Limited Discrepancy AND/OR Search and its Application to Optimization Tasks in Graphical Models

Javier Larrosa, Emma Rollon
UPC Barcelona Tech
Barcelona, Spain

Rina Dechter
University of California, Irvine
Irvine, California, USA

Abstract

Many combinatorial problems are solved with a Depth-First search (DFS) guided by a heuristic and it is well-known that this method is very fragile with respect to heuristic mistakes. One standard way to make DFS more robust is to search by increasing number of discrepancies. This approach has been found useful in several domains where the search structure is a height-bounded OR tree. In this paper we investigate the generalization of discrepancy-based search to AND/OR search trees and propose an extension of the Limited Discrepancy Search (LDS) algorithm. We demonstrate the relevance of our proposal in the context of Graphical Models. In these problems, which can be solved with either a standard OR search tree or an AND/OR tree, we show the superiority of our approach. For a fixed number of discrepancies, the search space visited by the AND/OR algorithm strictly contains the search space visited by standard LDS, and many more nodes can be visited due to the multiplicative effect of the AND/OR decomposition. Besides, if the AND/OR tree achieves a significant size reduction with respect to the standard OR tree, the cost of each iteration of the AND/OR algorithm is asymptotically lower than in standard LDS. We report experiments on the min-sum problem on different domains and show that the AND/OR version of LDS usually obtains better solutions given the same CPU time.

1 Introduction

Heuristic Search [Pearl, 1984] is a standard AI method to model and solve combinatorial problems. The general idea is to abstract the space of possibilities as a graph where solutions correspond to paths from an initial node to one of the possibly many goal nodes. The method is suitable for problems in which there is a *heuristic* telling how promising it is to find a good path from each node to a goal node.

In this paper we restrict ourselves to the class of problems where the graph is a height-bounded tree (the height of a tree is the length of the longest path from the root to a leaf). For these problems, the usual search method is Depth-First and the heuristic is used to decide the expansion order of the current node successors. It is well-known that the performance of the algorithm is heavily dependent on this ordering, especially at shallow levels on the tree. If a bad decision is made on one of this early nodes, a large sub-tree must be explored before this decision can be reversed. To make things worse, it has been observed that heuristics tend to be less reliable at shallow nodes where sub-problems are both larger and less biased by the (shorter) path from the root.

Search based on *discrepancies* (LDS [Harvey and Ginsberg, 1995], ILDS [Korf, 1996], DBDS [Walsh, 1997] ,...) is a very successful method to make Depth-First more robust with respect to early heuristic mistakes [Hmida *et al.*, 2010; Sakurai *et al.*, 2000; Joncour *et al.*, 2010]. These algorithms tend to stick to the heuristic advice and only disregard it gradually as search proceeds. Disregarding the heuristic is called a *discrepancy* and discrepancy-based algorithms visits tree leaves in increasing number of discrepancies.

While standard OR search trees suffice for modeling the search space of many problems, some are better modeled with AND/OR trees [Pearl, 1984; Nilsson, 1980]. Some well-known examples are *Graphical Models* [Dechter, 2013] and *Probabilistic Planning* [Geffner and Bonet, 2013].

In this paper we introduce LDSAO, a direct extension of LDS to AND/OR search trees and demonstrate its potential in the *min-sum* problem over Graphical Models (GM). This problem includes important tasks like MPE/MAP in *Bayesian Networks* and *Markov Random Fields* [Koller and Friedman, 2009], and weighted CSPs [Meseguer *et al.*, 2006], and has been the topic of intense research in the last decade. There are two main reasons for chosing this domain. Firstly, GMs can be modeled both as an OR and an AND/OR search space so it seems a natural setting for comparing the two alternatives. Secondly, in a recent, very extensive evaluation [Allouche *et al.*, 2015], LDS was found a very competitive method.

We show that, in the GM context, each iteration of LDSAO includes the search space region that LDS would visit with the same number of discrepancies, but LDSAO can visit a much larger region. Moreover, if the AND/OR tree is significantly smaller than the OR tree, each iteration can be done in asymptotically less time. Our experiments on six different benchmarks corroborate the theory and show that LDSAO can in general find better quality solutions than LDS, given the same amount of time.

