Ann Math Artif Intell (2017) 79:77–128
DOI 10.1007/s10472-015-9495-1

Weighted heuristic anytime search: new schemes for optimization over graphical models

Natalia Flerova[1] · Radu Marinescu[2] · Rina Dechter[1]

Published online: 9 January 2016

Abstract Weighted heuristic search (best-first or depth-first) refers to search with a heuristic function multiplied by a constant w [31]. The paper shows, for the first time, that for optimization queries in graphical models the weighted heuristic best-first and weighted heuristic depth-first branch and bound search schemes are competitive energy-minimization anytime optimization algorithms. Weighted heuristic best-first schemes were investigated for path-finding tasks. However, their potential for graphical models was ignored, possibly because of their memory costs and because the alternative depth-first branch and bound seemed very appropriate for bounded depth. The weighted heuristic depth-first search has not been studied for graphical models. We report on a significant empirical evaluation, demonstrating the potential of both weighted heuristic best-first search and weighted heuristic depth-first branch and bound algorithms as approximation anytime schemes (that have sub-optimality bounds) and compare against one of the best depth-first branch and bound solvers to date.

Keywords Graphical models · Heuristic search · Anytime weighted heuristic search · Combinatorial optimization · Weighted CSP · Most Probable Explanation

Mathematics Subject Classification (2010) 68T37 · 68R05

✉ Natalia Flerova
nflerova@uci.edu

Radu Marinescu
radu.marinescu@ie.ibm.com

Rina Dechter
dechter@uci.edu

[1] University of California Irvine, Irvine, CA, USA

[2] IBM Research, Dublin, Ireland

1 Introduction

The idea of weighing the heuristic evaluation function guiding search by a fixed (or varying) constant is well known [31]. It was revived in recent years in the context of path-finding domains, where a variety of algorithms using this concept emerged. The attractiveness of this scheme of *weighted heuristic search* is in transforming best-first search into an anytime scheme, where the weight serves as a control parameter, trading-off time, memory and accuracy. The common approach is to have multiple executions, gradually reducing the weight along some schedule. A valuable by-product of these schemes is that the weight offers a sub-optimality bound on the generated cost.

In this paper we investigate the potential of weighted heuristic search for probabilistic and deterministic graphical models queries. Because graphical models are characterized by having many solutions which are all at the same depth, they are typically solved by depth-first schemes. These schemes allow flexible use of memory and they are inherently anytime (though require a modification for AND/OR spaces). Best-first search schemes, on the other hand, do not offer a significant advantage over depth-first schemes for this domain, yet they come with a significant memory cost and a lack of anytime behavior, and therefore are rarely used. In this paper we show that weighted heuristics can facilitate an effective and competitive best-first search scheme, useful for graphical models as well. The following paragraphs elaborate.

In the path-finding domain, where the solution length varies (e.g., planning), best-first search, and especially its popular variant A* [12], is clearly favoured among the exact schemes. However, A*'s exponential memory needs, coupled with its inability to provide a solution any time before termination, lead to extension into more flexible anytime schemes based on the *Weighted A** (WA*) [31]. Several anytime weighted heuristic best-first search schemes were proposed in the context of path-finding problems in the past decade [11, 22, 32, 32, 35, 36].

Our contribution We extend and evaluate weighted heuristic search for graphical models. As a basis, we used AND/OR Best First search (AOBF) [26] and AND/OR Branch and Bound search (AOBB) [25], both described in Section 2.3. We compare against a variant called Breadth-Rotating AND/OR Branch and Bound (BRAOBB) [29]. BRAOBB (under the name daoopt) was instrumental in winning the 2011 Probabilistic Inference Challenge[1] in all optimization categories. This algorithm also won the second place for the 20 minute and 1 hour time bounds in the MAP category, as well as the first place for all time bounds in the MMAP category during the UAI Inference Challenge 2014.[2] We also compare our schemes against traditional depth-first branch and bound (DFBB) and A* exploring the OR search graph and against Stochastic Local Search (SLS).

We explored a variety of weighted heuristic schemes. After an extensive preliminary empirical evaluation, the two best-first schemes that emerged as most promising were wAOBF and wR-AOBF. Both apply weighted heuristic best-first search iteratively while

[1] http://www.cs.huji.ac.il/project/PASCAL/realBoard.php

[2] http://www.hlt.utdallas.edu/~vgogate/uai14-competition/leaders.html

decreasing the weight w. wAOBF starts afresh at each iteration, while wR-AOBF reuses search efforts from previous iterations, extending ideas presented in Anytime Repairing A* (ARA*) [22]. Our empirical analysis revealed that weighted heuristic search can be competitive with BRAOBB on a significant number of instances from a variety of domains.

We also explored the benefit of weighting the heuristic function for depth-first search, resulting in wAOBB and wBRAOBB schemes. The weights facilitate an alternative anytime approach and, most importantly, equip those schemes with sub-optimality guarantees. Our empirical evaluation showed that for many instances our algorithms yielded best results.

To explain the behavior of weighted heuristic search we introduce a notion of *focused search* that yields a fast and memory efficient search. Moreover, we derive the optimal value of the weight that a) yields a greedy search with least loss of accuracy; b) when computed over an arbitrary solution path provides a guarantee on the solution accuracy.

Note that for the purpose of this work we intentionally focus primarily on the complete schemes that guarantee optimal solutions if given enough time and space. Thus, many approximate schemes developed for graphical models, e.g., [9, 13, 27, 34, 37] remain beyond the scope of our consideration, as do a number of exact methods, developed for weighted CSP problems, e.g., [7, 21]. A relevant work on generating both a lower bound and an upper bound in an anytime fashion and providing a gap of optimality when terminating was done by [2], though their approach is orthogonal to our investigation of the power of weighted heuristic search in generating anytime schemes with optimality guarantees.

The paper is organized as follows. In Section 2 we present relevant background information on best-first search (2.1), graphical models (2.2) and AND/OR search spaces (2.3). In Section 3 we consider the characteristics of the search space explored by the weighted heuristic best-first search and reason about values of the weights that make this exploration efficient. Section 4 presents our extension of anytime weighted heuristic best-first schemes to graphical models. Section 5 shows the empirical evaluation of the resulting algorithms. It presents the overview of methodology (5.1), discusses the impact of the weight on runtime and accuracy of solutions found by the weighted heuristic best-first (5.2), reports on our evaluation of different weight policies (5.3) and compares the anytime performances of our two anytime weighted heuristic best-first schemes against the previously developed schemes (5.4). Section 6 introduces the two anytime weighted heuristic depth-first branch and bound schemes (6.1) and presents their empirical evaluation (6.2). Section 7 summarizes and concludes.

2 Background

2.1 Best-first search

Consider a search space defined implicitly by a set of states (the nodes in the graph), operators that map states to states having costs or weights (the directed weighted arcs), a starting state n_0 and a set of goal states. The task is to find the least cost solution path from n_0 to a goal [28], where the cost of a solution path is the sum of the weights or the product of the weights on its arcs.

Best-first search (BFS) maintains a graph of explored paths and a frontier of OPEN nodes. It chooses from OPEN a node n having the lowest value of an evaluation function $f(n)$, expands it, and places its child nodes in OPEN. The most popular variant, A*, uses $f(n) = g(n) + h(n)$, where $g(n)$ is the current minimal cost from the root to n, and $h(n)$ is a heuristic function that estimates the optimal cost $h^*(n)$ to go from n to a goal node. For a minimization task, $h(n)$ is *admissible* if $\forall n\ h(n) \leq h^*(n)$. Heuristic $h(n)$ is *consistent*, if for every node n and for every successor n' of n it true that $h(n) \leq c(n, n') + h(n')$, where $c(n, n')$ is the weight of the arc (n, n').

Weighted A* search (WA*) [31] differs from A* only in using the evaluation function: $f(n) = g(n) + w \cdot h(n)$, where $w > 1$. Higher values of w typically yield greedier behavior, finding a solution earlier during search and with less memory. WA* is guaranteed to terminate with a solution cost C such that $C \leq w \cdot C^*$, where C^* is the cost of the optimal solution. Such solution is called w-optimal.

Formally, after [31]:

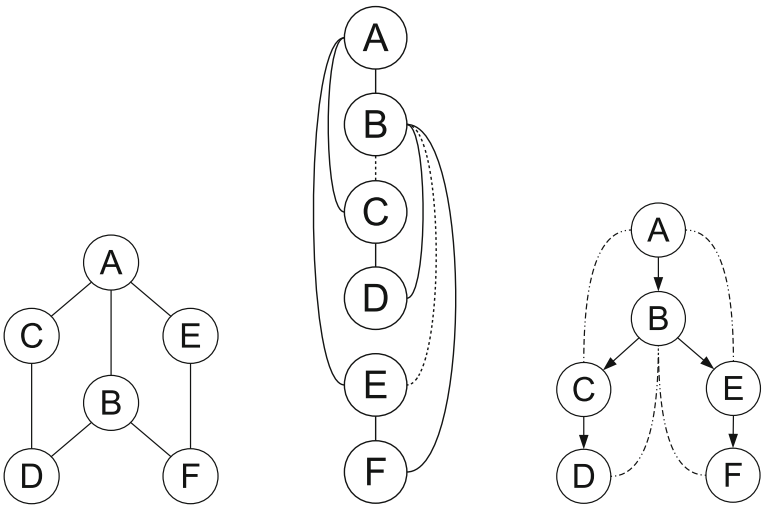
Theorem 1 *The cost C of the solution returned by Weighted A* is guaranteed to be within a factor of w from the optimal cost C^*.*

Proof Consider an optimal path to the goal t. If all nodes on the path were expanded by WA*, the solution found is optimal and the theorem holds trivially. Otherwise, let n' be the deepest node on the optimal path, which is still on the OPEN list when WA* terminates. It is known from the properties of A* search [30] that the unweighted evaluation function of n' is bounded by the optimal cost: $g(n') + h(n') \leq C^*$. Using some algebraic manipulations: $f(n') = g(n') + w \cdot h(n')$, $f(n') \leq w \cdot (g(n') + h(n'))$. Consequently, $f(n') \leq w \cdot C^*$.

Let n be an arbitrary node expanded by WA*. Since it was expanded before n', $f(n) \leq f(n')$ and $f(n) \leq w \cdot C^*$. It holds true to all nodes expanded by WA*, including goal node t: $g(t) + w \cdot h(t) \leq w \cdot C^*$. Since $g(t) = C$ and $h(t) = 0$, $C \leq w \cdot C^*$. □

2.2 Graphical models

A *graphical model* is a tuple $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \bigotimes \rangle$, where $\mathbf{X} = \{X_1, \ldots, X_n\}$ is a set of variables and $\mathbf{D} = \{D_1, \ldots, D_n\}$ is the set of their finite domains of values. $\mathbf{F} = \{f_1(X_{\mathbf{S}_1}), \ldots, f_r(X_{\mathbf{S}_r})\}$ is a set of non-negative real-valued functions defined on subsets of variables $\mathbf{X}_{\mathbf{S}_i} \subseteq \mathbf{X}$, called *scopes* (i.e., $\forall i\ f_i : \mathbf{X}_{\mathbf{S}_i} \rightarrow \mathbb{R}^+$). The set of function scopes implies a *primal graph* whose vertices are the variables and which includes an edge connecting any two variables that appear in the scope of the same function (e.g. Fig. 1a). Given an ordering of the variables, the *induced graph* is an ordered graph, such that each node's earlier neighbours are connected from last to first, (e.g., Fig. 1b). The maximum, over all nodes in the induced graph, number of preceding neighbours is called *induced width* w^* (not to be confused with the weight w). For more details see [18]. The combination operator $\bigotimes \in \{\prod, \sum\}$ defines the complete function represented by the graphical model $\mathcal{M}$ as $\mathcal{C}(\mathbf{X}) = \bigotimes_{j=1}^{r} f_j(\mathbf{X}_{\mathbf{S}_j})$.



(a) Primal graph. (b) Induced graph. (c) Pseudotree.

Fig. 1 Example problem with six variables, induced graph along ordering A, B, C, D, E, F, corresponding pseudo tree

The most common optimization task is known as the *most probable explanation* (MPE) or *maximum a posteriori* (MAP), in which we would like to compute the optimal value C^* and/or its optimizing configuration x^*:

$$C^* = C(\mathbf{x}^*) = \max_{\mathbf{X}} \prod_{j=1}^{r} f_j\left(\mathbf{X}_{\mathbf{S}_j}\right) \tag{1}$$

$$\mathbf{x}^* = argmax_{\mathbf{X}} \prod_{j=1}^{r} f_j\left(\mathbf{X}_{\mathbf{S}_j}\right) \tag{2}$$

The MPE/MAP task is often converted into negative log-space and solved as an *energy minimization* (min-sum) problem. This is also known as the Weighted CSP (WCSP) problem [25] and is defined as follows:

$$C^* = C(\mathbf{x}^*) = \min_{\mathbf{X}} \sum_{j=1}^{r} f_j(\mathbf{X}_{\mathbf{S}_j}) \tag{3}$$

Bucket elimination (BE) [1, 4] solves MPE/MAP (WCSP) problems exactly by eliminating the variables in sequence. Given an elimination order BE partitions the functions into buckets, each associated with a single variable. A function is placed in the bucket of its argument that appears later in the ordering. BE processes each bucket, from last to first, by multiplying (summing for WCSP) all functions in the current bucket and eliminating the bucket's variable by maximization (minimization for WCSP), resulting in a new function which is placed in a lower bucket. The complexity of BE is time and space exponential in the induced width corresponding to the elimination order.

Mini-bucket elimination (MBE) [6] is an approximation algorithm designed to avoid the space and time complexity of full bucket elimination by partitioning large buckets into smaller subsets, called *mini-buckets*, each containing at most i (called i-bound) distinct variables. The mini-buckets are processed separately. MBE generates an upper bound on the optimal MPE/MAP value (lower bound on the optimal WCSP value). The complexity of the algorithm, which is parameterized by the i-bound, is time and space exponential in i only. When i is large enough (i.e., $i \geq w^*$), MBE coincides with full BE. Mini-bucket elimination is often used to generate heuristics for both best-first and depth-first branch and bound search over graphical models [15, 16].

2.3 AND/OR search spaces

The concept of AND/OR search spaces for graphical models has been introduced to better capture the problem structure [5]. A *pseudo tree* of the primal graph defines the search space and captures problem decomposition (e.g., Fig. 1c).

Definition 1 A *pseudo tree* of an undirected graph $G = (V, E)$ is a directed rooted tree $\mathcal{T} = (V, E')$, such that every arc of G not included in E' is a back-arc in $\mathcal{T}$, namely it connects a node in $\mathcal{T}$ to an ancestor in $\mathcal{T}$. The arcs in E' may not all be included in E.

Given a graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \bigotimes \rangle$ with primal graph G and a pseudo tree $\mathcal{T}$ of G, the *AND/OR search tree* $S_{\mathcal{T}}$ based on $\mathcal{T}$ has alternating levels of OR and AND nodes. Its structure is based on the underlying pseudo tree. The root node of $S_{\mathcal{T}}$ is an OR node labelled by the root of $\mathcal{T}$. The children of an OR node $\langle X_i \rangle$ are AND nodes labelled with value assignments $\langle X_i, x_i \rangle$ (or simply $\langle x_i \rangle$); the children of an AND node $\langle X_i, x_i \rangle$ are OR nodes labelled with the children of X_i in $\mathcal{T}$, representing conditionally independent subproblems. Identical subproblems, identified by their *context* (the partial instantiation that separates the subproblem from the rest of the problem graph), can be merged, yielding an *AND/OR search graph* [5]. Merging all context-mergeable nodes yields the *context-minimal AND/OR search graph*, denoted by $C_{\mathcal{T}}$ (e.g., Fig. 2). The size of the context minimal AND/OR graph is exponential in the induced width of the primal graph G along a depth-first traversal of $\mathcal{T}$ [5].

The arcs from nodes X_i to $\langle X_i, x_i \rangle$ in an AND/OR search graph are annotated by the weights $c(X_i, \langle X_i, x_i \rangle)$ derived from the cost functions in $\mathbf{F}$. Namely $c(X_i, \langle X_i, x_i \rangle)$ equals to the combination (i.e. sum for WCSP and product for MPE) of all the functions, whose scope includes X_i and is fully assigned along the path from root to node corresponding to $\langle X_i, x_i \rangle$, evaluated at all values along the path.

A *solution tree* T of $C_{\mathcal{T}}$ is a subtree such that: (1) it contains the root node of $C_{\mathcal{T}}$; (2) if an internal AND node n is in T, then all its children are in T; (3) if an internal OR node n is in T, then exactly one of its children is in T; (4) every tip node in T (i.e., a node with no children) is a terminal node. The cost of a solution tree is the product (resp. sum for WCSP) of the weights associated with its arcs.

Each node n in $C_{\mathcal{T}}$ is associated with a *value* $v(n)$ capturing the optimal solution cost of the conditioned subproblem rooted at n. Assuming a MPE/MAP problem, it was shown that $v(n)$ can be computed recursively based on the values of n's successors: OR nodes by maximization, AND nodes by multiplication. For WCSPs, $v(n)$ is updated for OR and AND nodes by minimization and summation, respectively [5].

