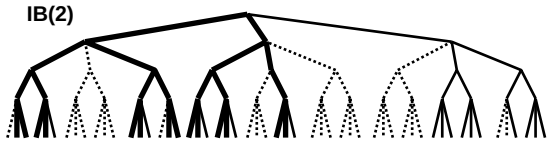

Fig. 6. Branches explored by Iterative Broadening with the breadth 2 (bold). The branches are explored from left to right. Dotted sub-trees are pruned via constraint propagation.

3.4 Limited Assignment Number Search (LAN)

Iterative Broadening has the advantage of giving equal chances to every variable to obtain its value while Depth-Bounded Backtrack Search and Credit Search prefer the variables selected earlier (more values are tried for these variables). However, the time complexity of Iterative Broadening is exponential. In [2], we proposed 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 introduced a so called LAN (Limited Assignment Number) limit indicating how many times a value can be assigned to each variable. 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 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 [13] and Figure 7 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 the 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
```

Fig. 7. An abstract code of Limited Assignment Number Search (LAN). The bold 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. Thus for each variable, at most l values are tried. Consequently, the worst case time complexity of the algorithm is $O(nl)$. Figure 8 illustrates the branches explored by LAN Search.

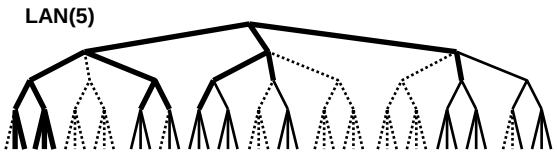

Fig. 8. Branches explored by LAN Search with the LAN limit 5 (bold). The branches are explored from left to right. Dotted sub-trees are pruned via constraint propagation.

4 DISCREPANCY BASED SEARCH

In this chapter, we will survey discrepancy based incomplete depth-first search algorithms, namely limited discrepancy search, improved limited discrepancy search, and depth-bounded discrepancy search. All these techniques are trying to recover as best as possible from the failure of the heuristics.

Heuristics guide search towards regions of the search space that are most likely to contain solutions. For some problems, the heuristics lead directly to a solution but not always. The discrepancy based search algorithms address the problem what to do when the heuristics fail. They are based on a common observation that even if the heuristic fails then the number of such failures is quite low on the branch leading to a solution. A *discrepancy* is a decision point in the search tree where the algorithm goes against the heuristic. According to the above observation, the techniques like Limited Discrepancy Search [8] and Improved Limited Discrepancy Search [9] propose to change the order of branches to be explored in such a way that the branches with a smaller number of discrepancies are explored first. These techniques treat all discrepancies alike, irrespective of their depth. However, heuristics tend to be less informed and make more mistakes at the top of the search tree. Thus, it may seem useful to explore first the branches where the discrepancies are at the top of the tree before the branches where the discrepancies are at the bottom. This idea is used by Depth-bounded Discrepancy Search [14].

Like in the previous section we will present the algorithms with possible non-binary branching based on selecting a value for the chosen variable (`select_free_variable/1`). We expect that the heuristic proposes the best value as the first value (`select_first_value/1`) and every other selected value (`select_next_value/2`) is a discrepancy. There exists another approach that increases the number of discrepancies with the distance from the first proposed value. In particular, the second value is counted as one discrepancy, the third value is counted as two discrepancies and so on. We are not aware about any study comparing these two techniques, in fact most of the discrepancy based algorithms were presented for binary branching only [3,8,9,14].

4.1 Limited Discrepancy Search (LDS)

Limited Discrepancy Search [8] was the first algorithm that highlighted the idea of preferring branches with fewer discrepancies. The algorithm uses the limit on the number of allowed discrepancies and it explores only the branches where the limit is not exceeded. Figure 9 shows an abstract code of Limited Discrepancy Search.