The structure of the paper is as follows. Section 2 revises basic concepts and properties of Discrepancy-based search. Section 3 makes the generalization to AND/OR trees and presents LDSAO. In the first part of Section 4 there are preliminaries on Graphical Models concepts and the description of their AND/OR search tree. In the second part, it is shown how LDSAO instantiates to Graphical Models and it is shown its theoretical advantage over LDS. Section 5 reports experimental results on the min-sum problem. Finally, in Section 6 we discuss some limitations of our approach and some directions of future work.

2 Limited Discrepancy Search for OR Search

As is customary in the Discrepancy-based context, we make the general assumption of a complete *binary* search tree with bounded height h. Leaf nodes may be goals or failures and the task of interest is to find a goal leaf. Each internal node represents a decision that has to be made to reach a goal.

We also follow the standard convention of assuming that the left child of each internal node represents following the heuristic and the right child represents going against the heuristic. The *number of discrepancies of a leaf* is the number of right turns in the path from the root to the leaf [Harvey and Ginsberg, 1995]. It is easy to see that the number of terminals with k discrepancies is $\binom{h}{k}$.

Figure 1 (left) shows a search tree of height 4. Shadow leaves represent goals. The number below each leaf indicates its number of discrepancies.

Limited Discrepancy Search (LDS) [Harvey and Ginsberg, 1995] is the algorithm that searches for a goal leaf increasing iteratively the number of discrepancies. The pseudo-code is given in Algorithm 1. The k-th iteration of the main loop will visit all the leaves having k or fewer discrepancies. The $Probe$ function is a standard recursive implementation of DFS with three particularities: $i)$ it keeps track (parameter k) of the number of discrepancies still available, $ii)$ if a discrepancy is consumed, k is decreased before the recursive call and $iii)$ if no further discrepancies are available, the algorithm does not disregard the heuristic. Since the n-th iteration visits all the leaves, the algorithm is complete. In the example of Figure 1 (left), LDS would stop in the $k = 1$ iteration where it would find the shadowed solution with 1 discrepancy.

In practice LDS is used in an any-time manner until a solution is found or the available time is exhausted. LDS has also been successfully used in optimization problems, where each leaf has an associated cost and the task of interest is to find the least-cost one. In this case the heuristic gives advise about the successor having the lowest cost leaf below and the resulting (any-time) algorithm outputs upper-bounds of the optimal solution that improve over time.

The following property shows why, in practice, it is only possible to run the first iterations of the algorithm.

Property 1. *The complexity of the k-th iteration of LDS searching on a binary tree of height h, and assuming $k < h/2$, is $O(h^{k+1})$.*

Algorithm 1: LDS

```
Function LDS()
begin
    for k = 0 ... n do
        if Probe(root, k) then return true
    return false
Function Probe(node, k)
begin
    if isLeaf(node) then return isGoal(node)
    if k = 0 then return Probe(left(node),0)
    else return (Probe(right(node),k − 1) or
    Probe(left(node),k))
```

3 Limited Discrepancy for AND/OR Search

3.1 AND/OR Search Trees

In AND/OR search trees [Nilsson, 1980; Pearl, 1984] there are two types of nodes: OR nodes and AND nodes. OR nodes represent branching points where a decision has to be made, and AND nodes represent sets of sub-goals that need to be accomplished. The two children of OR nodes are AND nodes, and the (not necessarily two) children of AND nodes are OR nodes. OR nodes are always internal nodes, while AND nodes may be internal nodes, or leaves. A leaf may be labeled as a goal or as a failure. The root of the tree is an OR node. As before, we assume that the left child follows the heuristic advice and the right child goes against it.

In the AND/OR context paths to leaves are generalized to trees. A *solution tree* is a sub-tree that (1) contains the AND/OR tree root, (2) if an OR node is in the solution tree, then exactly one of its children is in the solution tree, (3) if an AND node is in the solution tree, then all its children are. A solution tree is a *goal solution tree* if all its AND nodes with no children are labeled *goal*. The task of interest in an AND/OR search tree is to find a goal solution tree.