We next provide an overview of the state-of-the-art best-first and depth-first branch and bound search schemes that explore the AND/OR search space for graphical models. As it

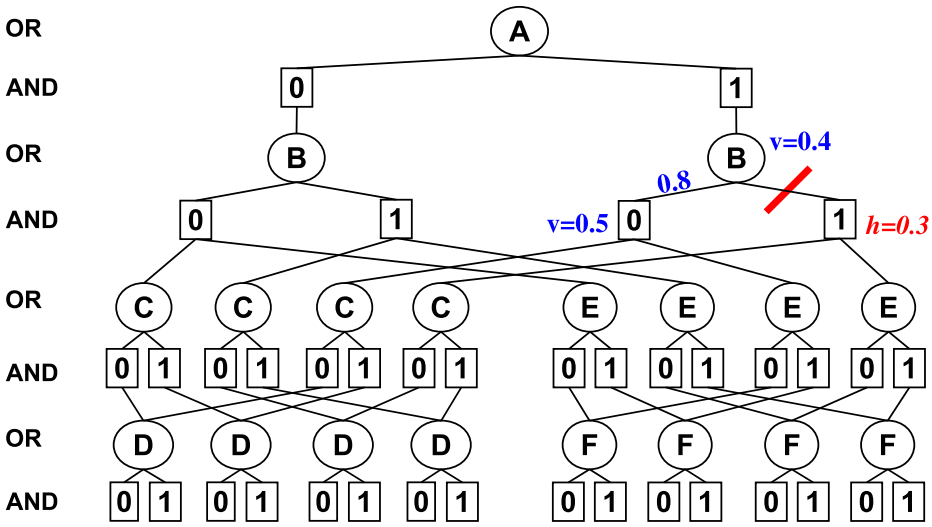


Fig. 2 Context-minimal AND/OR search graph with AOBB pruning example

is customary in the heuristic search literature, we assume without loss of generality a minimization task (i.e., min-sum optimization problem). Note that in the algorithm descriptions throughout the paper we assume the mini-bucket heuristic h_i, obtained with i-bound i, to be an input parameter to the search scheme. The heuristic is static and its computation is treated as a separate pre-processing step for clarity.

AND/OR best-first search (AOBF) The state-of-the-art version of A* for the AND/OR search space for graphical models is the AND/OR Best-First algorithm. AOBF is a variant of AO* [28] that explores the context-minimal AND/OR search graph. It was developed by [25].

AOBF, described by Algorithm 1 for min-sum tasks, maintains the explicated part of the context-minimal AND/OR search graph $\mathcal{G}$ and also keeps track of the current best partial solution tree T^*. AOBF interleaves iteratively a top-down node expansion step (lines 4-16), selecting a non-terminal tip node of T^* and generating its children in the explored search graph $\mathcal{G}$, with a bottom-up cost revision step (lines 17-25), updating the values of the internal nodes based on the children's values. If a newly generated child node is terminal, it is marked solved (line 9). During the bottom-up phase the OR nodes, whose best successor is solved, and the AND nodes, who have all children solved, are also marked as solved. The algorithm also marks the arc to the best AND child of an OR node through which the minimum is achieved (line 20). Following the backward step, a new best partial solution tree T^* is recomputed (line 28). AOBF terminates when the root node is marked solved. If the heuristic used is admissible, at the point of termination T^* is the optimal solution with cost $v(s)$, where s is the root node of the search space.

Note that, unlike the traditional best-first search schemes exploring OR search spaces, AOBF does not maintain explicit OPEN and CLOSED lists. Same is true for all AND/OR algorithms discussed in this paper.

Algorithm 1 AOBF(h_i, w_0) exploring the AND/OR search tree (Marinescu and Dechter (2009b))

Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$, weight w_0 (default value 1), pseudo tree $\mathcal{T}$ rooted at X_1, heuristic h_i calculated with i-bound i;
Output: Optimal solution to $\mathcal{M}$

1 create root OR node s labelled by X_1 and let $\mathcal{G}$ (explored search space) $= \{s\}$;
2 initialize $v(s) = w_0 \cdot h_i(s)$ and best partial solution tree T^* of $\mathcal{G}$, $T^* = \{s\}$;
3 **while** *s is not SOLVED* **do**
4 select non-terminal tip node n in T^*. If there is no such node then **exit**;
// expand node n
5 **if** $n = X_i$ *is OR* **then**
6 **forall the** $x_i \in D(X_i)$ **do**
7 create AND child $n' = \langle X_i, x_i \rangle$;
8 **if** *n' is TERMINAL* **then**
9 mark n' SOLVED;
10 $succ(n) \leftarrow succ(n) \cup n'$;
11 **else if** $n = \langle X_i, x_i \rangle$ *is AND* **then**
12 **forall the** *successor* X_j *of* X_i *in* $\mathcal{T}$ **do**
13 create OR child $n' = X_j$;
14 $succ(n) \leftarrow succ(n) \cup n'$;
15 initialize $v(n') = w_0 \cdot h_i(n')$ for all new nodes;
16 add new nodes to the explored search space space $\mathcal{G} \leftarrow \mathcal{G} \cup succ(n)$;
// update n and its AND and OR ancestors in $\mathcal{G}$, bottom-up
17 **repeat**
18 **if** *n is OR node* **then**
19 $v(n) = \min_{k \in succ(n)} (c(n,k) + v(k))$; // c(n,k) is the weight of the arc (n,k)
20 mark best successor k of OR node n, such that $k = \arg\min_{k \in succ(n)} (c(n,k) + v(k))$ (maintaining previously marked successor if still best);
21 mark n as SOLVED if its best marked successor is solved;
22 **else if** *n is AND node* **then**
23 $v(n) = \sum_{k \in succ(n)} v(k)$;
24 mark all arcs to the successors;
25 mark n as SOLVED if all its children are SOLVED;
26 $n \leftarrow p$; //p is a parent of n in $\mathcal{G}$
27 **until** *n is root node s*;
28 recompute T^* by following marked arcs from the root s;
29 **return** $\langle v(s), T^* \rangle$;

Theorem 2 [26]. *The AND/OR Best-First search algorithm with no caching (traversing an AND/OR search tree) has worst case time and space complexity of $O(n \cdot k^h)$, where n is the number of variables in the problem, h is the height of the pseudo tree and k bounds the domain size. AOBF with full caching traversing the context-minimal AND/OR graph has worst case time and space complexity of $O(n \cdot k^{w^*})$, where w^* is the induced width of the pseudo tree.*

AND/OR branch and bound (AOBB) The AND/OR Branch and Bound [25] algorithm traverses the context-minimal AND/OR graph in a *depth-first* rather than best-first manner, while keeping track of the current upper bound on the minimal solution cost. The algorithm (Algorithm 2, described here with no caching) interleaves forward node expansion (lines 4-18) with a backward cost revision (or propagation) step (lines 19-30) that updates node values (capturing the current best solution to the subproblem rooted at each node), until

search terminates and the optimal solution has been found. A node n will be pruned (lines 12-13) if the current upper bound is smaller or equal to the node's heuristic lower bound, computed recursively using the procedure described in Algorithm 3. Although branch and bound search is inherently anytime, AND/OR decomposition hinders the anytime performance of AOBB, which has to solve completely at each AND node almost all independent child subproblems (except for the last one), before obtaining any solution at all.

Algorithm 2 AOBB(h_i, w_0, $UB = \infty$) exploring the AND/OR search tree (Marinescu and Dechter (2009b))

Input: A graphical model $\mathscr{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$, weight w_0 (default value 1), pseudo tree $\mathscr{T}$ rooted at X_1, heuristic h_i calculated with i-bound i;
Output: Optimal solution to $\mathscr{M}$

```
1  create root OR node s labelled by X_1 and let stack of created but not expanded nodes OPEN = {s};
2  initialize v(s) = UB and best partial solution tree rooted at s is T*(s) = ∅ ;
3  while OPEN≠ ∅ do
4      select top node n on OPEN;
       //EXPAND
5      if n is OR node labeled X_i then
6          foreach x_i ∈ D(X_i) do
               //Expand node n:
7              add AND child n' = ⟨X_i, x_i⟩ to list of successors of n;
8              initialize v(n') = 0, best partial solution tree rooted in n' T*(n') = ∅;
9      if n is AND node labeled ⟨X_i, x_i⟩ then
10         foreach OR ancestor k of n do
11             recursively evaluate the cost of the partial solution tree rooted in k, based on heuristic
               h_i and weight w_0, assign its cost to f(k); // see evalPartialSolutionTree(T*_n,
               h_i(n), w_0) in Algorithm 3
12             if evaluated partial solution is not better than current upper bound at k (e.g.
               f(k) ≥ v(k) for minimization) then
13                 prune the subtree below the current tip node n;
14             else
15                 foreach successor X_j of X_i ∈ 𝒯 do
16                     add OR child n' = X_j to list of successors of n;
17                     initialize v(n') = ∞, best partial solution tree rooted in n' T*(n') = ∅;
18     add successors of n on top of OPEN;
       //PROPAGATE
       //only propagate if all children of n are evaluated and the final v's are determined
19     while list of successors of node n is empty do
20         if node n is the root node then
21             return solution: v(n), T*(n) ;
22         else
               //update ancestors of n, AND and OR nodes p, bottom up:
23             if p is AND node then
24                 v(p) = v(p) + v(n), T*(p) = T*(p) ∪ T*(n);
25             else if p is OR node then
26                 if the new value of better than the old one, e.g. v(p) > (c(p,n) + v(n)) for
                   minimization then
27                     v(p) = c(p,n) + v(n), T*(p) = T*(p) ∪ ⟨x_i, X_i⟩;
28                     if p is root then UB = min(UB, v(p))
29             remove n from the list of successors of p;
30             move one level up: n ← p;
31     return ⟨v(s), T*(s)⟩;
```

We use notation AOBB(h_i, w_0, UB) to indicate that AOBB uses the mini-bucket heuristic h_i obtained with i-bound=i, which is multiplied by the weight w_0 and initializes the upper bound used for pruning to UB. The default values of the weight and upper bound for AOBB traditionally are $w_0 = 1$, corresponding to regular unweighted heuristic, and $UB = \infty$, indicating that initially there is no pruning.

Algorithm 3 Recursive computation of the heuristic evaluation function (Marinescu and Dechter (2009b))

```
function evalPartialSolutionTree(T'_n, h(n), w)
Input: Partial solution subtree T'_n rooted at node n, heuristic function h(n);
Output: Heuristic evaluation function f(T'_n);
1 if succ(n) == ∅ then
2     return h(n) · w;
3 else
4     if n is an AND node then
5         let k_1, ..., k_l be the OR children of n;
6         return Σ_{i=1}^{l} evalPartialSolutionTree(T'_{k_i}, h(k_i), w);
7     else if n is an OR node then
8         let k be the AND child of n;
9         return c(n,k) + evalPartialSolutionTree(T'_k, h(k), w);
```

Theorem 3 [26]. *The AND/OR Branch and Bound search algorithm with no caching (traversing an AND/OR search tree) has worst case time and space complexities of $O(n \cdot k^h)$ and $O(n)$, respectively, where n is the number of variables in the problem, k bounds the domain size and h is the height of the pseudo tree. AOBB with full caching traversing the context-minimal AND/OR graph has worst case time and space complexity of $O(n \cdot k^{w^*})$, where w^* is the induced width of the pseudo tree.*

Although AOBF and AOBB with full caching have the same worst case space complexity, namely exponential in the induced width of the pseudo tree, often AOBB is far more memory efficient than AOBF because it can avoid caching nodes appearing on just a single path from the root. These nodes are called dead-caches. AOBB can avoid dead-caches, while AOBF cannot. In particular, when the induced width is high, most of the nodes are dead-caches and the search space is close to a tree. In these cases AOBB will rarely cache anything, while AOBF will still have to cache the whole search frontier. The most extreme case illustrating this point is when $w^* = n$ and we have a tree. AOBB will have no caching and will require no extra memory, while AOBF will have the memory overhead exponential in n. Furthermore, from an implementation perspective, the memory footprint of a cache entry (or node) recorded by AOBB is minimal (it is a single double value), whereas AOBF needs to record for every node a lot more information, such as lists of pointers to parents and children, heuristic and node values, in order to maintain the explicated search space in memory. All these issues carry over to the weighted heuristic best-first and depth-first search algorithms discussed in the paper.

Breadth rotating AND/OR branch and bound (BRAOBB) To remedy the relatively poor anytime behavior of AOBB, the *Breadth-Rotating AND/OR Branch and Bound* algorithm has been introduced recently by [29]. The basic idea is to rotate through different subproblems in a breadth-first manner. Empirically, BRAOBB finds the first sub-optimal solution significantly faster than plain AOBB [29].

3 Some properties of the weighted heuristic search

In this section we explore the interplay between the weight w and the heuristic function h, and their impact on the explored search space. It was observed early on that the search space explored by WA* when $w > 1$ is often smaller than the one explored by A*. Intuitively, the increased weight of the heuristic h transforms best-first search into a greedy search. Consequently, the number of nodes expanded tends to decrease as w increases, because a solution may be encountered early in the search process. In general, however, such behavior is not guaranteed [38]. For some domains greedy search can be less efficient than A*.

A search space is a directed graph having a root node. Its leaves are solution nodes or dead-ends. A greedy depth-first search always explores the subtree rooted at the current node representing a partial solution path. This leads us to the following definition.

Definition 2 (**focused search space**) An explored search space is *focused* along a path π, if for any node $n \in \pi$ once n is expanded, the only nodes expanded afterwards belong to the subtree rooted at n.

Having a focused explored search space is desirable because it would yield a fast and memory efficient search. In the following paragraphs we will show that there exists a weight w_h that guarantees a focused search for WA*, and that its value depends on the costs of the arcs on the solution paths and on the heuristic values along the path.

Proposition 1 *Let π be a solution path in a rooted search space. Let arc (n, n') be such that $f(n) > f(n')$. If n is expanded by A* guided by f, then a) any node n'' expanded after n and before n' satisfies that $f(n'') \leq f(n')$, b) n'' belongs to the subgraph rooted at n, and c) under the weaker condition that $f(n) \geq f(n')$, parts a) and b) still hold given that the algorithm breaks ties in favour of deeper nodes.*

Proof a) From the definition of best-first search, the nodes n'' are chosen from OPEN list (which after expansion of n includes all n's children, and in particular n'). Since n'' was chosen before n', it must be that $f(n'') \leq f(n')$.

b) Consider the OPEN list at the time when n is chosen for expansion. Clearly, any node q on OPEN satisfy that $f(q) \geq f(n)$. Since we assured $f(n) > f(n')$, it follows that $f(q) > f(n')$ and node q will not be expanded before n', and therefore any expanded node is in the subtree rooted at n.

c) Assume $f(n) \geq f(n')$. Consider any node q in OPEN: it either has an evaluation function $f(q) > f(n)$, and thus $f(q) > f(n')$, or $f(q) = f(n)$ and thus $f(q) \geq f(n')$. However, node q has smaller depth than n, otherwise it would have been expanded before n (as they have the same f value), and thus smaller depth than n', which is not expanded yet and thus is the descendant of n. Either way, node q will not be expanded before n'.

□

In the following, we assume that the algorithms we consider always break ties in favor of deeper nodes.

Definition 3 (f **non-increasing solution path**) Given a path $\pi = \{s, \ldots, n, n' \ldots, t\}$ and a heuristic evaluation function $h \leq h^*$, if $f(n) \geq f(n')$ for every n' (a child of n along π), then f is said to be *monotonically non-increasing* along π.

From Proposition 1 it immediately follows:

Theorem 4 *Given a solution path π along which evaluation function f is monotonically non-increasing, the search space is focused along path π.*

We will next show that this focused search property is achieved by WA* when w passes a certain threshold. We denote by $c(n, n')$ the cost of the arc from node n to its child n'.

Definition 4 (**the h-weight of an arc**) Restricting ourselves to problems where $h(n) - h(n') \neq 0$, we denote

$$w_h(n, n') = \frac{c(n, n')}{h(n) - h(n')}$$

Assumption 1 We will assume that for any arc (n, n'), $w_h(n, n') \geq 0$.

Assumption 1 is satisfied iff $c(n, n')$ and $h(n) - h(n')$ have the same sign, and if $h(n) - h(n') \neq 0$. Without loss of generality we will assume that for all (n, n'), $c(n, n') \geq 0$.

Definition 5 (**The h-weight of a path**) Consider a solution path π, s.t,

$$w_h(\pi) = \max_{(n,n')\in\pi} w_h(n, n') = \max_{(n,n')\in\pi} \frac{c(n, n')}{h(n) - h(n')} \tag{4}$$

Theorem 5 *Given a solution path π in a search graph and a heuristic function h such that $w_h(\pi)$ is well defined, then, if $w \geq w_h(\pi)$, WA* using w yields a focused search along π.*

Proof We will show that under the theorem's conditions f is monotonically non-increasing along π. Consider an arbitrary arc $(n, n') \in \pi$. Since $w \geq w_h(\pi)$, then

$$w \geq \frac{c(n, n')}{h(n) - h(n')}$$

or, equivalently,

$$c(n, n') \leq w \cdot h(n) - w \cdot h(n')$$

adding $g_\pi(n)$ to the both sides and some algebraic manipulations yield

$$g_\pi(n) + c(n, n') + w \cdot h(n') \leq g_\pi(n) + w \cdot h(n)$$

which is equivalent to

$$g_\pi(n') + w \cdot h(n') \leq g_\pi(n) + w \cdot h(n)$$

and therefore, for the weighted evaluation functions we have

$$f(n') \leq f(n).$$

Namely, f is monotonically non-increasing. From Theorem 1 it follows that WA* is focused along π with this w.

□

Clearly therefore,

Corollary 1 *If WA* uses $w \geq w_h(\pi)$ for each solution path π, then WA* performs a greedy search, assuming ties are broken in favour of deeper nodes.*

Corollary 2 *When $h = h^*$, then on an optimal path π, $\frac{c(n,n')}{h(n)-h(n')} = 1$. Therefore, any value $w \geq 1$ will yield a focused search relative to all optimal paths.*

Corollary 3 *If we found a solution path π, such that $w_h(\pi) = 1$, then the solution must be optimal and heuristic h is exact.*

Clearly, when h is exact, the weight $w = 1$ should be preferred since it guarantees the optimal solution. But if $w > 1$ and $h = h^*$, the solution found by the greedy search may not be optimal.

Example 1 Consider the graph in Fig. 3. Given $w = 10$, WA* will always find the incorrect path A-B-D instead of the exact solution path A-C-D.