```
procedure LDS(Variables,DiscrLimit)
  if all_instantiated(Variables) then
    return true
  Var ← select_free_variable(Variables)
  FirstVal ← select_first_value(Var)
  Value ← select_next_value(Var,FirstVal)
  while DiscrLimit>0 and Value≠nil do
    if assign(Var,Value) then
      if LDS(Variables,DiscrLimit-1) then
         return true
      unassign(Var)
    Value ← select_next_value(Var,Value)
  end while
  update_best(Variables)
  if assign(Var,FirstValue) then
    return LDS(Variables,DiscrLimit)
end LDS
```

Fig. 9. An abstract code of Limited Discrepancy Search.

Let n be the number of variables, d be the domain size, and l be the limit on the number of discrepancies. Then Limited Discrepancy Search explores $(d-1)\sum_{i=0}^{l}\binom{n}{i}$ branches in the worst case. Figure 10 illustrates the branches explored by LDS with the limit 1. The left branch is assumed to be recommended by the heuristic there.

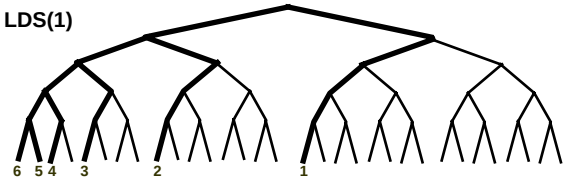

Fig. 10. Branches explored by Limited Discrepancy Search with the limit 1 (bold). The numbers indicate the order in which the branches are visited.

4.2 Improved Limited Discrepancy Search (ILDS)

The main drawback of Limited Discrepancy Search is that it visits branches more than once when increasing the limit on the number of discrepancies. In fact, all the branches explored by LDS(i) are visited again by LDS(i+1). Therefore Korf proposed Improved Limited Discrepancy Search [9] that explores only the branches with a given number of discrepancies. We present a modified version of his algorithm where we count the number of non-assigned variables instead of using the depth of the choice point. If the number of non-assigned variables is smaller or equal to the number of requested discrepancies then discrepancies are forced to achieve the requested number of discrepancies. Figure 11 shows an abstract code of Improved Limited Discrepancy Search.

```
procedure ILDS(Variables,DiscrLimit)
  if all_instantiated(Variables) then
    return true
  NoFree ← |free_variables(Variables)|
  Var ← select_free_variable(Variables)
  FirstVal ← select_first_value(Var)
  if DiscrLimit<NoFree and
     assign(Var,FirstValue) then
    if ILDS(Variables,DiscrLimit) then
      return true
  Value ← select_next_value(Var,FirstVal)
  while DiscrLimit>0 and Value≠nil do
    if assign(Var,Value) then
      if LDS(Variables,DiscrLimit-1) then
         return true
      unassign(Var)
    Value ← select_next_value(Var,Value)
  end while
  update_best(Variables)
  return fail
end ILDS
```

Fig. 11. An abstract code of Improved Limited Discrepancy Search.

In the worst case, Improved Limited Discrepancy Search explores $(d-1)\binom{n}{l}$ branches, where n is the number of variables, d is the domain size, and l is the limit on the number of discrepancies. Figure 12 illustrates the branches explored by ILDS with the limit 1. Notice also, the ILDS uses a different the order of branches than LDS.

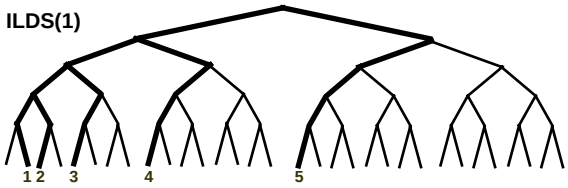

Fig. 12. Branches explored by Improved Limited Discrepancy Search with the limit 1 (bold). The numbers indicate the order in which the branches are visited.