Figure 1 (right) shows an AND/OR tree where circles are OR nodes and squares are AND nodes. Goal leaves are shadowed. The tree has 2 goal solution trees and thicker edges show one of them.

The *height* h of an AND/OR tree is the number of OR nodes of the longest path from the root. For simplicity, we consider AND/OR trees such that all their solution trees have the same tree structure, which implies the same number of leaves, noted l. In the running example $h = 3$ and $l = 2$.

The following property follows directly from the definition,

Property 2. *The size of an AND/OR tree is $O(l \cdot 2^h)$.*

We extend now the concept of discrepancy to the AND/OR context,

Definition 1. *The number of* discrepancies of a leaf *is the number of right turns made from OR to AND nodes in the path from the root. The* number of discrepancies of a terminal tree *is the* maximum *among its leaves.*

Note that for measuring how discrepant a solution tree is we take the *maximum* among its leaves. There are several

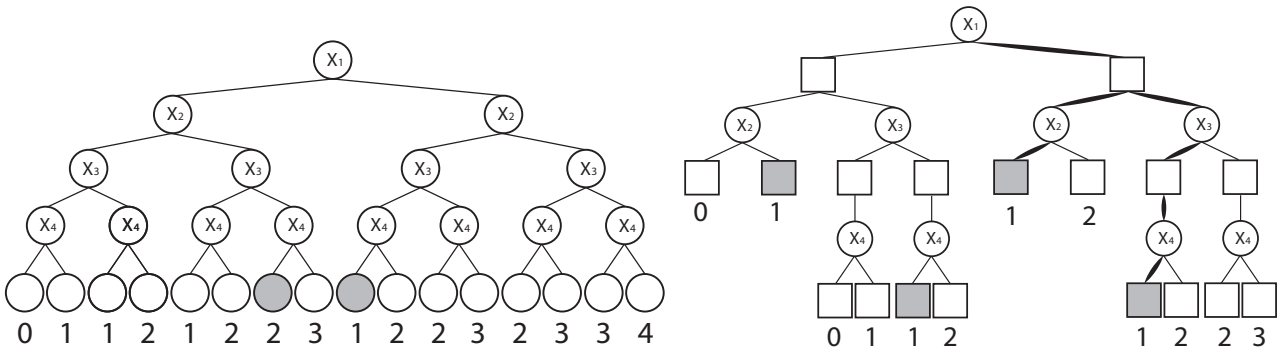

Figure 1: An OR search space (left) and an AND/OR search space (right).

other alternatives, but, as we will see, this one allows simple and efficient discrepancy-based algorithms.

Figure 1 (right) shows below each leaf its number of discrepancies. The two goal solution trees have two branches, each. In both cases the number of discrepancies is $\max\{1, 1\} = 1$.

3.2 LDSAO

LDAO (Algorithm 2) is the generalization of LDS to AND/OR trees. It searches for goal solution trees and, as in the LDS case, each iteration of the main loop allows one more discrepancy.

The main difference between LDS and LDSAO is the management of the AND nodes. In LDSAO each AND node searches recursively for sub-goals at each of its OR successors (Function $ProbeAnd$). Note that these transitions do not consume discrepancies (i.e, the value of k passes unmodified along the function) because they are not associated with choices and the heuristic does not play any role.

The following two properties show the behaviour and the complexity of LDSAO.

Property 3. *The k-th iteration of LDSAO visits all terminal trees with k or fewer discrepancies.*

Property 4. *The complexity of the k-th iteration of LDSAO searching on a binary AND/OR tree of height h, where terminal trees have l leaves, and assuming $k < h/2$, is $O(l \cdot h^{k+1})$.*

4 LDSAO for Graphical Models

In this Section we demonstrate the potential of LDSAO for Graphical Models. We start with some basic definitions and then give the properties that show the theoretical advantage of LDSAO over LDS.