Notice that, if the search is focused only along some solution paths, it can still be very unfocused relative to the entire solution space. More significantly, as we consider smaller weights, the search would be focused relative to a smaller set of paths, and therefore less contained. Yet with smaller weights, upon termination WA* yields a superior guarantee on the solution quality. We next provide an explicit condition showing that, under certain conditions, the weight on a path can provide a bound on the relative distance of the cost of the path from the optimal cost.

Theorem 6 *Given a search space and given an admissible heuristic function h having $w_h(\pi)$ function for each path π, let π be a solution path from s to t, satisfying:*

1) *for all arcs $(n, n') \in \pi$, $c(n, n') \geq 0$, and for at least one arc $c(n, n') > 0$*
2) *for all arcs $(n, n') \in \pi$, $h(n) - h(n') > 0$.*

Then the cost of the path C_π is within a factor of $w_h(\pi)$ from the optimal solution cost C^*. Namely,

$$C_\pi \leq w_h(\pi) \cdot C^* \tag{5}$$

Proof Denote by $f(n) = f_\pi(n)$ the weighted evaluation function of node n using weight $w = w_h(\pi)$: $f_\pi(n) = g(n) + w_h(\pi) \cdot h(n)$. Clearly, based on Theorem 2 we have that $\forall (n, n') \in \pi$, $f(n) \geq f(n')$. Namely, that search is focused relative to π.

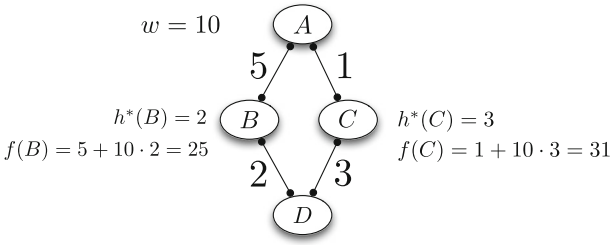


Fig. 3 WA* with the exact heuristic

Since for any arc (n, n') on path π, starting with s and ending with t, f is monotonically non-increasing when using $w_h(\pi)$, we have

$$f(s) \geq f(t)$$

Since $g(s) = 0$, $f(s) = w_h(\pi) \cdot h(s)$ and since $h(t) = 0$, $f(t) = g(t) = C_\pi$. We get that

$$w_h(\pi) \cdot h(s) \geq C_\pi$$

Since h is admissible, $h(s) \leq h^*(s)$ and $h^*(s) = C^*$, we have $h(s) \leq C^*$ and

$$w_h(\pi) \cdot h(s) \leq w_h(\pi) \cdot h^*(s) = w_h(\pi) \cdot C^*$$

We get from the above two inequalities that

$$w_h(\pi) \cdot C^* \geq C_\pi \qquad or \qquad \frac{C_\pi}{C^*} \leq w_h(\pi)$$

□

In practice, conditions (1) and (2) hold for many problems formulated over graphical models, in particular for many instances in our datasets (described in Section 5.1), when the MBE heuristic is used. However, in the presence of determinism the h-value is sometimes not well-defined, since for certain arcs $c(n, n') = 0$ and $h(n) - h'(n) = 0$.

In the extreme we can use $w_{max} = \max_\pi w_h(\pi)$ as the weight, which will yield a focused search relative to all paths, but we only guarantee that the accuracy factor will be bounded by w_{max} and in the worst case the bound may be loose.

It is interesting to note that:

Proposition 2 *If the heuristic function $h(n)$ is consistent and if for all arcs $(n, n') \in \pi$, $h(n) - h(n') > 0$ and $c(n, n') > 0$, then $w_h(\pi) \geq 1$.*

Proof From definition of consistency: $h(n) \leq c(n, n') + h(n')$. After some algebraic manipulation it is easy to obtain: $\max_{(n,n') \in E_\pi} \frac{c(n,n')}{h(n)-h(n')} \geq 1$ and thus $w_h(\pi) \geq 1$. □

We conclude that:

Proposition 3 *For every π, and under the conditions of Theorem 6*

$$C_\pi \geq C^* \geq \frac{C_\pi}{w_h(\pi)} \qquad and \quad therefore, \qquad \min_\pi \{C_\pi\} \geq C^* \geq \max_\pi \left\{ \frac{C_\pi}{w_h(\pi)} \right\}$$

In summary, the above analysis provides some intuition as to why weighted heuristic best-first search is likely to be more focused and therefore more time efficient for larger weights, and how it can provide a user-control parameter exploring the trade-off between time and accuracy. There is clearly room for exploration of the potential of Proposition 3 that we leave for future work.

4 Tailoring weighted heuristic BFS to graphical models

After analyzing a number of existing weighted heuristic search approaches we extended some of the ideas to the AND/OR search space over graphical models. In this section, we describe wAOBF and wR-AOBF - the two approaches that proved to be the most promising after our initial empirical evaluation (not reported here).

4.1 Weighted heuristic AOBF

The fixed-weighted heuristic version of the AOBF algorithm is obtained by multiplying the mini-bucket heuristic function by a weight $w > 1$ (i.e., substituting $h_i(n)$ by $w \cdot h_i(n)$, where $h_i(n)$ is the heuristic obtained by mini-bucket elimination with i-bound equal to i). This scheme is identical to WAO*, an algorithm introduced by [3], but it is adapted to the specifics of AOBF. Clearly, if $h_i(n)$ is admissible, which is the case for mini-bucket heuristics, the cost of the solution discovered by weighted heuristic AOBF is w-optimal, same as is known for WA* [31] and WAO* [3].

Consider an example problem with 4 binary variables in Fig. 4 that we will use to illustrate the work of our algorithms. Figure 4a shows the primal graph. We assume a Weighted CSP problem, namely the functions are not normalized and the task is the min-sum one: $C^* = \min_{A,B,C,D} \big(f(A,B)+f(B,C)+f(B)+f(A,D)\big)$. Figure 4c presents the AND/OR search graph of the problem, showing the heuristic functions and the weights derived from functions defined in Fig. 4b on the arcs. Since the problem is very easy, the MBE heuristic is exact.

Figure 5 shows the part of the search space explored by AOBF (on the left) and weighted heuristic AOBF with weight $w = 10$ (on the right). The numbers in grey boxes mark the order of nodes expansions. For clarity the entire sequence of node values' updates is not presented and only the latest assigned values v are shown. We use notation v_{SN} to indicate that the value was last updated during step N, namely when expanding the N^{th} node. The reported solution subtree is highlighted in bold. AOBF finds the exact solution with cost $C^* = 7$ and assignment $\mathbf{x}^* = \{A = 1, B = 0, C = 1, D = 1\}$. Weighted heuristic AOBF discovers a sub-optimal solution with cost $C = 12$ and assignment $\mathbf{x} = \{A = 0, B = 1, C = 1, D = 0\}$. In this example both algorithms expand 8 nodes.

Algorithm 4 wAOBF(w_0, h_i)

Input: A graphical model $\mathscr{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$; heuristic h_i calculated with i-bound i; initial weight w_0, weight update schedule S
Output: Set of sub-optimal solutions $\mathscr{C}$

```
Initialize w = w_0 and let C ← ∅;
while w >= 1 do
    ⟨C_w, T*_w⟩ ← AOBF(w · h_i);
    C ← C ∪ {⟨w, C_w, T*_w⟩};
    Decrease weight w according to schedule S;
return C;
```

4.2 Iterative weighted heuristic AOBF (wAOBF)

Since weighted heuristic AOBF yields w-optimal solutions, it can be extended to an anytime scheme **wAOBF** (Algorithm 4), by decreasing the weight from one iteration

to the next. This approach is identical to the Restarting Weighted A* by [32] applied to AOBF.

Algorithm 5 wR-AOBF(h_i, w_0)

Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$; pseudo tree $\mathcal{T}$ rooted at X_1; heuristic h_i for i-bound=i; initial weight w_0, weight update schedule S
Output: Set of sub-optimal solutions $\mathscr{C}$
1 initialize $w = w_0$ and let $\mathscr{C} \leftarrow \emptyset$;
2 create root OR node s labelled by X_1 and let $\mathscr{G} = \{s\}$;
3 initialize $v(s) = w \cdot h_i(s)$ and best partial solution tree $T^* = \{s\}$ of $\mathscr{G}$;
4 **while** $w >= 1$ **do**
5 expand and update nodes in $\mathscr{G}$ using AOBF(w,h_i) search with heuristic function $w \cdot h_i$;
6 if T^* has no more tip nodes then $\mathscr{C} \leftarrow \mathscr{C} \cup \{\langle w, v(s), T^* \rangle\}$;
7 decrease weight w according to schedule S;
8 for all leaf nodes in $n \in \mathscr{G}$, update $v(n) = w \cdot h_i(n)$. Update the values of all nodes in $\mathscr{G}$ using the values of their successors. Mark best successor of each OR node.;
9 recalculate T^* following the marked arcs;
10 **return** $\mathscr{C}$;

Theorem 7 *The worst case time and space complexity of a single iteration of wAOBF with full caching is $O(n \cdot k^{w^*})$, where n is the number of variables, k is the largest domain size and w^* is the induced width of the pseudo tree.*

Proof During each iteration wAOBF executes AOBF from scratch with no additional overhead. The number of required iterations depends on the start weight value and the weight decreasing policy. □

We note that certain weight decreasing policies (or schedules) do not guarantee that the weight converges to 1 in a finite number of steps (see for example the $sqrt(k)$ policy in Section 5.3). In practice, for these kinds of policies, if the current weight is less than 1.0001, then it is set to 1.0, thus enabling a termination condition for the respective algorithm.

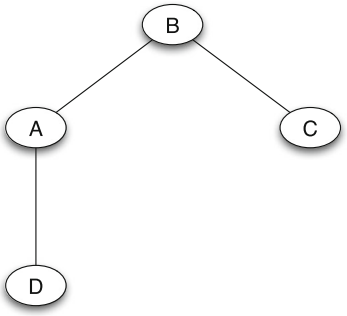
4.3 Anytime repairing AOBF (wR-AOBF)

Running each search iteration from scratch seems redundant, so we introduce Anytime Repairing AOBF (Algorithm 5), which we call **wR-AOBF**. It is an extension of the *Anytime Repairing A* (ARA*)* algorithm [22] to the AND/OR search spaces over graphical models. The original ARA* algorithm utilizes the results of previous iterations by recomputing the evaluation functions of the nodes with each weight change, and thus re-using the inherited OPEN and CLOSED lists. The algorithm also keeps track of the previously expanded nodes whose evaluation function changed between iterations and re-inserts them back to OPEN before starting each iteration.

Extending this idea to the AND/OR search space is fairly straightforward. Since AOBF does not maintain explicit OPEN and CLOSED lists, wR-AOBF keeps track of the partially explored AND/OR search graph, and after each weight update it performs a bottom-up update of all the node values starting from the leaf nodes (whose h-values are multiplied by the new weight) and continuing towards the root node (line 8). During this phase, the algorithm also marks the best AND successor of each OR node in the search graph. These markings are used to recompute the best partial solution tree T^*. Then, the search resumes in the usual manner by expanding a tip node of T^* (line 5).

Like ARA*, wR-AOBF is guaranteed to terminate with a solution cost C such that $C \leq w \cdot C^*$, where C^* is the optimal solutions cost.

Theorem 8 *The worst case space complexity of a single iteration of wR-AOBF with caching is $O(n \cdot k^{w^*})$, where n is the number of variables, k is the largest domain size and w^* is the induced width of the pseudo tree. The worst case time complexity can be loosely bounded by $O(2 \cdot n \cdot k^{w^*})$. The total number of iteration depends on the starting weight and the parameters of the weight updating policy/schedule.*



(a) Primal graph.

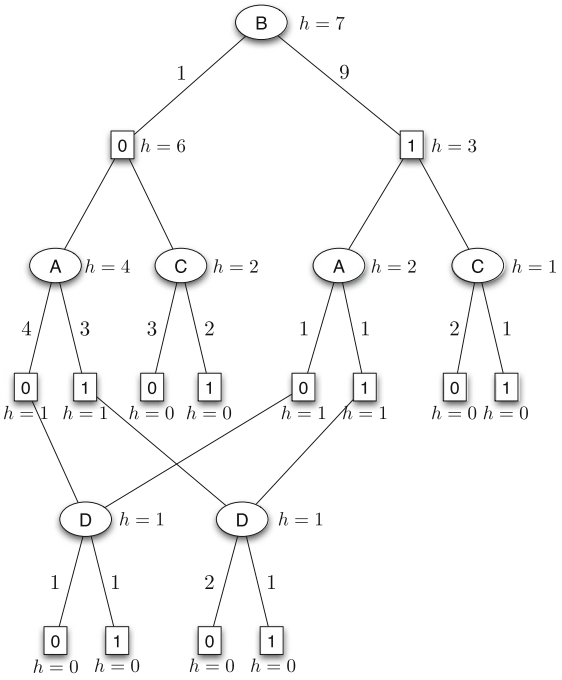
$f(A,B)$	A	B
4	0	0
1	0	1
3	1	0
1	1	1

$f(B,C)$	B	C
3	0	0
2	0	1
2	1	0
1	1	1

$f(A,D)$	A	D
1	0	0
1	0	1
3	1	0
1	1	1

$f(B)$	B
1	0
9	1

(b) Functions.



(c) Context-minimal weighted AND/OR graph.

Fig. 4 Example problem with four variables, the functions defined over pairs of variables and resulting AND/OR search graph

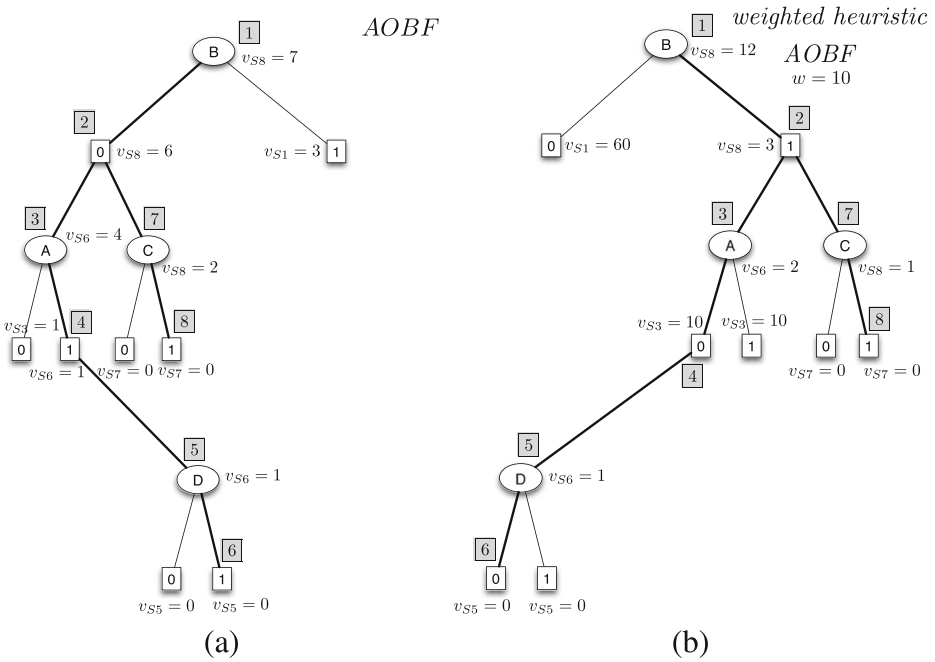


Fig. 5 Search graphs explored by **a** AOBF, **b** Weighted heuristic AOBF, w=10. The numbers in *grey* boxes indicate the order in which the nodes are expanded. v_{SN} indicates that value v was last assigned during step N, i.e. while expanding the N^{th} node

Proof In the worst case at each iteration wR-AOBF explores the entire search space of size $O(n \cdot k^{w^*})$, having the same theoretical space complexity as AOBF and wAOBF. In practice there exist an additional space overhead due to the book-keeping required by best-first search to maintain in memory the explicated AND/OR search graph (see also Section 2.3).

The time complexity comprises of the time it takes to expand nodes and the time overhead due to updating node values during each iteration. Every time the weight is decreased, wR-AOBF needs to update the values of all nodes in the partially explored AND/OR graph, at most $O(n \cdot k^{w^*})$ of them. □

The updating of node values is a costly step because it involves all nodes the algorithm has ever generated. Thus, although wR-AOBF expands less nodes that wAOBF, in practice it is considerably slower, as we will see next in the experimental section.

5 Empirical evaluation of weighted heuristic BFS

Our empirical evaluation of weighted heuristic search schemes consists of two parts. In this section, we focus on the two weighted heuristic best-first algorithms described in Section 4: wAOBF and wR-AOBF, respectively. In Section 6.2 we additionally compare these algorithms with the two weighted heuristic depth-first branch and bound schemes that will be introduced in Section 6.1.

5.1 Overview and methodology

In this section, we evaluate the behavior of wAOBF and wR-AOBF and contrast their performance with a number of previously developed algorithms. The main point of reference for our comparison is the depth-first branch and bound scheme, BRAOBB [29], which is known to be one of the most efficient anytime algorithms for graphical models.[3,4] In a subset of experiments we also compare against A* search [12] and depth-first branch and bound (DFBB) [20], both exploring an OR search tree, and against Stochastic Local Search (SLS) [13, 17]. We implemented our algorithms in C++ and ran all experiments on a 2.67GHz Intel Xeon X5650, running Linux, with 4 GB allocated for each job.

All schemes traverse the same context-minimal AND/OR search graph, defined by a common static variable ordering, obtained using the well-known *min-fill* ordering heuristic [19]. The algorithms return solutions at different time points until either the optimal solution is found, until a time limit of 1 hour is reached or until the scheme runs out of memory. No evidence was used. A* and DFBB are using a chain (following the min-fill ordering) instead of a pseudo tree like the AND/OR algorithms do. All branch and bound algorithms use a dynamic value ordering heuristic that sorts the domain values of a variable in increasing order of their corresponding heuristic upper bounding estimates.