4.3 Depth-bounded Discrepancy Search (DDS)

As we already mentioned LDS and ILDS have been proposed to explore the branches with a smaller number of discrepancies first. However, another observation says that branches with discrepancies at a top part should be preferred because the heuristics tend to be less informed and make more mistakes there. Therefore Depth-bounded Discrepancy Search has been proposed to address this issue [14]. The idea is to allow discrepancies only till some depth of the search tree and below this depth the search procedure must follow the heuristic. Note also that the depth restricts naturally the number of discrepancies that cannot be greater than the depth limit. Moreover, it is surprisingly easy to enforce that the branches are not re-visited in the next iteration by forcing a discrepancy at given depth. Figure 13 shows an abstract code of Depth-bounded Discrepancy Search.

```
procedure DDS(Variables,Depth)
  if all_instantiated(Variables) then
    return true
  Var ← select_free_variable(Variables)
  FirstVal ← select_first_value(Var)
  if Depth≠1 and
     assign(Var,FirstValue) then
    if DDS(Variables,Depth-1) then
      return true
  Value ← select_next_value(Var,FirstVal)
  while Depth>0 and Value≠nil do
    if assign(Var,Value) then
      if DDS(Variables,Depth-1) then
         return true
      unassign(Var)
    Value ← select_next_value(Var,Value)
  end while
  update_best(Variables)
  return fail
end DDS
```

Fig. 13. An abstract code of Depth-bounded Discrepancy Search.

Let d be the domain size and l be the depth limit. Then Depth-bounded Discrepancy Search explores at most $d^{(l-1)}$ branches. Figure 14 illustrates the branches explored by DDS with the depth limit 3.

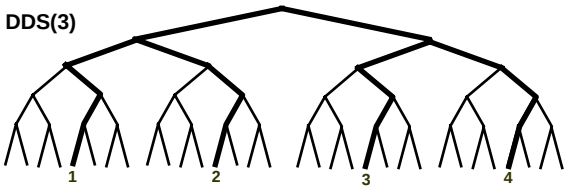

Fig. 14. Branches explored by Depth-bounded Discrepancy Search with the limit 3 (bold). The numbers indicate the order in which the branches are visited.

5 RELATED WORKS

In addition to search algorithms presented in this paper, there exist other incomplete depth-first search techniques. The only missing algorithm among the generic (non-discrepancy) incomplete depth-first search techniques that we are aware about is bounded backtrack search that limits the number of backtracks [7]. We decided to omit this algorithm because it can be modeled using the above algorithms by including a time limit. 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.

Among the discrepancy-based search techniques we are aware about two additional algorithms: interleaved depth-first search [10] and discrepancy-bounded depth-first search [3]. Interleaved depth-first search (IDFS) biases search to discrepancies high in the tree similarly to DDS. The idea of IDFS is to search in parallel several sub-trees. When a leaf is found in one sub-tree then IDFS switches to a parallel sub-tree and continues there until a leaf is found. Then it switches to the next parallel sub-tree etc. and the process is terminated when a goal leaf is found. There exists a pure version of IDFS with an exponential space complexity in search depth (Figure 15) and a limited version of IDFS with a linear space complexity in search depth. Opposite to other algorithms presented in this paper, IDFS is a complete algorithm that does not use restarts. IDFS is experimentally compared to DDS in [11].

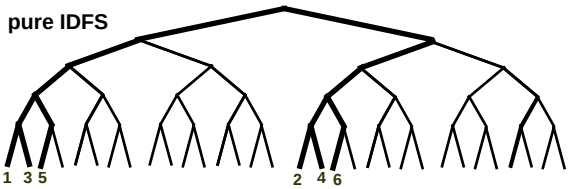

Fig. 15. The first branches explored by Interleaved depth-first search (bold). The numbers indicate the order in which the branches are visited.