4.1 Graphical Models

A *binary graphical model* is a pair $\mathcal{M} = (X, F)$, where $X = \{X_1, X_2, \ldots, X_n\}$ is the set of *boolean variables*. F is a set of non-negative real valued *cost function*, where each $f \in F$ is defined on a subset of the variables called its *scope* and noted $scope(f)$.

The *min-sum problem* is the computation of the minimum cost complete assignment,

Algorithm 2: LDSAO

```
Function LDSAO()
begin
    for k = 0..n do
        if ProbeOr(root, k) then return true
    return false
Function ProbeOr(nodeOr, k)
begin
    if k = 0 then return
    ProbeAnd(left(nodeOr),0)
    return ProbeAnd(right(nodeOr),k − 1) or
    ProbeAnd(left(nodeOr),k)
Function ProbeAnd(nodeAnd, k)
begin
    if isLeaf(nodeAnd) then
        return isGoal(nodeAnd)
    for nodeOr ∈ Successors(nodeAnd) do
        if not ProbeOr(nodeOr,k) then
            return false
    return true
```

$$x^* = argmin_{x \in \mathbf{X}} \sum_{f \in \mathbf{F}} f(x^f) \quad (1)$$

where $\mathbf{X}$ denotes the set of complete assignments to the variables and x^f denotes the projection of x on the scope of f.

The *primal graph* G of a graphical model $\mathcal{M}$ has the variable as nodes, and an edge (X_p, X_q) is in G iff the pair of variables X_p, X_q appears in the scope of any $f \in \mathbf{F}$. The primal graph is a useful tool, because it is a graphical representation of the problem structure in terms of explicit interaction between variables.

A *pseudo-tree* $\mathcal{T} = (V, E')$ of the primal graph $G = (V, E)$ is a rooted tree over the same set of vertexes V such that every edge in $E - E'$ must connect a node to one of its ancestors in $\mathcal{T}$. The root is noted X_{root}.

The purpose of the pseudo-tree is to make explicit how the graphical model can be *broken* into independent components that can be solved separately. More precisely, it captures conditional independencies based on cost-function

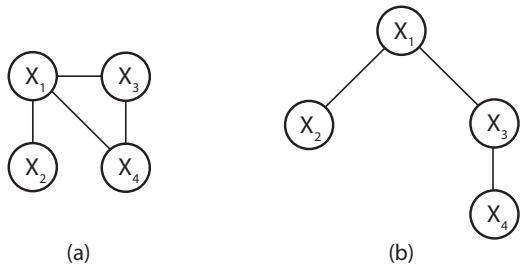

Figure 2: (a) A primal graph G of a graphical model over 4 variables; and (b) A pseudo-tree of G

scopes. If a partial assignment of variables follows the top-down order of the tree, each unassigned sub-tree is an independent sub-problem (conditioned to the assignment) and can be solved separately. The algorithms that exploit such independences are those that use the AND/OR tree to be defined next [Dechter and Mateescu, 2007].

Figure 2 (left) depicts the primal graph G of a graphical model with 4 variables. One possible set of cost functions that would have such a graph has the following scopes $\{\{X_1, X_2\}, \{X_1, X_3\}, \{X_1, X_4\}, \{X_3, X_4\}\}$. Figure 2 (right) depicts one possible pseudo-tree $\mathcal{T}$ for G. The pseudo-tree indicates that the assignment of variable X_1 will produced two conditionally independent sub-problems whose set of variables will be the two sub-trees rooted by its two children, X_2 and X_3.

4.2 OR and AND/OR search trees for Graphical Models

A search space must capture the set of all possible assignments. For Graphical Models there are two possibilities: an OR, and an AND/OR tree. The obvious one is an OR tree of height n (the number of variables). Each node is labeled with a variable and its two children correspond to its two possible assignments. It is clear that the size of such a tree is $O(2^n)$. The OR tree in Figure 1 (left) corresponds indeed to the Graphical Model with the primal graph in Figure 2 (left).