All schemes use the Mini-Bucket Elimination heuristic, described in Section 2.3, with ten i-bounds, ranging from 2 to 20. However, for some hard problems computing the mini-bucket heuristic with the larger i-bounds proved infeasible, so the actual range of i-bounds varies among the benchmarks and among instances within a benchmark.

We evaluated the algorithms on four benchmarks: Weighted CSPs (WCSP), Binary grids, Pedigree networks and Type4 genetic networks. The **pedigree instances ("pedigree*")** arise from the domain of genetic linkage analysis and are associated with the task of haplotyping. The haplotype is the sequence of alleles at different loci inherited by an individual from one parent, and the two haplotypes (maternal and paternal) of an individual constitute this individual's genotype. When genotypes are measured by standard procedures, the result is a list of unordered pairs of alleles, one pair for each locus. The maximum likelihood haplotype problem consists of finding a joint haplotype configuration for all members of the pedigree which maximizes the probability of data. It can be shown that given the pedigree data the haplotyping problem is equivalent to computing the most probable explanation of a Bayesian network that represents the pedigree [8]. In **binary grid networks ("50-*", "75-*"** and **"90-*")**[5] the nodes corresponding to binary variables are arranged in an N by N square and the functions are defined over pairs of variables and are generated uniformly randomly. The **WCSP ("*.wcsp")** benchmark includes random binary WCSPs, scheduling problems from the SPOT5 benchmark, and radio link frequency assignment problems, providing a large variety of problem parameters. **Type4 ("type4b_*")** instances come from the domain of genetic linkage analysis, just as the Pedigree problems, but are known to be significantly harder. Table 1 describes the benchmark parameters.

For each anytime solution found by an algorithm we record its cost, CPU time in seconds and the corresponding weight (for weighted heuristic schemes) at termination. For uniformity, we consider all problems as solving the max-product task, also known as Most

[3] http://www.cs.huji.ac.il/project/PASCAL/realBoard.php

[4] http://www.hlt.utdallas.edu/~vgogate/uai14-competition/leaders.html

[5] http://graphmod.ics.uci.edu/repos/mpe/grids/

Table 1 Benchmark parameters: # inst - number of instances, n - number of variables, # func - number of functions (include uniary ones), k - domain size, w^* - induced width, h_T - pseudo tree height

Benchmark	# inst	n	# func	k	w^*	h_T
Pedigrees	11	437–1152	437–1152	3–7	16–39	52–104
Grids	32	144–2500	144–2500	2–2	15–90	48–283
WCSP	56	25–1057	301–18017	2–100	5–287	11–337
Type4	10	3907–8186	3907–8186	5–5	21–32	319–625

Probable Explanation problem (MPE). The computed anytime costs returned by the algorithms are lower bounds on the optimal solutions. In our experiments we evaluate the impact of the weight on the solution accuracy and runtime, analyze the choice of weight decreasing policy, and study the anytime behavior of the schemes and the interaction between heuristic strength and the weight.

5.2 The impact of weights on the weighted heuristic AOBF performance

One of the most valuable qualities of weighted heuristic search is the ability to flexibly control the trade-off between speed and accuracy of the search using the weight, which provides a w-optimality bound on the solution.

In order to evaluate the impact of the weight on the solution accuracy and runtime we run wAOBF and we consider iterations individually. Each iteration j is equivalent to a single run of AOBF(w_j, h_i), namely the AND/OR Best-First algorithm that uses MBE heuristic h_i, obtained with an i-bound equal to i, multiplied by weight w_j.

Table 2 reports the results for selected weights (w=2.828, 1.033, 1.00), for several selected instances representative of the behavior prevalent over each benchmark. Following the names and parameters of each instance, the table is vertically split into two blocks, corresponding to two i-bounds. In the first column of each block we report the time (in seconds) it took BRAOBB to find the optimal solution to the problem (the higher entry in each row) and the solution cost on a logarithmic scale (the lower entry in each row). The symbol "—" indicates that the corresponding algorithm ran out of memory. The next three columns show the runtime in seconds and the cost on the log scale obtained by AOBF when using a specific weight value. The entries mentioned in this section are highlighted. Note that, since calculation of the mini-bucket heuristics is time and space exponential in i-bound, for some instances the heuristics can't be obtained for large i-bounds (e.g. 1502.wcsp, i-bound=10).

Comparison between the exact results by AOBF obtained with weight $w = 1$ (columns 5 and 9) and by BRAOBB (columns 2 and 6) with any one of the other columns reveals that abandoning optimality yields runtime savings and allows to find approximate solutions when exact ones cannot be obtained within an hour.

In more details, let us consider, for example, the columns of Table 2 where the costs generated are guaranteed to be a factor of 2.828 away from the optimal. We see orders of magnitude time savings compared to BRAOBB, for both i-bounds. For example, for pedigree9, i-bound=16, for $w = 2.828$ the runtime of the weighted heuristic AOBF is merely 6.24 seconds, while BRAOBB's is 1082.02 seconds. For WCSP networks, the algorithms' runtimes are often quite similar. For example, for bwt3ac.wcsp, i-bound=10, BRAOBB takes 54.34 seconds and weighted heuristic AOBF for $w = 2.828$ takes 54.88. On some

Table 2 Runtime (sec) and cost (on logarithmic scale) obtained by AOBF(w, h_i) for selected w, and by BRAOBB (that finds C^* - optimal cost)

Instance (n, k, w^*, h_T)	BRAOBB	**AOBF(w, h_i) weights**			BRAOBB	**AOBF(w, h_i) weights**		
		2.828	1.033	1.00		2.828	1.033	1.00
	time C^*	time log(cost)	time log(cost)	time log(cost)	time C^*	time log(cost)	time log(cost)	time log(cost)
Grids	I-bound=6				I-bound=20			
50-16-5 (256, 2, 21, 79)	2601.46 -16.916	0.16 -21.095	—	—	7.16 -16.916	7.01 -17.57	7.01 -16.916	7.02 -16.916
50-17-5 (289, 2, 23, 77)	1335.44 -17.759	0.05 -23.496	—	—	9.42 -17.759	9.44 -17.829	9.44 -17.759	9.44 -17.759
75-18-5 (324, 2, 24, 85)	390.72 -8.911	0.42 -10.931	74.69 -8.911	88.53 -8.911	13.52 -8.911	13.95 -9.078	13.95 -8.911	13.96 -8.911
75-20-5 (400, 2, 27, 99)	time out	1.78 -16.282	—	—	22.52 -12.72	19.35 -14.067	24.96 -12.72	27.85 -12.72
90-21-5 (441, 2, 28, 106)	187.75 -7.658	1.13 -8.871	41.38 -7.658	42.48 -7.658	17.01 -7.658	17.32 -9.476	17.65 -7.658	17.74 -7.658
Pedigrees	I-bound=6				I-bound=16			
pedigree9 (935, 7, 27, 100)	time out	0.83 -137.178	—	—	1082.02 -122.904	6.24 -133.063	34.66 -122.904	—
pedigree13 (888, 3, 32, 102)	time out	0.18 -88.563	—	—	time out	4.13 -76.429	—	—
pedigree37 (726, 5, 20, 72)	4.36 -144.882	0.08 -163.325	4.42 -145.082	9.99 -144.882	388.36 -144.882	388.96 -155.259	389.02 -145.341	389.07 -144.882
pedigree39 (953, 5, 20, 77)	time out	0.11 -174.304	—	—	4.34 -155.608	4.3 -162.381	4.37 -155.608	4.83 -155.608
WCSP	I-bound=2				I-bound=10			
1502.wcsp (209, 4, 5, 11)	time out	0.0 -1.258	0.01 -1.258	0.0 -1.258	—	—	—	—
42.wcsp (190, 4, 26, 72)	time out	—	—	—	1563.44 -2.357	11.69 -2.418	—	—
bwt3ac.wcsp (45, 11, 16, 27)	2.47 -0.561	0.93 -0.561	1.84 -0.561	1.85 -0.561	54.34 -0.561	54.88 -0.561	54.92 -0.561	54.92 -0.561
capmo5.wcsp (200, 100, 100, 100)	time out	1.18 -0.262	—	—	time out	24.04 -0.262	—	—
myciel5g_3.wcsp (47, 3, 19, 24)	2661.91 -64.0	—	—	—	12.93 -64.0	2.5 -72.0	—	—
Type4	I-bound=6				I-bound=16			
type4b_100_19 (3938, 5, 29, 354)	time out	5.02 -1309.91	—	—	time out	33.32 -1171.002	—	—
type4b_120_17 (4072, 5, 24, 319)	time out	4.16 -1483.588	—	—	time out	26.06 -1362.607	104.37 -1327.776	—
type4b_140_19 (5348, 5, 30, 366)	time out	7.28 -1765.403	—	—	time out	44.94 -1541.883	—	—
type4b_150_14 (5804, 5, 32, 522)	time out	16.15 -2007.388	—	—	time out	38.22 -1727.035	—	—
type4b_170_23 (5590, 5, 21, 427)	time out	9.65 -2191.859	—	—	time out	18.62 -1978.588	38.4 -1925.883	—

Instance parameters: n - number of variables, k - max domain size, w^* - induced width, h_T - pseudo tree height. "—" - running out of memory. 4 GB memory limit, 1 hour time limit, MBE heuristic. The entries mentioned in the text are highlighted

WCSP instances, such as myciel5g_3.wcsp, i-bound=2, BRAOBB is clearly superior, finding an optimal solution within the time limit, while weighted heuristic AOBF runs out of memory and does not report any solution for $w = 2.828$.

Comparing columns 5 and 9, exhibiting full AOBF with $w = 1$ (when it did not run out of memory) against $w = 2.828$ we see similar behavior. For example, for grid 75-18-5, i-bound=6, we see that the exact AOBF ($w = 1$) requires 88.53 seconds, which is about 200 times longer than with weight $w = 2.828$ which requires 0.42 seconds.

More remarkable results can be noticed when considering the column of weight $w = 1.033$, especially for the higher i-bound (stronger heuristics). These costs are just a factor of 1.033 away from optimal, yet the time savings compared with BRAOBB are impressive. For example, for pedigree9, i-bound=16 weighted heuristic AOBF runtime for $w = 1.033$ is 34.66 seconds as opposed to 1082.02 seconds by BRAOBB. Observe that the actual results are often far more accurate than what the bound suggests. In particular, on several test cases, the optimal solution is obtained with $w > 1$. For example, see grid 75-18-5, i-bound=20, $w = 1.003$. Sometimes the exact AOBF with $w = 1$ is faster than BRAOBB.

Impact of heuristic strength The i-bound parameter allows to flexibly control the strength of the mini-bucket heuristics. Clearly, more accurate heuristics yield better results for any heuristic search and thus should be preferred. However, running the mini-buckets with sufficiently high i-bound is not always feasible due to space limitations and has a considerable time overhead, since the complexity of the Mini-Bucket Elimination algorithm is exponential in the i-bound. Thus, we are interested to understand how the heuristic strength influences the behavior of weighted heuristic best-first schemes when the value of the i-bound is considerably smaller than the induced width of the problem.

Comparing the results Table 2 across i-bounds for the same algorithm and the same weight, we observe a number of instances where a more accurate heuristic comes at too high a price. For example, for pedigree37 weighted heuristic AOBF finds a w-optimal solution with $w = 2.828$ in 0.08 seconds for i-bound=6, but takes 388.96 seconds for i-bound=16. One of the examples to the contrary, where the higher i-bound is beneficial, is grid 90-21-5, where weighted heuristic AOBF takes 41.38 seconds to terminate for $w = 1.033$ when i-bound=6, but only 17.65 seconds, when the i-bound=20.

Table 2 shows that weighted heuristic AOBF is less sensitive to the weak heuristics compared with BRAOBB. For example, for grid 90-21-5 and for i-bound=20, BRAOBB terminates in 17.01 seconds. However, if the heuristic is weak (i-bound=6), it requires 187.75 seconds, an order of magnitude more. On the other hand, for the same instance weighted heuristic AOBF with weight $w = 1.033$ has a much smaller difference in performance for the two i-bounds. Weighted heuristic AOBF terminates in 17.65 seconds for i-bound=20 and in 41.38 seconds for i-bound=6. This may suggest that wAOBF could be preferable when the i-bound is small relative to the problem's induced width.

Overall, weighted heuristic AOBF solves some hard problems that are infeasible for the exact scheme and often yields solutions with relatively tight bounds considerably faster than the optimal solutions obtained by BRAOBB or exact AOBF.

5.3 Exploring weight policies

How should we choose the starting weight value and weight decreasing policy? Previous works on weighted heuristic search usually avoid disclosing the details of how the starting weight is defined and how it is decreased at each iteration [e.g., [11, 22], etc.]. To answer this we evaluated 5 different policies.

The first two policies we considered were *substract*, which decreases the weight by a fixed quantity, and *divide*, which at each iteration divides the current weight by a constant. These policies lay on the opposite ends of the strategies spectrum. The first method changes the weight very gradually and consistently, leading to a slow improvement of the solution. The second approach yields less smooth anytime behavior, since the weight rapidly approaches 1.0 and fewer intermediate solutions are found. This could potentially allow the schemes to produce the exact solution faster, but on hard instances presents a danger of leaping directly to a prohibitively small weight and thus failing prematurely due to memory issues. The other policies we considered were constructed manually based on the intuition that it is desirable to improve the solution rapidly by decreasing the weight fast initially and then "fine-tune" the solution as much as the memory limit allows, by decreasing the weight slowly as it approaches 1.0.

Overall, we evaluated the following five policies, each for several values of parameters. We denote by w_j the weight used at the j^{th} iteration of the algorithm, the k and d denote real-valued policy parameters, where appropriate. Given k and d, assuming $w_1 = w_0$ (start weight), for $j > 1$:

- $substract(k)$: $w_j = w_{j-1} - k$
- $divide(k)$: $w_j = w_{j-1}/k$
- $inverse$: $w_j = w_1/j$
- $piecewise(k, d)$: if $w_j \geq d$ then $w_j = w_1/j$ else $w_j = w_{j-1}/k$
- $sqrt(k)$: $w_j = \sqrt{w_{j-1}}/k$

The initial weight value needs to be large enough 1) to explore the schemes' behavior on a large range of weights; 2) to make the search focused enough initially to solve harder instances, known to be infeasible for regular best-first search within the memory limit. After some preliminary experiments (not included) we chose the starting weight w_0 to be equal to 64. We noticed that further increase of w_0 typically did not yield better results. Namely, the instances that were infeasible for wAOBF and wR-AOBF with $w = 64$ also did not fit in memory when weight was larger. Such behavior can be explained by many nodes having evaluation function values so similar, that even a very large weight did not yield much difference between them, resulting in a memory-prohibitive search frontier.

Figure 6 illustrates the weight changes during the first 50 iterations according to the

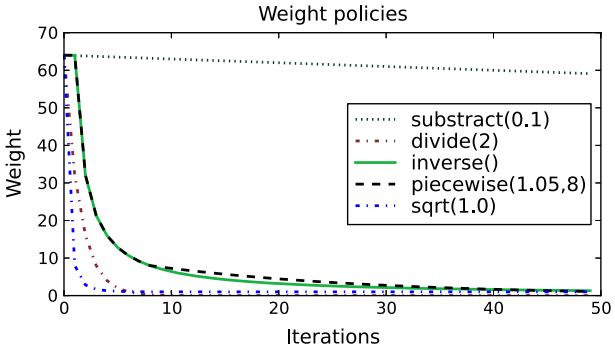


Fig. 6 The dependency of the weight value on iteration index according to considered weight policies, showing first 50 iterations, starting weight w_0= 64

considered policies. We use the parameter values that proved to be most effective in the preliminary evaluation: $substract(k = 0.1)$, $divide(k = 2)$, $inverse()$, $piecewise(k = 1.05, d = 8)$, $sqrt(k = 1.0)$.

Figures 7 and 8 show the anytime performance of wAOBF and wR-AOBF with various weight scheduling schemes, namely how the solution cost changes as a function of time in seconds. We plot the solution cost on a logarithmic scale. Figure 7 displays the results for wAOBF using each of our 5 weight policy. We display results for a mid-range i-bound, on two instances from each of the benchmarks: Grids, Pedigrees, WCSPs and Type4. Figure 8 shows analogous results for wR-AOBF, on the same instances.

Comparing the anytime performance of the two schemes, we consider as better the one that finds the initial solutions faster and whose solutions are more accurate (i.e., have higher costs). Graphically, the curves closer to the left top corner of the plot are better.

Several values of numerical parameters for each policies were tried, but only the ones that yielded the best performance are presented. The starting weight is 64, $w!$ denotes the weight at the time of algorithms termination. In this set of experiments the memory limit was 2 GB, while the time limit was set to 1 hour. The behavior depicted here was quite typical across instances and i-bounds.

We observe in Fig. 7 that for most Pedigrees, Grids and Type4 problems wAOBF finds the initial solution the fastest using the $sqrt$ policy (the reader is advised to consult the colored plots online). This can be seen, for example, on grid instance 75-23-5 and on type4b_120_17, respectively. The $sqrt$ policy typically facilitates the fastest improvement of the initial solutions. For most of the WCSP instances, however, there is no clear dominance between the weight policies. On some instances (not shown) the $sqrt$ policy is again superior. On others, such as instance 505, the difference is negligible.

Figure 8 depicts the same information for wR-AOBF. The variance between the results yielded by different weight policies is often very small. On many instances, such as pedigree31 or WCSP 505, it is almost impossible to tell which policy is superior. The dominance of $sqrt$ policy is less obvious for wR-AOBF, than is was for wAOBF. On a number of problems $piecewise$ and $inverse$ policies are superior, often yielding almost identical results, see for example, pedigree7 or WCSP 404. However, there are still many instances, for which $sqrt$ policy performs well, for example, pedigree7.