Discrepancy-bounded depth-first search [3] (DBDFS) has been proposed for finding and proving optimal solutions. The algorithm is designed to minimize the number of revisited nodes while, at the same time, to preserve as much as possible the discrepancy-based order in which branches are explored. During the i-th iteration, the algorithm explores branches with discrepancies between $(i\text{-}1)k$ and $ik\text{-}1$ inclusive, where k is called a width of the search and it is an additional parameter of the algorithm. Figure 16 illustrates the search process. Notice that the branches are explored in a different order than LDS (Figure 10)

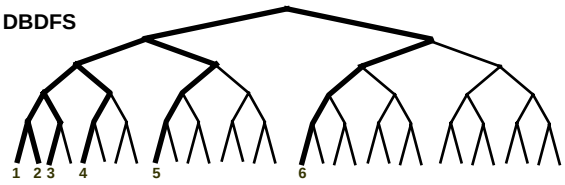

Fig. 16. Branches explored by Discrepancy-bounded depth-first search during the first iteration with width 2 (bold). The numbers indicate the order in which the branches are visited.

6 CONCLUSIONS

In the paper we presented the main incomplete depth-first search algorithms including generic and discrepancy-based techniques. The paper did not give just a survey of these algorithms but we described the algorithms in a new uniform way that makes their comparison easier. We modified some of the algorithms to work with non-binary branching that is more typical for variable assignments. Additionally, the algorithms are presented in such a way that they can keep the maximal consistent assignment found during search. A deeper comparison of generic incomplete depth-first search techniques is given in [2], where the LAN Search algorithm is introduced.

Our future work goes in the direction of studying the restart strategies for these algorithms as well as studying implementation details and performance of the discrepancy-based search techniques when applied to various types of problems.

7 ACKNOWLEDGEMENTS

The author is supported by the Czech Science Foundation under the contract No. 201/04/1102 and by the project LN00A056 of the Ministry of Education of the Czech Republic. The LAN Search technique was developed in co-operation with Hana Rudová from Masaryk University, Brno.

8 REFERENCES

1. Roman Barták, Tomáš Müller, Hana Rudová. *A New Approach to Modelling and Solving Minimal Perturbation Problems*. In Recent Advances in Constraints. Springer-Verlag LNAI 3010, pp. 233-249, 2004.
2. Roman Barták and Hana Rudová. *Limited Assignments:A New Cutoff Strategy for Incomplete Depth-First Search.* Submitted to KI-2004.
3. J. Christopher Beck and Laurent Perron. *Discrepancy-Bounded Depth First Search.* In Proceedings of CP-AI-OR, pp. 7-17, 2000.
4. Nicolas Beldiceanu, Eric Bourreau, Peter Chan, and David Rivreau. *Partial Search Strategy in CHIP*. In Proceedings of 2nd International Conference on Metaheuristics-MIC97, 1997.
5. Andrew M. Cheadle, Warwick Harvey, Andrew J. Sadler, Joachim Schimpf, Kish Shen and Mark G. Wallace. *ECLiPSe: An Introduction.* IC-Parc, Imperial College London, Technical Report IC-Parc-03-1, 2003.
6. Matthew L. Ginsberg and William D. Harvey. *Iterative Broadening*. In Proceedings of National Conference on Artificial Intelligence (AAAI-90). AAAI Press, pp. 216-220, 1990.
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 14th International Joint Conference on Artificial Intelligence, pp. 607-615, 1995.
9. Richard E. Korf. *Improved Limited Discrepancy Search.* In Proceedings of National Conference on Artificial Intelligence (AAAI-96). AAAI Press, pp. 286-291, 1996.
10. Pedro Meseguer. *Interleaved Depth-First Search.* In Proceedings of 15th International Joint Conference on Artificial Intelligence, pp. 1382-1387, 1997.
11. Pedro Meseguer and Toby Walsh. *Interleaved and Discrepancy Based Search*. In Proceedings of 13th European Conference on Artificial Intelligence, Wiley, pp. 239-243, 1998.
12. 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, pp. 310-328, 2003.
13. Kamil Veřmiřovský and Hana Rudová. *Limited Assignment Number Search Algorithm.* In Maria Bielikova (ed.): SOFSEM 2002 Student Research Forum, pp. 53-58, 2002.
14. Toby Walsh. *Depth-bounded Discrepancy Search.* In Proceedings of 15th International Joint Conference on Artificial Intelligence, pp. 1388-1393, 1997.