A more compact approach is the AND/OR search tree, an alternative search space that exploits conditional decompositions. It is driven by a pseudo-tree $\mathcal{T}$. Each OR node is labeled with a variable $X_p \in \mathbf{X}$ and its two children are AND nodes, each one corresponding to one possible assignment of X_p (left child follows the heuristic advise, right child goes against it). The children of an AND node are OR nodes labeled with the children of X_p in $\mathcal{T}$. The children of an AND node represent the independent sub-problems given the assignments from the root up to that node. The root of the AND/OR search *tree* is an OR node labeled with X_{root}, the root of $\mathcal{T}$. The AND/OR tree in Figure 1 (right) corresponds to the Graphical Model with the primal graph in Figure 2 (left) and pseudo-tree in Figure 2 (right).

Clearly, the size of the AND/OR tree of a Graphical Model is $O(l \cdot 2^h)$ where h and l are the height and the number of leaves of the pseudo-tree, respectively [Dechter and Mateescu, 2007]. In our running example, the size of the OR tree is exponential on its height (4) while the size of the AND/OR tree is exponential in its height (3) which is also the height of its corresponding pseudo-tree.

4.3 LDS vs LDSAO

We now compare the performance of Limited Discrepancy Search over the OR tree (LDS) versus the AND/OR counterpart (LDSAO). First, we show that each iteration of LDSAO is at least as powerful in terms of visited terminals than the same iteration of LDS.

It can be seen that there is a one-to-one mapping between complete assignments of the variables, leaves in the OR tree and solution trees in the AND/OR tree. It follows the fact that in both trees there is a left-or-right turn below each OR node and there is one OR node associated to each variable.

Let x be an assignment of all the variables of a Graphical Model. The number of discrepancies of the leaf associated to x in the OR tree (according to the standard definition) is noted $D_{OR}(x)$, the number of discrepancies of the solution tree associated to x in the AND/OR tree (according to Definition 1) is noted $D_{AND/OR}(x)$. It can be seen that,

Lemma 1. *For any complete assignment x, it holds that $D_{AND/OR}(x) \leq D_{OR}(x)$.*

Proof. In both cases, they have the same left and right turns in their associated search spaces. By definition $D_{OR}(x)$ is the number of right turns, while $D_{AND/OR}(x)$ takes the maximum over diferent fragments, which may be smaller or equal, but not larger. □

What follows is a direct consequence of the previous lemma,

Property 5. *If assignment x, is visited during the k-th iteration of LDS, then it is visited during the k-th iteration of LDSAO.*

The two goal leaves in Figure 1 (left) correspond to the same assignments as the two goal solution trees in Figure 1 (right). Note that in one of them the number of discrepancies decreases in the AND/OR tree. As a result, LDSAO would visit the two of them in the $k = 1$ iteration.

Regarding time complexity it is clear that the k-th iteration of LDS and LDSAO (assuming $k \ll h, n$) is $O(n^{k+1})$ and $O(l \cdot h^{k+1})$, respectively. Since h is smaller than n (and in many important cases much smaller), LDSAO will be faster than LDS.

The actual set of assignments that LDSAO will visit and LDS will not at the k-iteration depends on the pseudo-tree structure. We illustrate it with the $k = 1$ case in three different simple scenarios where we also discuss the different time complexities.

- Consider a pseudo-tree whose root has the rest of nodes as children (that is, $h = 2, l = n - 1$). The $k = 1$ iteration of LDSAO will visit some assignments that under the OR tree perspective will have up to $n - 1$ discrepancies. The cost of this iteration will be time $O(n)$.
- Consider a pseudo-tree whose root has two children, each one rooting a chain of size $n/2$ and $n/2 - 1$, respectively (that is, $h = (n/2) + 1, l = 2$). The $k = 1$ iteration of LDSAO will visit assignments that under the

OR perspective will have up to 2 discrepancies. The cost of such iteration will be $O(n^2)$, similar to the LDS case.

- If the pseudo-tree is a complete binary tree (that is, $h = \log n$ and $l = n/2$), the $k = 1$ iteration of LDSAO will visit assignments that under the OR tree would have up to $O(n/2)$ discrepancies. The cost of such iteration will be $O(n \cdot (\log n)^2)$.