Overall, we chose to use the $sqrt$ weight policy in our subsequent experiments, as it is superior on more instances for wAOBF than other policies, and is often either best or close second best for wR-AOBF.

5.4 Anytime behavior of weighted heuristic best-first search

We now turn to our main focus of evaluating the anytime performance of our two iterative weighted heuristic best-first schemes wAOBF and wR-AOBF and comparing against BRAOBB, A*, depth-first branch and bound on the OR search tree (DFBB) and Stochastic Local Search (SLS). We ran each scheme on all instances from the same four benchmarks using the MBE heuristic with i-bounds ranging from 2 to 20, respectively. We recorded the solutions at different time points, up until either the optimal solution was found or until the algorithm ran out of 4 GB of memory or the time cut off of 3600 seconds (1 hour) was reached. When comparing two anytime algorithms, we consider one to be superior to another if: 1) it discovers the initial solution faster and also 2) for a fixed time it returns a more accurate solution. We rarely encounter conflict on these two measures.

We first illustrate the results using a selected set of individual instances in Table 3 and the bar charts in Figs. 9 and 10, respectively. Then, we provide summaries over all

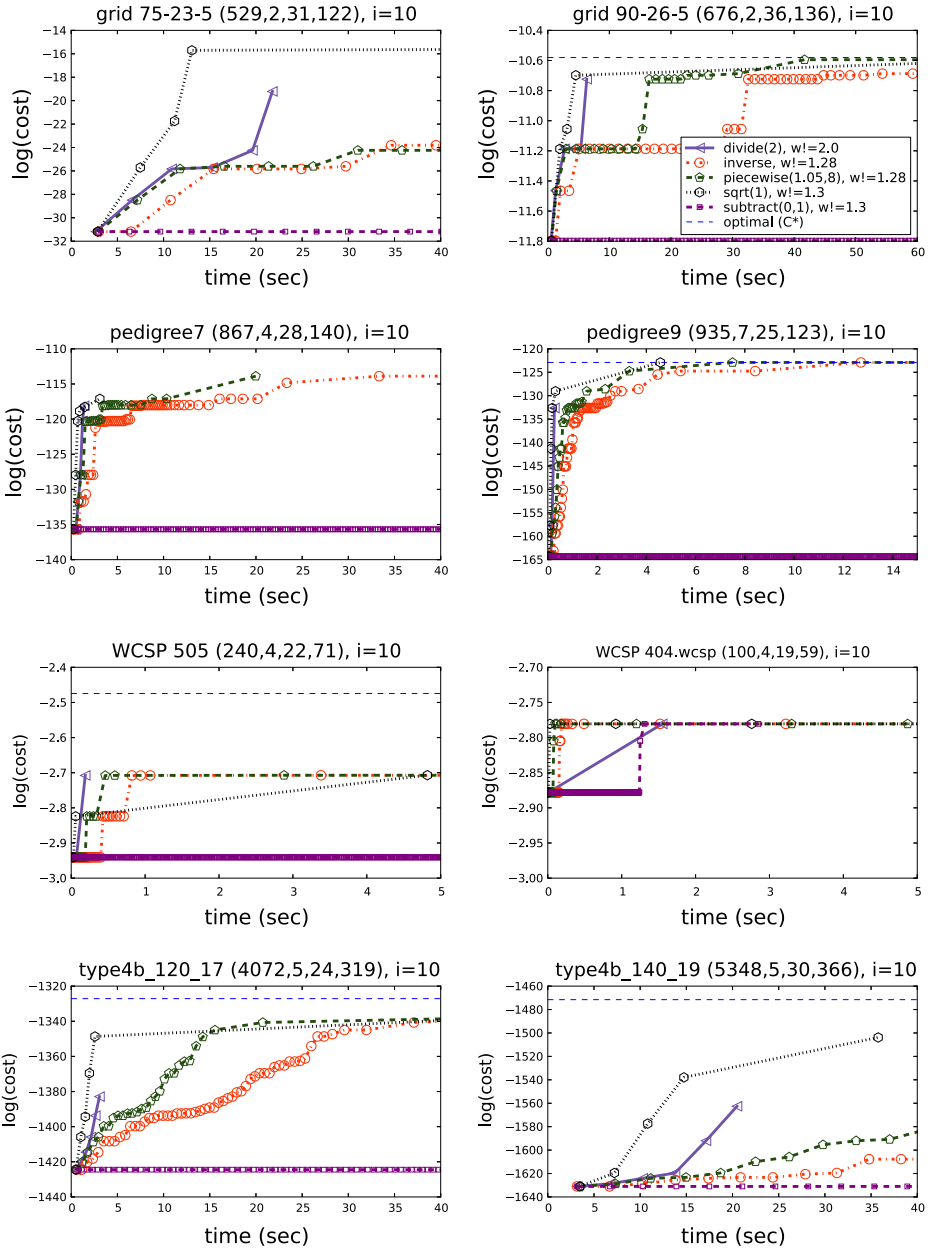


Fig. 7 wAOBF: solution cost (on logarithmic scale) vs time (sec) for different weight policies, starting weight = 64. Instance parameters are in format (n, k, w^*, h_T), where n - number of variables, k - max. domain size, w^* - induced width, h_T - pseudo tree height. Time limit - 1 hour, memory limit - 2 GB, MBE heuristic

instances using the bar charts in Figs. 11, 12, 13 and 14 and the scatter plots in Fig. 15, respectively.

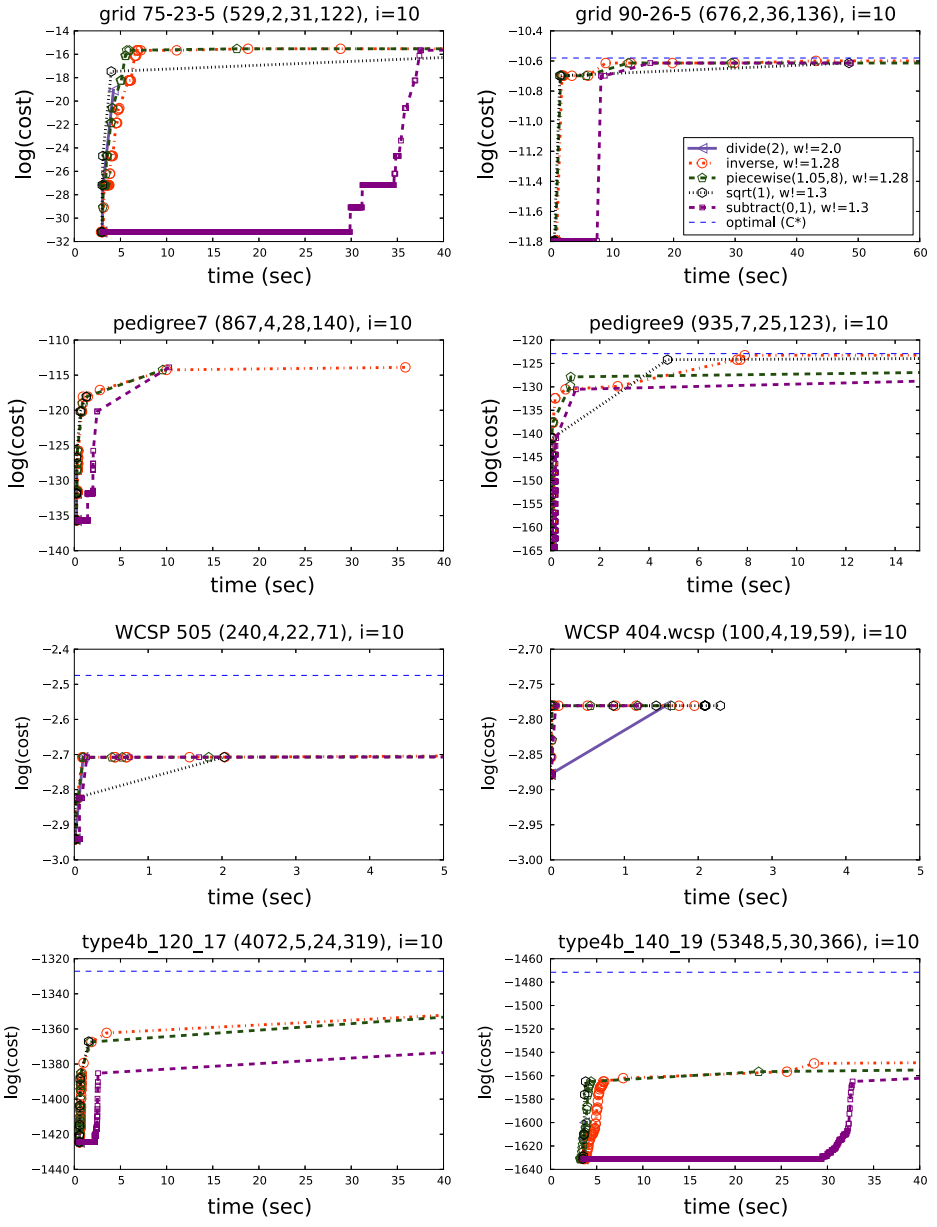


Fig. 8 wR-AOBF: solution cost (on logarithmic scale) vs time (sec) for different weight policies, starting weight = 64. Instance parameters are in format (n, k, w^*, h_T), where n - number of variables, k - max. domain size, w^* - induced width, h_T - pseudo tree height. Time limit - 1 hour, memory limit - 2 GB, MBE heuristic

5.4.1 wAOBF vs wR-AOBF

Table 3 shows solution cost and corresponding weight for wAOBF and wR-AOBF on two selected instances from each benchmark, for medium i-bound, and for several time

Table 3 Solution cost (on logarithmic scale) and corresponding weight for a fixed time bound for wAOBF and wR-AOBF

Instance	Algorithm	Time bounds				
		10	30	60	600	3600
		log(cost) weight	log(cost) weight	log(cost) weight	log(cost) weight	log(cost) weight
Grids, i-bound=10						
75-16-5 (256,2,21,73)	wAOBF	-8.1262 1.0671	-8.0642 1.0082	-8.0642 1.0	-8.0642 1.0	-8.0642 1.0
	wR-AOBF	8.0642 1.0	-8.0642 1.0	-8.0642 1.0	-8.0642 1.0	-8.0642 1.0
75-26-5 (676,2,36,129)	wAOBF	-24.5951 2.8284	-23.0522 1.6818	-23.0522 1.6818	-23.0522 1.6818	-23.0522 1.6818
	wR-AOBF	-25.2884 1.6818	-25.2884 1.6818	-25.2884 1.6818	-25.2884 1.6818	-25.2884 1.6818
Pedigrees, i-bound=10						
pedigree7 (867,4,32,90)	wAOBF	-114.4256 1.2968	-114.4256 1.2968	-113.8887 1.1388	-113.8887 1.1388	-113.8887 **1**.1388
	wR-AOBF	-118.8305 1.2968	-118.8305 1.2968	-114.5481 1.1388	-114.5481 1.1388	-114.5481 1.1388
pedigree41 (885,5,33,100)	wAOBF	-123.6391 1.2968	-121.3366 1.1388	-121.3366 1.1388	-121.3366 1.1388	-121.3366 1.1388
	wR-AOBF	-124.656 1.2968	-121.3366 1.1388	-121.3366 1.1388	-121.3366 1.1388	-121.3366 1.1388
WCSPs, i-bound=10						
408.wcsp (200,4,34,87)	wAOBF	-2.6798 1.2968	-2.6798 1.2968	-2.6798 1.2968	-2.6798 1.2968	-2.6798 1.2968
	wR-AOBF	-2.6811 1.2968	-2.6811 1.2968	-2.6811 1.2968	-2.6811 1.2968	-2.6811 1.2968
capmo5.wcsp (200,100,100,100)	wAOBF	—	-0.2622 2.8284	-0.2622 2.8284	-0.2622 2.8284	-0.2622 2.8284
	wR-AOBF	—	-0.2622 2.8284	-0.2622 2.8284	-0.2622 2.8284	-0.2622 2.8284
Type4, i-bound=10						
type4b_150_14 (5804,5,32,522)	wAOBF	—	-1698.1897 1.6818	-1652.7112 1.2968	-1652.7112 1.2968	-1652.7112 1.2968
	wR-AOBF	—	-1763.7714 1.2968	-1763.7714 1.2968	-1763.7714 1.2968	-1763.7714 1.2968
type4b_190_20 (8186,5,29,625)	wAOBF	—	—	-2605.3849 1.2968	-2605.3849 1.2968	-2605.3849 1.2968
	wR-AOBF	—	—	-2803.4548 1.2968	-2603.6145 1.1388	-2603.6145 1.1388

"—" denotes no solution found by the time bound. 4 GB memory, 1 hour time limit, MBE heuristic. The entries mentioned in the text are highlighted

bounds. We observe that for these instances (that are quite representative), the simpler scheme wAOBF provides more accurate solutions than wR-AOBF (e.g., pedigree7, for 10-30 seconds). Still, on some instances the solution costs are equal for the same time bound (e.g., capmo05.wcsp, time between 30 and 3600 seconds), and there are examples, where wR-AOBF manages to find a more accurate solution in a comparable time, such as on type4b_190_20, for 600 or 3600 seconds.

Figures 9 and 10 present the anytime performance of wAOBF, wR-AOBF, BRAOBB, A*, DFBB and SLS on representative instances using bar charts. Figure 9 shows results for Grids and Pedigrees, while Fig. 10 for WCSPs and Type4, respectively. Each two rows display two problems from the same benchmark for two i-bounds, smaller i-bounds on the left and larger on the right. The height of each bar is proportional to the ratio between the solution cost generated by an algorithm at a time point (at 10, 60, 600 and 3600 seconds) and the optimal cost (if known) or overall maximal cost. The closer the ratio is to 1, the better. For readability we display the ratios greater than 0.7. Above each bar we also show

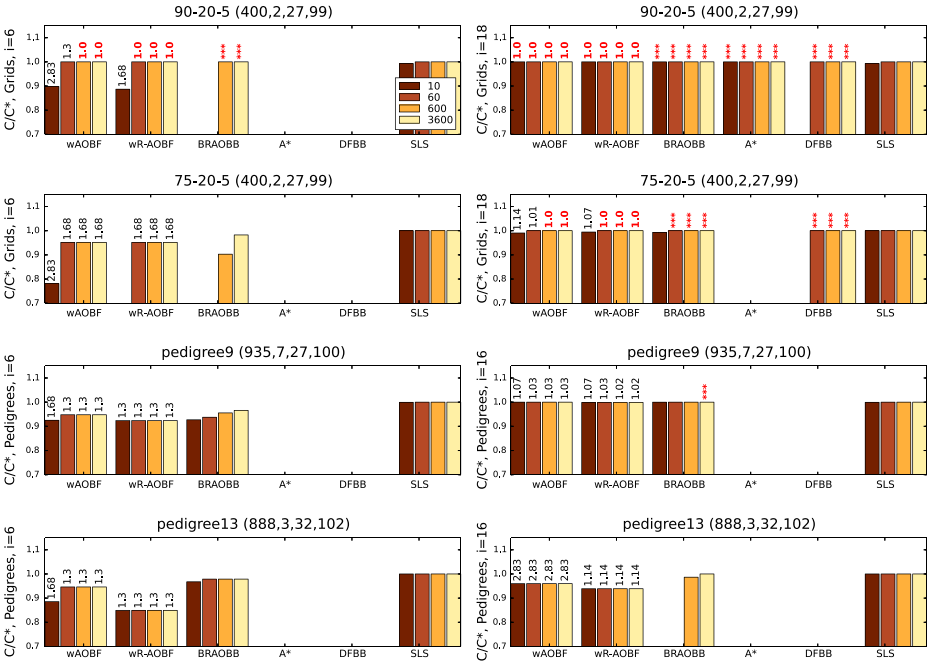


Fig. 9 Ratio of the cost obtained by some time point (10, 60, 600 and 3600 sec) and max cost (optimal, if known, otherwise - best cost found for the problem by any of the schemes). Corresponding weight - above the bars. '***' indicated proven solution optimality. Instance parameters are in format (n, k, w^*, h_T), where n - number of variables, k - max. domain size, w^* - induced width, h_T - pseudo tree height. Memory limit 4 GB, time limit 1 hour. Grids and Pedigrees benchmarks, MBE heuristic

the weight corresponding to the returned solution, where $w = 1$ is shown in red. The symbol '***' above bars indicate proven solution optimality for the schemes using non-weighted heuristic (i.e. BRAOBB, A*, and DFBB).

In these figures we again observe that wR-AOBF has worse anytime behavior than wAOBF. For example, in Fig. 9 for pedigree9 and pedigree13, i-bound=6, wAOBF finds more accurate solutions for the same time bounds.

These observations, based on the selected instances displayed in the table and in the figures, are quite representative, as we will see shortly in the summary Section 5.4.3.

The overall superiority of wAOBF may be explained by the large overhead of wR-AOBF at each iteration, due to the need to keep track of the already expanded nodes and to update their values as the weight changes. As a result, wR-AOBF explores the search space slower and discovers new improved solutions less frequently than wAOBF.

5.4.2 Comparing against state-of-the-art

Figures 9 and 10 provide a different perspective of the strength of weighted heuristic schemes compared against several state-of-the-art anytime algorithms. We observe that the domineering scheme varies from benchmark to benchmark. In Fig. 9 we see that wAOBF is superior on the Grid instances when the heuristic is weak. For example, for grid 75-20-5, i-bound=6 wAOBF is the only scheme that finds a solution within 10 seconds, aside from SLS. Note that SLS does not provide any guarantees, while wAOBF and wR-AOBF report

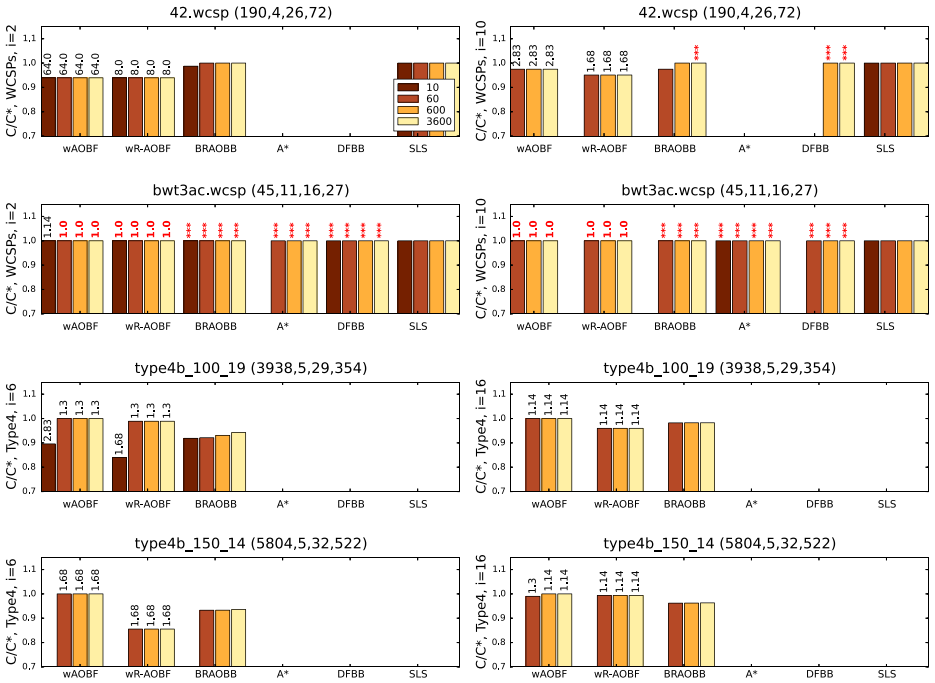


Fig. 10 Ratio of the cost obtained by some time point (10, 60, 600 and 3600 sec) and max cost (optimal, if known, otherwise - best cost found for the problem by any of the schemes). Corresponding weight - above the bars. '***' indicated proven solution optimality. Instance parameters are in format (n, k, w^*, h_T), where n - number of variables, k - max. domain size, w^* - induced width, h_T - pseudo tree height. Memory limit 4 GB, time limit 1 hour. WCSPs and Type4 benchmarks, MBE heuristic

for time bounds of 60 seconds or more solutions that are guaranteed to be within a factor of 1.68 from the optimal. A* and DFBB are always inferior to all other schemes. The former managed to find solutions for a small number of instances and only for highly accurate heuristics, e.g., 90-20-5, i-bound=18. The latter often reports solutions of such low accuracy that they are not seen on the plots, which only show solutions of relative accuracy greater than 0.7.

On Pedigrees, BRAOBB seems to be superior among the complete schemes for weak heuristics, e.g., pedigree9, i-bound=6. SLS reports optimal costs quite fast, usually under 10 seconds. When the heuristics are stronger, the performance is less varied, except for both OR schemes on Grids and on many of the Pedigrees. A* and DFBB are usually inferior even when the i-bound is high, e.g., pedigree13, i-bound=16.

Figure 10 shows that the weighted heuristic best-first schemes are superior to all other schemes for the selected Type4 instances, for all levels of heuristic strengths. Interestingly, on most of the Type4 problems even SLS did not report solutions with relative accuracy greater than 0.7 and thus its results are not shown on our plots, while wAOBF often quickly finds solutions close to optimal.

Notice that the weighted heuristic schemes provide tight w-optimality guarantees in many cases. For example, on the two Type4 instances presented in Fig. 10 wAOBF and wR-AOBF guarantee that the costs reported at 60 seconds are 1.14-optimal. This is in contrast to competing schemes (e.g., BRAOBB) that do not prove optimality within the time

limit and so the accuracy of their solutions is unknown. On WCSPs, however, wAOBF and wR-AOBF are less effective. For example, both are inferior to BRAOBB and SLS on the 42.wcsp instance, for both reported i-bounds. For stronger heuristic even DFBB, which in general is not particularly successful, is superior to the weighted heuristic BF schemes.

As expected, A* and DFBB, both exploring an OR search tree, are clearly inferior to AND/OR schemes, in particular to BRAOBB (see, for example, [23, 24] for more comparisons between AND/OR and OR search). Additionally, A* is not inherently anytime. Stochastic Local Search, although often finding accurate solutions fast, is not a complete algorithm and thus is outside the state-of-the-art schemes we systematically compared with. It also does not provide any bound on its solution.

5.4.3 Summarizing performance of wAOBF and wR-AOBF against BRAOBB

We now show data summarizing the results from all the instances. Figures 11, 12, 13 and 14 provide summaries of our experimental results in the form of bar charts. Note that the summaries include also a weighted heuristic depth-first branch and bound algorithm wAOBB that will be introduced in Section 6. We defer the comparison with wAOBB till Section 6.2.2.

The figures compare each of the three algorithms (wAOBF, wR-AOBF and wAOBB) with BRAOBB. There are two columns of bar charts in each figure. The left one summarizes the percentage of instances in which an algorithm is more accurate compared with BRAOBB, and the right one provides the percentages on instances where an algorithm yields the same costs as BRAOBB. Therefore, the heights of the bars in Figs. 11, 12, 13 and 14 are proportional to these percentages. The ratio at the top of each bar is the actual number of instances where an algorithm is better (left column) or equal (right column), divided by the total number of instances solved by the algorithm. The results in a table form, that include an additional time bound, can be found in Appendix B.

From these figures we see that on two out of four benchmarks the weighted heuristic schemes dominated, finding solutions of better accuracy within the time limits. This superior behavior on Grids and Type4 benchmarks was mostly consistent across time bounds and heuristics of various strengths and can be observed in Figs. 11 and 13. Specifically, for certain i-bounds and time bounds wAOBF returned more accurate solutions than BRAOBB on up to 68 % of the Grid instances and on 90-100 % of the Type4 instances. Interestingly, for the Type4 benchmark the weighted heuristic schemes almost never found the same solution costs as BRAOBB, as is indicated by the zeros in the plots on the right side of Fig. 13.

The comparative strength of wAOBF is more pronounced for short time intervals and for weak heuristics. The dependence of wR-AOBF's performance on time is less predictable. We observe here that for most i-bounds and for most time limits wAOBF performs better than wR-AOBF. This is especially obvious on the Type4 benchmark and for small i-bounds, e.g., for i-bound=6, 3600 seconds, wR-AOBF is superior to BRAOBB only on 50 % of the instances, while wAOBF dominates on 90 %.

Figure 15 uses scatter diagrams to compare between the most successful of the two weighted heuristic best-first schemes, wAOBF, and BRAOBB. Each plot corresponds to a specific time point. The x-axis gives the relative accuracy of the solution obtained by wAOBF, while the y-axis corresponds to the relative accuracy of BRAOBB. As for the bar charts, the accuracy is defined relative to the optimal cost, if known, or to the best cost found. Values closer to 1.0 indicate better results. In each row we show two time bounds for a particular benchmark. In parenthesis we show the number of instances, for which at least one of the displayed algorithms found a solution, and the total number of instances in the

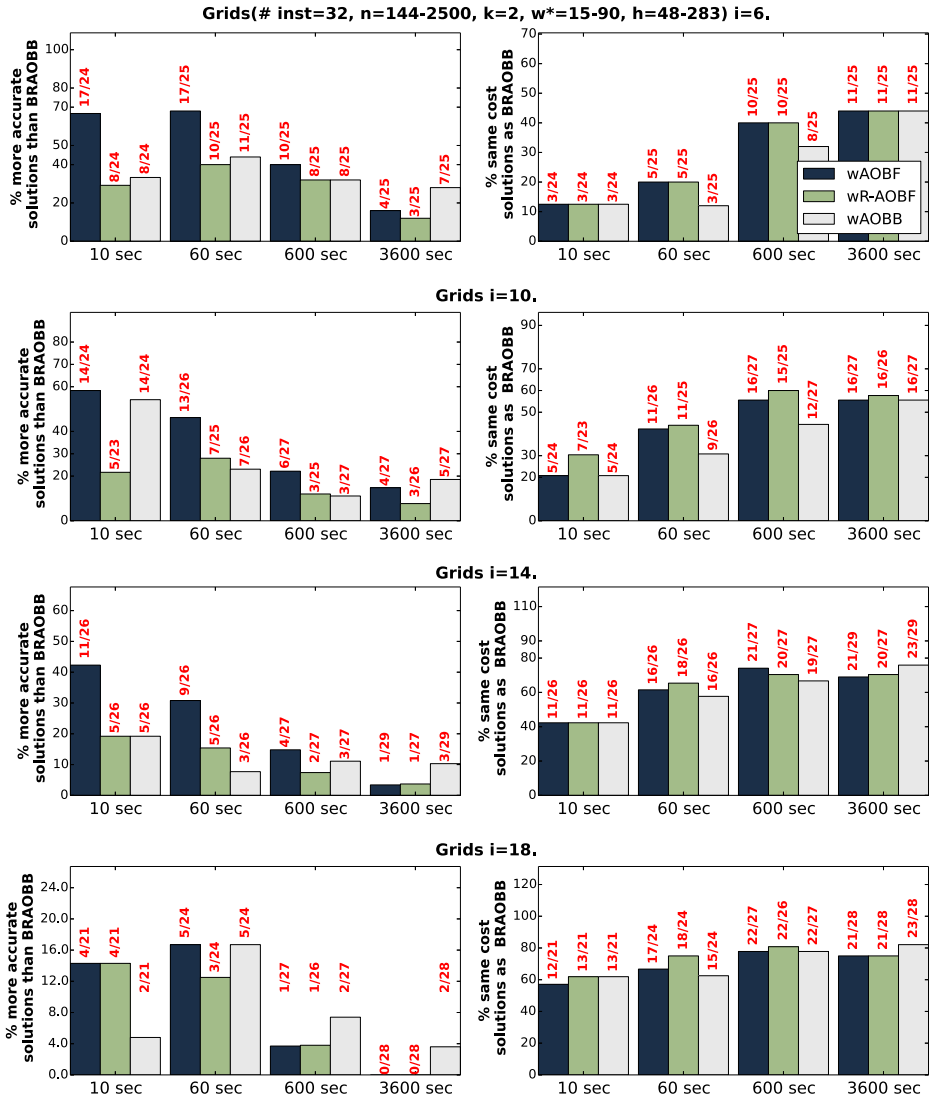


Fig. 11 Grids: percentage of instances for which each algorithm found solutions of more accurate cost than BRAOBB at a specific time bound (*left*), percentage of instances for which algorithm is tied with BRAOBB, i.e. found solution of equal cost (*right*). Above bars - number of instances where algorithm is better than BRAOBB (tied with BRAOBB) and number of instances solved by the algorithm. Parameters: # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo tree height. 4 GB memory, 1 hour time limit

benchmark. Each marker represents an instance. The markers under the red diagonal correspond to problems where wAOBF is superior. We do not account in these plots for the MBE heuristic calculation time, which is the same for all schemes.

We show results with the relatively weak heuristic because it yields greater diversity in solution accuracy by wAOBF and BRAOBB. Typically, for strong heuristics the algorithms

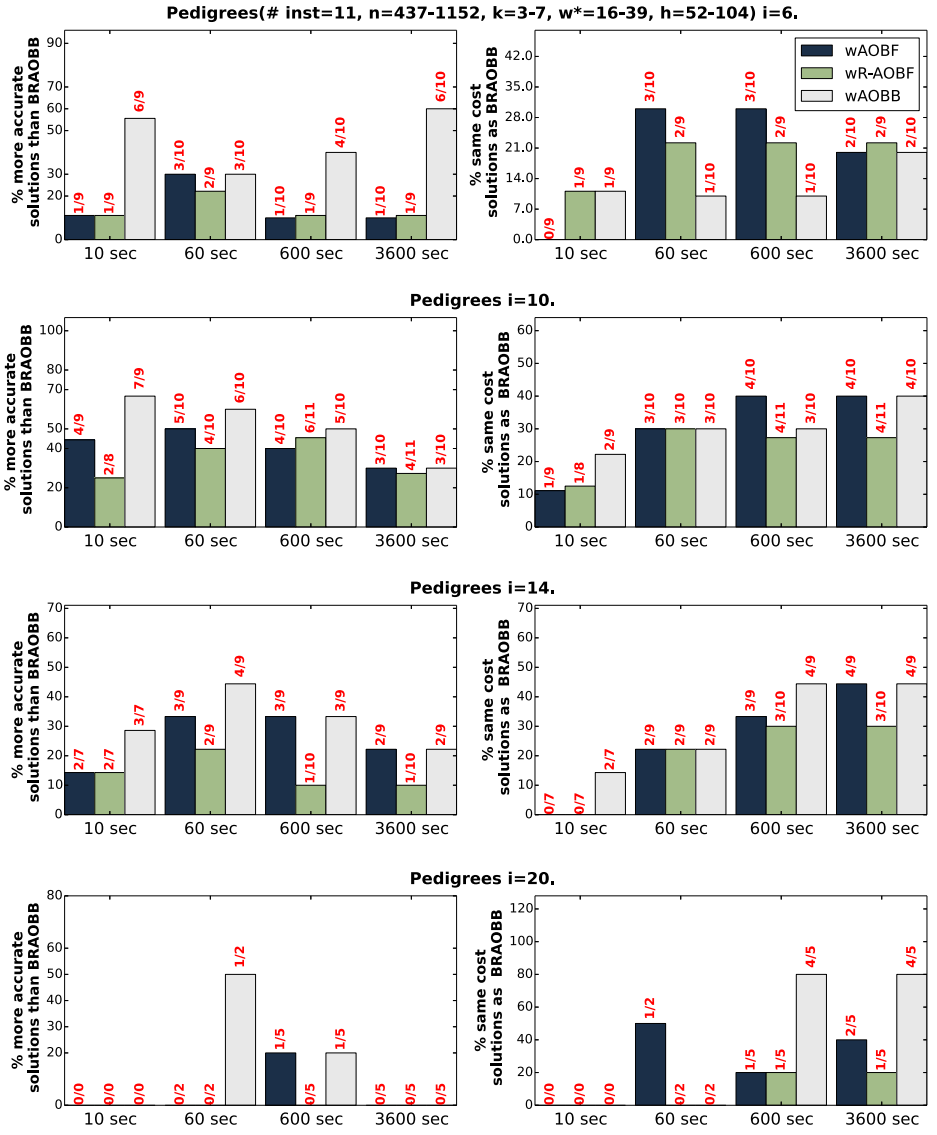


Fig. 12 Pedigrees: percentage of instances for which each algorithm found solutions of more accurate cost than BRAOBB at a specific time bound (*left*), percentage of instances for which algorithm is tied with BRAOBB, i.e. found solution of equal cost (*right*). Above bars - number of instances where algorithm is better than BRAOBB (tied with BRAOBB) and number of instances solved by the algorithm. Parameters: # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo tree height. 4 GB memory, 1 hour time limit

yield solutions having similar costs. Appendix B includes additional scatter diagrams showing results for three i-bounds and three time bounds per benchmark, along with scatter plots comparing wAOBF and wR-AOBF, respectively.

Figure 15 confirms our previous conclusions on the superiority of wAOBF over BRAOBB on the Grids benchmark and its lack of success on WCSPs. On Type4 for small

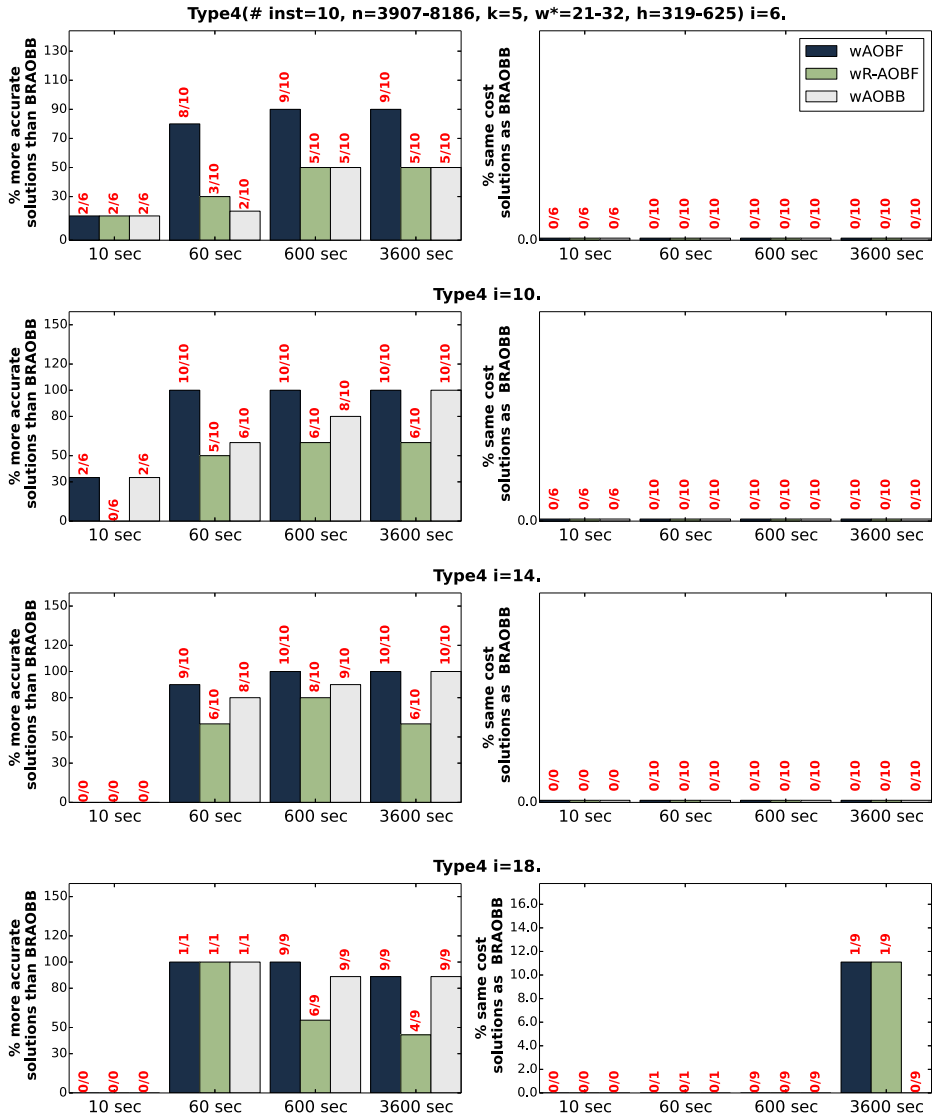


Fig. 13 Type4: percentage of instances for which each algorithm found solutions of more accurate cost than BRAOBB at a specific time bound (*left*), percentage of instances for which algorithm is tied with BRAOBB, i.e. found solution of equal cost (*right*). Above bars - number of instances where algorithm is better than BRAOBB (tied with BRAOBB) and number of instances solved by the algorithm. Parameters: # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo tree height. 4 GB memory, 1 hour time limit

time limits BRAOBB and wAOBF are on average equally successful (e.g., for 10 sec), but when given more time, wAOBF produces more accurate solutions (e.g., for 600 seconds). For Pedigrees there is no clear dominance.