5 Experimental Results

In this section we report results comparing LDS vs LDSAO as any-time schemes in the min-sum problem of Graphical Models. Note that LDS was recently found extremely efficient in that domain (see e.g. Figure 4 in [Allouche *et al.*, 2015]). All executions had a time limit of 1 hour. We have experimented with six different well-known benchmarks. Instances have been taken from `http://genoweb.toulouse.inra.fr/ degivry /evalgm` and `http://bioinfo.cs.technion.ac. il/superlink`. All instances have been read in *wcsp* format and in those containing more than one connected component, only the largest one has been considered.

In our experiments both algorithms guided the search with the static Mini-Bucket-Elimination heuristic [Kask and Dechter, 1999; Ihler *et al.*, 2012; L. Otten and Dechter, 2012]. This heuristic has a control parameter, called i-bound, that allows to trade memory for accuracy. In the experiments the two algorithms ran with the same i-bound (10 for all benchmarks except for *Linkage* and *Type4 pedigree* where it was set to 15 and 16, respectively). Since many instances contain non-boolean variables, we considered all assignments but the one with the highest heuristic value as a discrepancy (ties were broken lexicographically).

Figure 3 shows any-time plots for one instance from each benchmark (note the logarithmic scale of time). Results for the rest of instances (a total of 138) can be downloaded from `http://www.cs.upc.edu/larrosa/ lds-experiments.pdf`. It can be seen that consistently LDSAO has better solutions than LDS given the same amount of time. As previously discussed, there are two reasons why LDSAO is likely to be superior than LDS: it iterates faster and it visits more complete assignments at each iteration. To be able to separately assess these two factors, plots also indicate the ending point of each iteration. For instance, in the *Spot5 5.wcsp* instance (top-left plot) the $k = 3$ iteration ends after 20 and 400 seconds for LDSAO and LDS, respectively. Besides, at the end of this iteration LDSAO has an upper bound of 150, while LDS has an upper bound of 151. Similar results can be observed in the *combinatorial auction, linkage and grid* instances (i.e, top and middle rows). In the *Type 4 pedigree* and *DBN* insntaces (i.e, bottom row) it can be observed that the advantage comes exclusively from the faster iteration of LDSAO. It means that the additional solutions that LDSAO visits for a given iteration do not provide any improvement in the upper bound. This is a situation that we have encountered in many other instances. There are two reasons that explain this phenomenon: i) the additional solutions that LDSAO may visit with respect to LDS contain their discrepancies at deep levels of the search space and Branch-and-Bound usually prunes before these discrepancies may occur and ii) discrepancies at deep levels are more unlikely to do any good since the heuristic is much more accurate there [Walsh, 1997].

As it can be seen in the plots, LDS updates more frequently the value of its best-so-far solution (see for instance the $k = 2$ iteration in the bottom-right *rus-2* plot). The reason is that LDS is fully any-time in the sense that it can identify an improving solution in the middle of an iteration and report it immediately. In the LDSAO case, the situation is more complex. The algorithm goes from one sub-problem to another sequentially trying to find good sub-solutions in one sub-problem after the others. Consequently, LDSAO cannot report improvements until the end of each iteration. The natural way to overcome this drawback is to incorporate to LDSAO a breadth rotation scheme like the one proposed in [Otten and Dechter, 2012].

6 Conclusions and Future Work

In this paper we have presented a generalization of LDS [Harvey and Ginsberg, 1995] to AND/OR tree search and demonstrated its practical interest on the min-sum problem for Graphical Models. In this domain, we have shown theoretically and corroborated empirically that our LDSAO can search more solutions with (asymptotically) less effort.

One practical disadvantage of our approach inherited from the AND/OR tree usage is that LDSAO must use a static variable ordering (while LDS does not). This is an important limitation that needs to be addressed, since some state-of-the-art min-sum solvers find dynamic orderings instrumental to their success. One possible solution is to allow LDSAO to use a semi-dynamic variable ordering, where variables can be assigned in any order within pseudo-tree chains. Another possibility is to have more than one pseudo-tree, and use one or another depending on what variables are dynamically selected at the first levels of the search.