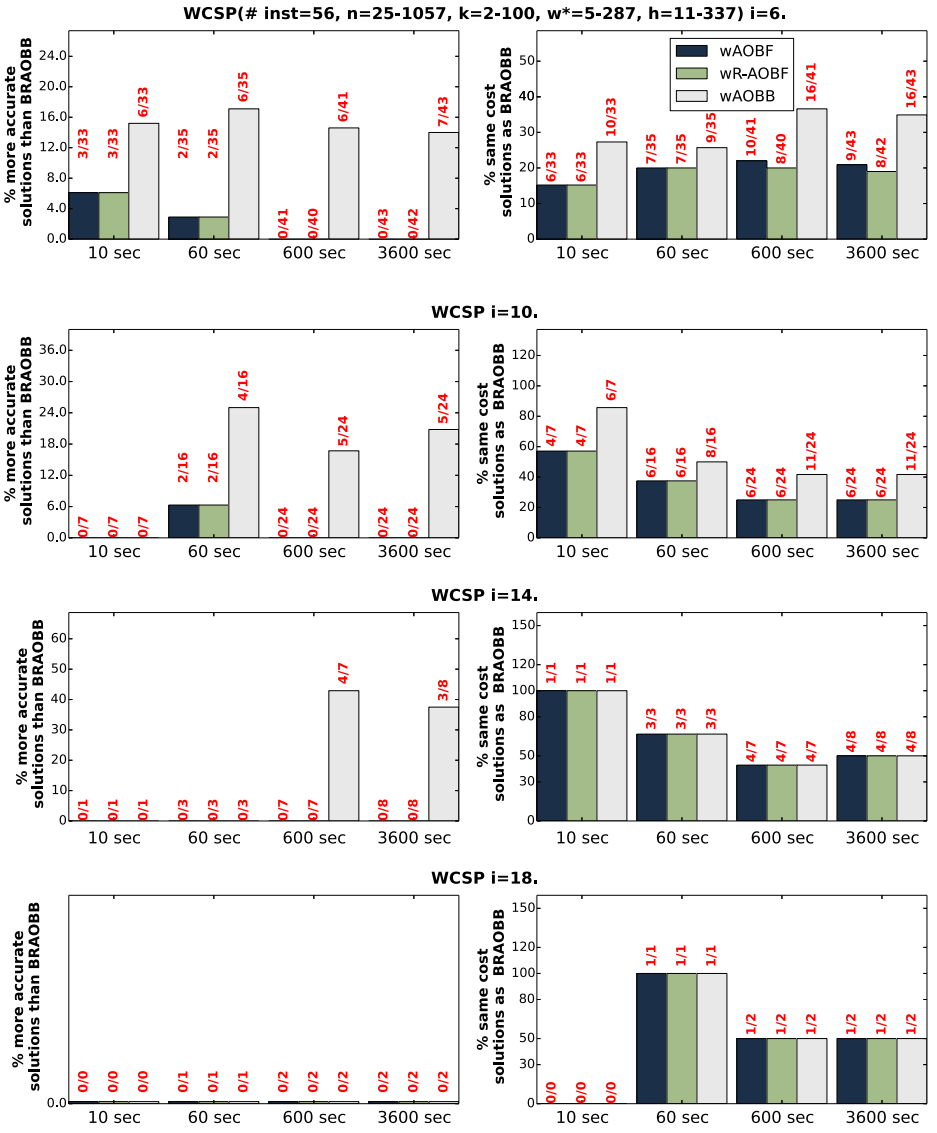


Fig. 14 WCSP: percentage of instances for which each algorithm found solutions of more accurate cost than BRAOBB at a specific time bound (*left*), percentage of instances for which algorithm is tied with BRAOBB, i.e. found solution of equal cost (*right*). Above bars - number of instances where algorithm is better than BRAOBB (tied with BRAOBB) and number of instances solved by the algorithm. Parameters: # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo tree height. 4 GB memory, 1 hour time limit

6 Weighted heuristic depth-first branch and bound for graphical models

The primary reason for using weighted heuristics in the context of best-first search is to convert it into memory-effective anytime scheme and to get a solution with some bounded

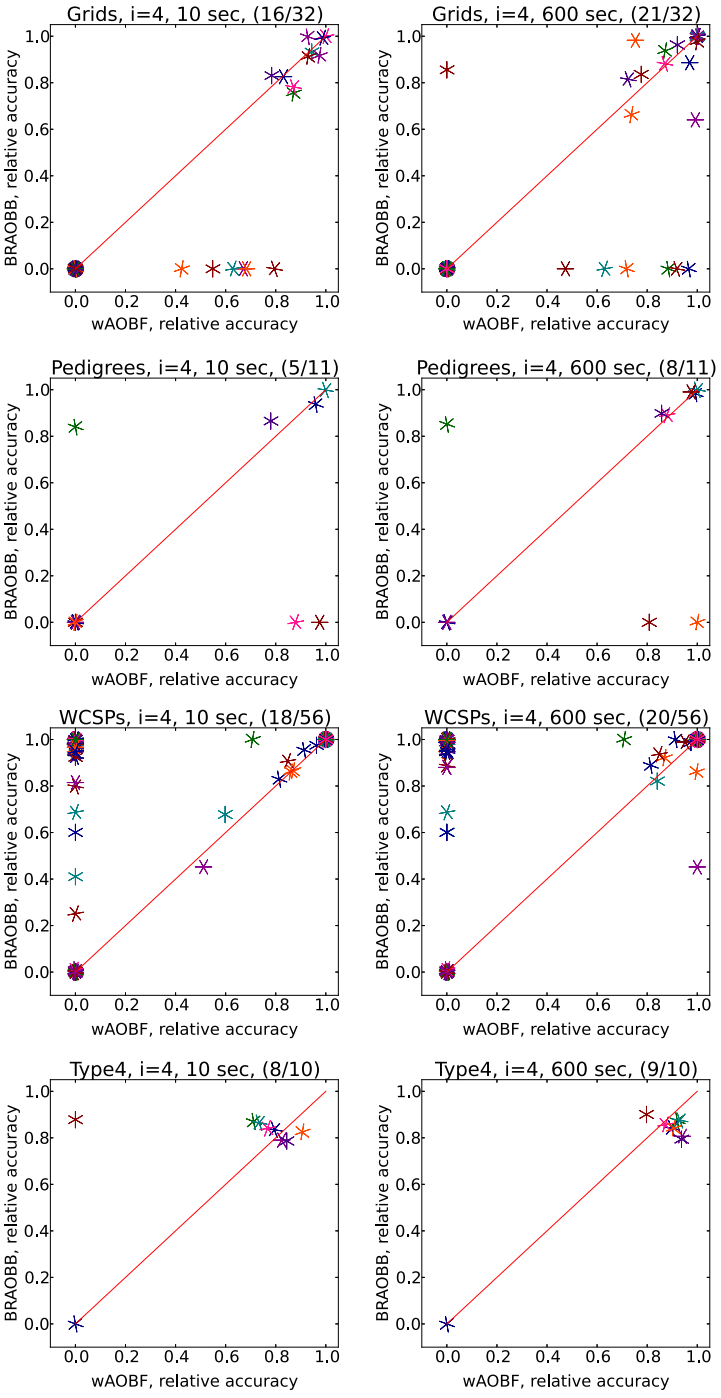


Fig. 15 wAOBF vs BRAOBB: comparison of relative accuracy at times 10, 3600 sec. Each row - a single time bound. Each marker represents a single instance. Memory limit 4 GB, time limit 1 hour. In parenthesis (X/Y): X - # instances, for which at least one algorithm found a solution, Y - total # instances, MBE heuristic

guarantee. Since depth-first branch and bound schemes are already inherently anytime, the idea of using weighted heuristic search may seem irrelevant. However, branch and bound schemes do not provide any guarantees when terminating early. So a desire to have beneficial w-optimality bounds on solutions and to possibly improve the anytime performance intrigued us into exploring the principle of weighted heuristic search for depth-first search schemes as well. Specifically, weighted heuristic may guide the traversal of the search space in a richer manner and may lead to a larger and more effective pruning of the space.

Therefore, in this section we extend the depth-first branch and bound algorithms AOBB and BRAOBB to weighted heuristic anytime schemes, yielding wAOBB and wBRAOBB, and evaluate their performance in much the same way we did for the weighted heuristic best-first search algorithms.

6.1 Introducing the weighted heuristic branch and bound schemes

The extension of AOBB to weighted heuristic search is straightforward. Just multiply the heuristic value by the weight $w > 1$ and conduct AOBB as usual. We denote by AOBB(h_i, w_0,UB) a weighted heuristic version of AOBB that uses the mini-bucket heuristic h_i having i-bound=i, multiplied by the weight w_0, and an initial upper bound equal to UB, as shown in Algorithm 2. It is easy to show that:

Theorem 9 *Algorithm AOBB(h_i, $w_0 > 1$,$UB = \infty$) (and similarly BRAOBB) terminates with a solution π, whose cost C_π is a factor w_0 away from the optimal cost C^*. Namely, $C_\pi \leq w_0 \cdot C^*$.*

Proof By definition, due to pruning, AOBB generates solutions in order of decreasing costs: $C_1 \geq \cdots \geq C_i \cdots \geq C_\pi$, where C_π is the returned solution. If π is not optimal (otherwise the claim is trivially proved), there exists an optimal solution π^* which must have been pruned by the algorithm. Let n be the last node on π^* that was generated and which was pruned. Since the heuristic h is admissible, the un-weighted evaluation function of f along π^* satisfies that

$$f_{\pi^*}(n) = g(n) + h(n) \leq g(n) + h^*(n) = C^* \tag{6}$$

Let C_i be the solution cost used to prune n (namely it was pruned relative to the weighted evaluation function). Therefore,

$$C_i \leq g(n) + w_0 \cdot h(n)$$

Therefore (as $w_0 \geq 1$) and from (6)

$$C_i \leq w_0 \cdot (g(n) + h(n)) \leq w_0 \cdot C^*$$

and since $C_\pi \leq C_i$, we get

$$C_\pi \leq w_0 \cdot C^*.$$

□

We present two iterative weighted heuristic branch and bound schemes denoted wAOBB and wBRAOBB. Similar to wAOBF, these algorithms iteratively execute the corresponding base algorithm with the weighted heuristic, AOBB(h_i, w_0,UB) and BRAOBB(h_i, w_0,UB), respectively. However, there are some inherent differences between these two schemes and wAOBF, explained next.

Iterative weighted heuristic AOBB (wAOBB) At the first iteration (Algorithm 6) wAOBB executes AOBB(h_i, w_0, $UB = \infty$), namely AOBB with heuristic $h_i \cdot w_0$ and with default upper bound of infinity. The algorithm does no pruning until it discovers its first solution. Then the upper bound is set to the current best cost. At termination of the first iteration, the algorithm returns the final solution and its cost C_1 with the corresponding weight $w_1 = w_0$. During each subsequent iteration $j \geq 2$ wAOBB executes AOBB(h_i, w_j, UB_j) to completion. The weight w_j is decreased according to the weight policy. The input upper bound UB_j is the cost of the solution returned in iteration $j-1$, i.e., $UB_j = C_{j-1}$. We denote by C'_j the costs of the intermediate solutions wAOBB generates during iteration j, until it terminates with the final solution, having cost C_j.

Proposition 4 *At each iteration $j > 0$ the solution cost C_j of AOBB(h_i, w_j, UB_j) is guaranteed to be within the factor w_j from the optimal cost C^*. Moreover, for iterations $j \geq 1$ all the intermediate solutions generated by AOBB(h_i, w_j, UB_j) are guaranteed to have costs within the factor of w_{j-1} from the optimal.*

Proof The solution cost C_j with which AOBB(h_i, w_j, UB_j) terminates at iteration j is bounded: $C_{w_j} \leq w_j \cdot C^*$. The upper bound used for pruning at iteration $j > 1$ is equal to the cost of the solution on the previous iteration ($UB_j = C_{j-1}$). No intermediate solutions worse than this upper bound are ever explored. Thus the costs C'_j of all solutions generated at iteration j prior to its termination are bounded by the upper bound $C'_j \leq C_{j-1}$. Since C_{j-1} is bounded by a factor of w_{j-1} from the optimal, it follows $C'_j \leq w_j \cdot C^*$. □

Figure 16 shows the search space explored by AOBB with unweighted heuristic (on the left) and by the first iteration of wAOBB with weight $w = 10$ (on the right), when solving the problem from Fig. 4. In this example, AOBB explores 15 nodes in order to find an optimal solution. wAOBB explores 12 nodes, discovering a sub-optimal solution with cost $C = 8$ and assignment $\mathbf{x} = \{A = 0, B = 0, C = 1, D = 0\}$.

Iterative weighted heuristic BRAOBB (wBRAOBB) wBRAOBB extends BRAOBB(h_i, w_0,UB) to a weighted heuristic iterative scheme in the same manner. Clearly, for both schemes the sequence of the solution costs is non-increasing.

Algorithm 6 wAOBB(w_0, h_i)

Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$; heuristic h_i obtained with i-bound i; initial weight w_0
Output: $\mathscr{C}$ - a set of sub-optimal solutions C_w, each with a bound w

```
1  Initialize j = 1, UB_j = ∞, w_j = w_0, weight update schedule S and let C ← ∅;
2  while w_j >= 1 do
3      while AOBB(h_i,w_j, UB_j) not terminated do
4          run AOBB(h_i,w_j, UB_j)
5          if AOBB found an intermediate solution C'_j then
6              output the solution bounded by the weight of previous iteration: C ← C ∪ {⟨w_{j-1},C'_j⟩}
7      output the solution with which AOBB terminated, bounded by the current weight:
       C ← C ∪ {⟨w_j,C_j⟩}
8      Decrease weight w according to schedule S;
9      UB ← C_j
10 return C
```

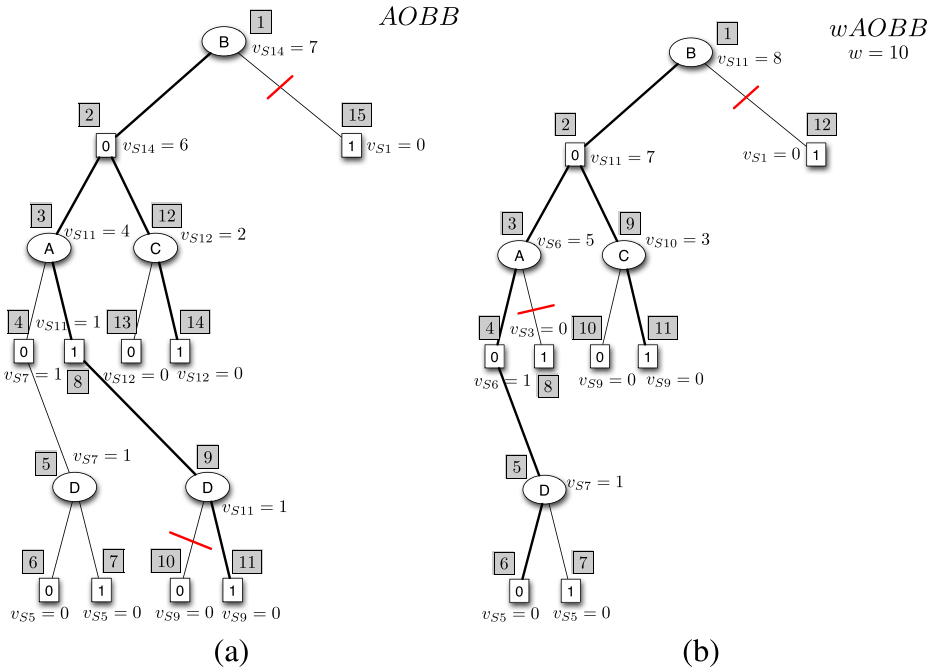


Fig. 16 Search graphs explored by **a** AOBB, **b** First iteration of wAOBB, w=10. The numbers in grey boxes indicate the order in which the nodes are expanded. v_{SN} indicates that value v was last assigned during step N, i.e. while expanding the N^{th} node

Theorem 10 *The worst case time and space complexity of each iteration of wAOBB and wBRAOBB that uses full caching is bounded by* $O(n \cdot k^{w^*})$. *The number of iterations varies based on weight policy and start value* w_0, *respectively.*

Proof At each iteration wAOBB and wBRAOBB execute AOBB and BRAOBB respectively, exploring at most the entire context-minimal AND/OR search graph. The precise number of expanded nodes depends on the pruning and is hard to characterize in general, as is always the case for branch and bound search. □

6.2 Empirical evaluation of weighted heuristic BB

We carry out an empirical evaluation of the two weighted heuristic depth-first branch and bound schemes: wAOBB and wBRAOBB, described above. The algorithms were implemented in C++ and the experiments were conducted in the same setting as before.