The investigation reported in this paper leaves room to many lines of future work. First, variations of LDS such as ILDS [Korf, 1996] or DBDS [Walsh, 1997] to the AND/OR context should be explored. We also believe that LDSAO can be improved to do a more intelligent management of the search effort. For instance, in problems with an unbalanced pseudo-tree it seems reasonable to allow more discrepancies on the largest sub-problems. More discrepancies should also be allowed on the harder sub-problems. Finally, we also want to explore the applicability of LDSAO in on-line planning [Geffner and Bonet, 2013] and other domains where AND/OR structures are used.

Acknowledgments

We thank William Lam for his help in running the experiments. This work was suppported in part by MINECO under project TIN2015-69175-C4-3-R, by NSF grants IIS-1065618, IIS-1526842, and IIS-1254071, and by the US Air Force under Contract No. FA8750-14-C-0011 under the DARPA PPAML program.

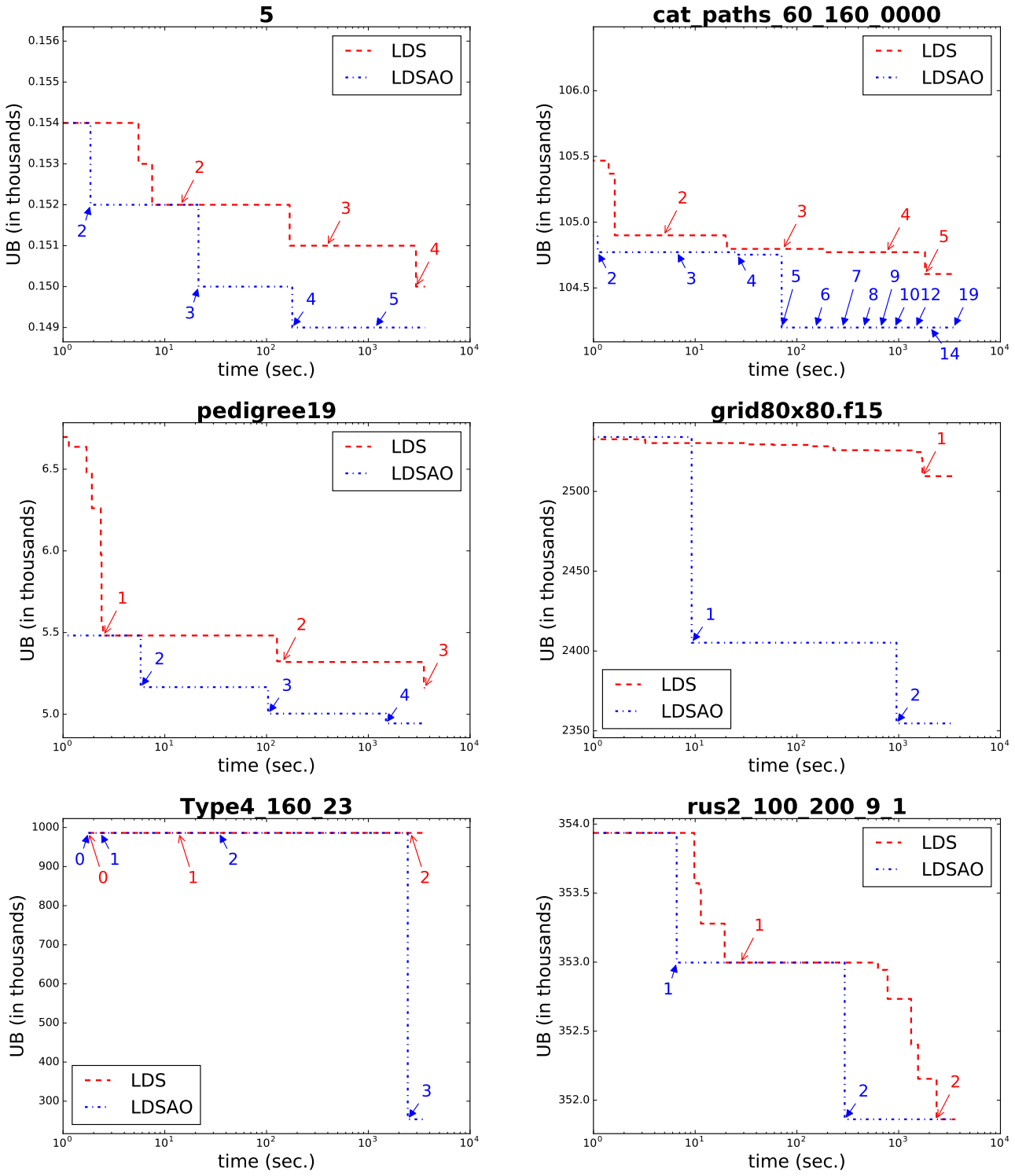

Figure 3: Experimental results on one representative instance from six different benchmarks. Top: SPOT5, Auctions. Middle: Linkage, Grids. Bottom: Type4 pedigrees, DBN. Plots report quality of best-found solution versus time. Note the logarithmic scale of time. Arrows indicate the end of each iteration.

References

[Allouche *et al.*, 2015] David Allouche, Simon de Givry, George Katsirelos, Thomas Schiex, and Matthias Zytnicki.

for weighted CSP. In *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, pages 12–29, 2015.

[Dechter and Mateescu, 2007] Rina Dechter and Robert Mateescu. And/or search spaces for graphical models. *Artif. Intell.*, 171(2-3):73–106, 2007.

[Dechter, 2013] Rina Dechter. *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013.

[Geffner and Bonet, 2013] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013.

[Harvey and Ginsberg, 1995] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, pages 607–615, 1995.

[Hmida *et al.*, 2010] Abir Ben Hmida, Mohamed Haouari, Marie-José Huguet, and Pierre Lopez. Discrepancy search for the flexible job shop scheduling problem. *Computers & Operations Research*, 37(12):2192–2201, 2010.

[Ihler *et al.*, 2012] Alexander Ihler, Natalia Flerova, Rina Dechter, and Lars Otten. Join-graph based cost-shifting schemes. In *Uncertainty in Artificial Intelligence (UAI)*, pages 397–406. AUAI Press, Corvallis, Oregon, August 2012.

[Joncour *et al.*, 2010] C. Joncour, S. Michel, Ruslan Sadykov, D. Sverdlov, and François Vanderbeck. Column generation based primal heuristics. *Electronic Notes in Discrete Mathematics*, 36:695–702, 2010.

[Kask and Dechter, 1999] K. Kask and R. Dechter. Branch and bound with mini-bucket heuristics. *Proc. IJCAI-99*, 1999.

[Koller and Friedman, 2009] D. Koller and N. Friedman. *Probabilistic Graphical Models*. MIT Press, 2009.

[Korf, 1996] Richard E. Korf. Improved limited discrepancy search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, August 4-8, 1996, Volume 1.*, pages 286–291, 1996.

[L. Otten and Dechter, 2012] K. Kask L. Otten, A. Ihler and R. Dechter. Winning the pascal 2011 map challenge with enhanced and/or branch-and-bound. In *Workshop on DISCML 2012 (a workshop of NIPS 2012)*, 2012.

[Meseguer *et al.*, 2006] Pedro Meseguer, Francesca Rossi, and Thomas Schiex. Soft constraints. In *Handbook of Constraint Programming*, pages 281–328. 2006.

[Nilsson, 1980] N. J. Nilsson. *Principles of Artificial Intelligence*. Tioga, Palo Alto, CA, 1980.

[Otten and Dechter, 2012] Lars Otten and Rina Dechter. Anytime AND/OR depth-first search for combinatorial optimization. *AI Commun.*, 25(3):211–227, 2012.

[Pearl, 1984] J. Pearl. *Heuristics: Intelligent Search Strategies*. Addison-Wesley, 1984.

[Sakurai *et al.*, 2000] Yuko Sakurai, Makoto Yokoo, and Koji Kamei. An efficient approximate algorithm for winner determination in combinatorial auctions. In *EC*, pages 30–37, 2000.

[Walsh, 1997] Toby Walsh. Depth-bounded discrepancy search. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*, pages 1388–1395, 1997.