We compared the two weighted heuristic branch and bound schemes against each other and against wAOBF, the superior of the weighted heuristic best-first schemes. We also compared against the state-of-the-art anytime BRAOBB. All algorithms use the same MBE heuristic. We use the same $sqrt(1.0)$ policy and starting weight equal to 64 as we did for the weighted heuristic best-first schemes in Section 5.

Table 4 Runtime (sec) and cost (on logarithmic scale) obtained by wAOBF, wAOBB and wBRAOBB for selected w, and by BRAOBB (that finds C^* - optimal cost)

Instance	BRAOBB	Weights			
		2.8284	1.2968	1.0330	1.000
		wAOBF wAOBB wBRAOBB	wAOBF wAOBB wBRAOBB	wAOBF wAOBB wBRAOBB	wAOBF wAOBB wBRAOBB
	time / cost	time / cost	time / cost	time / cost	time / cost
Grids, I-bound=18					
75-22-5 (484, 2, 30, 107)	115.45 / -15.605	5.92 / -15.72 5.87 / -19.08 5.91 / -19.08	6.66 / -15.72 6.1 / -17.55 6.22 / -17.55	59.81 / -15.61 52.46 / -15.65 79.91 / -15.7	423.76 / -15.61 — / — — / —
75-25-5 (625, 2, 34, 122)	3582.08 / -20.836	8.0 / -23.38 8.08 / -31.0 8.14 / -31.0	9.77 / -21.7 8.34 / -22.55 8.56 / -22.84	— / — — / — — / —	— / — — / — — / —
Pedigrees, I-bound=18					
pedigree9 (935, 7, 27, 100)	220.34 / -122.904	26.75 / -129.55 12.44 / -129.76 12.59 / -129.76	26.96 / -123.06 12.47 / -128.56 12.61 / -128.56	33.56 / -122.9 13.63 / -123.2 13.74 / -123.2	— / — — / — — / —
pedigree51 (871, 5, 39, 98)	3600 / -111.55	120.94 / -119.16 29.01 / -121.77 26.41 / -121.77	— / — 31.15 / -121.77 28.36 / -121.77	— / — — / — 3035.48 / -109.83	— / — — / — — / —
WCSP, I-bound=6					
capmo2.wcsp (200, 100, 100, 100)	3600 / -0.28	26.81 / -0.31 22.43 / -0.31 22.47 / -0.31	— / — — / — — / —	— / — — / — — / —	— / — — / — — / —
myciel5g_3.wcsp (47, 3, 19, 24)	12.93 / -64.0	2.52 / -72.0 0.96 / -72.0 0.9 / -72.0	50.47 / -64.0 7.8 / -64.0 7.37 / -64.0	— / — 37.63 / -64.0 35.63 / -64.0	— / — — / — — / —
Type4, I-bound=18					
type4b_120_17 (4072, 5, 24, 319)	3600 / -1332.18	80.49 / -1354.93 49.79 / -1353.83 54.91 / -1353.83	81.24 / -1329.59 49.97 / -1337.37 55.12 / -1337.37	84.98 / -1327.6 50.25 / -1334.85 55.44 / -1334.85	— / — — / — — / —
type4b_130_21 (4874, 5, 29, 416)	3600 / -1383.74	86.21 / -1438.24 58.12 / -1414.3 96.61 / -1512.32	88.89 / -1386.34 97.66 / -1405.72 96.93 / -1489.57	— / — — / — — / —	— / — — / — — / —

Instance parameters: n - number of variables, k - max domain size, w^* - induced width, h_T - pseudo tree height. "time out" - running out of time, "—" - running our of memory. 4 GB memory limit, 1 hour time limit, MBE heuristic

6.2.1 Weighted heuristic BB as approximation

In Table 4 we report the entire runtime required to find a first solution for a particular weight level (e.g., $w = 2.8284$) for each algorithm. In this sense these tables are different from Table 2, where we only reported the time it took the scheme to find the w-optimal solution starting from a particular weight, e.g., $w = 2.8284$, and not starting with initial weight $w_0 = 64$. The difference is due to the fact that wAOBB and wBRAOBB use the results of

previous iterations as upper bounds and their iterations are not completely independent runs of AOBB(h_i, w_j, UB_j) and BRAOBB(h_i, w_j, UB_j), respectively.

We also report the runtime and the solution cost found by BRAOBB at termination. The time equal to 3600 seconds for BRAOBB indicates that it failed to report the optimal solution within the time bound and we then report the best solution found. Interesting entries are highlighted.

Comparing with weighted heuristic best-first search Based on the table we see that time-wise none of the schemes dominates for a given weight. For example, in Table 4 for grid 75-22-5, for weight $w = 1.033$ wAOBB reports the solution the fastest, while for type4b_130_21 wAOBF reaches a 1.2968-optimal solution almost ten seconds before either wAOBB or wBRAOBB.

Time saving for w-bounded sub-optimality Comparing pairs of columns, in particular column 2 (exact results for BRAOBB) and columns 4-5 (1.2968- and 1.0330-optimal solutions) in Table 4, we observe remarkable time savings of the weighted heuristic schemes compared with BRAOBB.

Overall, based on these instances, which are quite representative, we see as before, that the weighted heuristic schemes can often provide good approximate solutions with tight sub-optimality bounds, yielding significant time savings compared to finding optimal solutions by running BRAOBB to completion. The weighted heuristic branch and bound schemes are more memory efficient than wAOBF. However, on the instances feasible for all three weighted heuristic schemes there is no clear winner.

6.2.2 Anytime performance comparison

Figures 17 and 18 display the anytime behavior of the schemes for typical instances from each benchmark using bar charts that show the ratio between the cost available at a particular time point (at 10, 60, 600 and 1800 sec) and the optimal (if known) or best cost found (similarly to Figs. 9 and 10 in the previous section). Figure 17 shows the results for Grids and Pedigrees, while Fig. 18 presents WCSPs and Type4 instances.

Grids (Fig. 17): we observe that the weighted heuristic branch and bound schemes are typically inferior to wAOBF, finding solutions slower and of lower accuracy. For example, wAOBF is the only scheme to return a solution within 10 seconds on grid 75-23-5, i-bound=6. However, there are exceptions (e.g., grid 50-16-5, i-bound=6).

Pedigrees both weighted heuristic branch and bound schemes are often the best (e.g., pedigree13, i-bound=6). For some problems they even provide solutions with accuracy approaching 1.0 while the other schemes fail to find any solution, e.g., pedigree7, i-bound=6.

WCSPs (Fig. 18): The wAOBB and wBRAOBB algorithms perform better than BRAOBB and wAOBF on these instances (e.g., capmo2.wcsp for all i-bounds), except for a number of problems (not explicitly shown), for which BRAOBB is the only scheme to return solutions.

Type4 wAOBF typically dominates the branch and bound schemes, including the weighted heuristic ones (wAOBB, wBRAOBB). However, for larger i-bounds on Type4, weighted

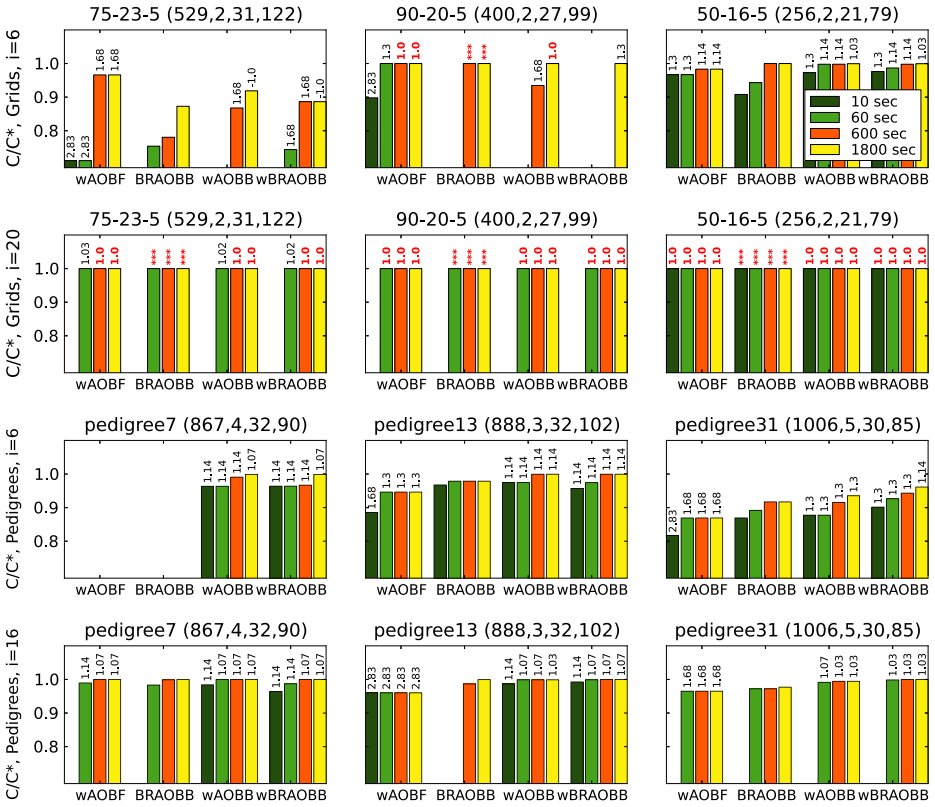


Fig. 17 Ratio of the cost obtained by some time point (10, 60, 600 and 1800 sec) and max cost. Max. cost = optimal, if known, otherwise = best cost found for the problem. Corresponding weight - above the bars. The cases where BRAOBB proved solution optimality is indicated by '***' above bars. In red - optimal solutions. Instance parameters are in format (n, k, w^*, h_T), where n - number of variables, k - max. domain size, w^* - induced width, h_T - pseudo tree height. Grids and Pedigrees. Memory limit 4 GB, time limit 1 hour, MBE heuristic

heuristic branch and bound schemes can find good solutions, sometimes even providing tighter sub-optimality guarantees than wAOBF. For example, for type4b_140_20, i-bound=12, for 1800 seconds the bound by wAOBF is $w = 1.3$, while for wAOBB it is $w = 1.14$. BRAOBB is inferior for this benchmark.

We now turn to Figs. 11, 12, 13 and 14 in order to summarize the performance of weighted heuristic depth-first branch and bound search algorithms, concentrating on wAOBB, as it is slightly better among the two. From these bar charts we see that wAOBB is more successful than BRAOBB when the heuristics are weak. For example, for Grids, 60 second, for i-bound=10, wAOBB finds solutions of higher accuracy than BRAOBB on 23.1 % of instances, while for i-bound=18, superiority is 16.7 %. The two schemes are tied on 82.1 % of instances for Grids, i-bound=18, 3600 seconds, and in general, when the i-bound is high, the difference between the solution costs reported by various algorithms diminishes.

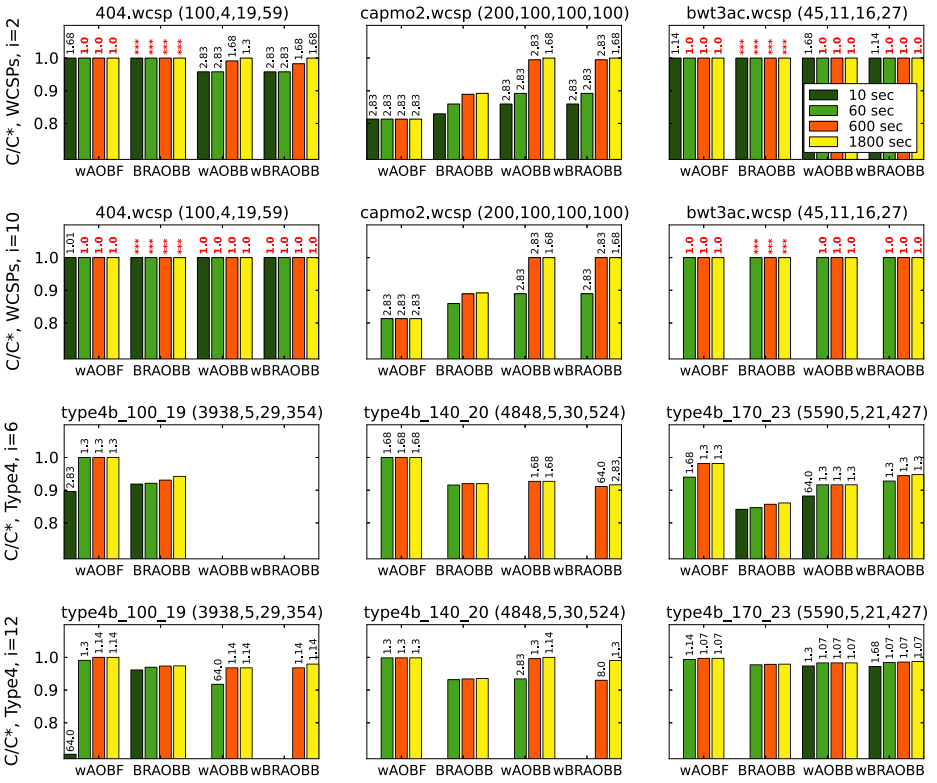


Fig. 18 Ratio of the cost obtained by some time point (10, 60, 600 and 1800 sec) and max cost. Max. cost = optimal, if known, otherwise = best cost found for the problem. Corresponding weight - above the bars. The cases where BRAOBB proved solution optimality is indicated by '***' above bars. In red - optimal solutions. Instance parameters are in format (n, k, w^*, h_T), where n - number of variables, k - max. domain size, w^* - induced width, h_T - pseudo tree height. Type4 and WCSPs. Memory limit 4 GB, time limit 1 hour, MBE heuristic

We conclude from these figures that wAOBB can definitely be superior to wAOBF on 2 out of 4 benchmarks (Pedigrees and WCSPs) for more instances, and can find solutions of better accuracy than BRAOBB on a number of instances from all four benchmarks.

7 Summary and concluding remarks

The paper provides the first study of weighted heuristic best-first and weighted heuristic depth-first branch and bound search for graphical models. These algorithms are distinguished by their ability to provide a w-optimality guarantee (namely they can guarantee solutions that are at most a w factor away from the optimal cost, for a given weight w). Alternatively, when run in anytime fashion, whenever stopped, they generate the best solution encountered thus far and a weight w bounding its sub-optimality.

The idea of weighted heuristic best-first search is widespread in the path-finding and planning communities. In this paper, we extended this idea to graphical models optimization tasks, specifically, to the AND/OR Best-First search scheme (AOBF) and the AND/OR

Depth-First Branch and Bound scheme (AOBB) (see [26]). This resulted in two anytime weighted heuristic best-first schemes, wAOBF and wR-AOBF, and two weighted heuristic depth-first schemes, wAOBB and wBRAOBB. We evaluated these algorithms against each other and against several competing schemes, and especially against the state-of-the-art BRAOBB [29], on a large variety of instances from four benchmarks. We evaluated the algorithms for varying heuristic strength and for different time bounds. The heuristic functions were generated by the mini-bucket elimination scheme [6], whose strength is controlled by an i-bound parameter. Our experiments revealed the following primary trends:

- **On the anytime performance of weighted heuristic schemes:** First and foremost we showed that the weighted heuristic schemes can perform better than BRAOBB in many cases. In addition, the schemes provide the useful w-optimality guarantee. We saw that overall wAOBF had a better anytime behavior than wR-AOBF and produced more accurate solutions in a comparable time on many instances (see Figs. 11, 12, 13, 14 and 15).
- **Performance varied per benchmark:** On two out of four benchmarks wAOBF and wR-AOBF dominated BRAOBB, often finding better accuracy solutions within the time limit. This good behavior on Grids and Type4 benchmarks was mostly consistent across heuristic strengths and time bounds. The weighted heuristic depth-first search schemes wAOBB and wBRAOBB were better than wAOBF and BRAOBB on Pedigrees and, especially, on WCSPs. This dominance often corresponded to cases where wAOBF ran out of memory, since branch and bound schemes are more memory efficient. Therefore, together the weighted heuristic schemes had an effective performance on instances across all benchmarks.
- **On the impact of the heuristic strength:** the weighted heuristic schemes were more powerful compared with BRAOBB for weak heuristics, namely, when the i-bound characterizing the heuristic strength was far smaller than the problem's induced width. One explanation is that the weight may make the weak admissible heuristic more accurate (a weak lower bound may become stronger lower bound, closer to the actual optimal cost, when multiplied by a constant).
- **On w-optimality in practice:** In quite a few cases the weighted heuristic schemes (e.g., wAOBF, wR-AOBF, wAOBB and wBRAOBB) reported solutions orders of magnitude faster, even for small w, than the time required to generate an exact solution by the schemes BRAOBB or AOBF that used admissible (un-weighted) heuristics. Moreover, in some cases the weighted heuristic schemes generated w-optimal solutions for a small w even for some hard instances that were infeasible (within the time limit) for the baseline algorithms.

Selection and combination of algorithms Clearly, however, due to the fact that no algorithm is always superior, the question of algorithm selection requires further investigation. We aim to identify problem features that could be used to predict which scheme is best suited for solving a particular instance, and to combine the algorithms within a portfolio framework, known to be successful for such solvers as SATzilla [39] and PBP [10].

In this context, however, it is important to note that the weighted heuristic schemes can be valuable anyway by supplying any approximation with w-optimality guarantees. For example, it can be used alongside an incomplete approximate schemes, such as Stochastic Local Search. If the solution by SLS has a cost better or equal to that of the generated w-optimal solution, it yields a w-optimal solution to SLS solution.

The potential impact of weights on various heuristics While we evaluated the weighted heuristic schemes relative to the mini-bucket heuristics only, these ideas are orthogonal to the heuristic type and they are likely to similarly boost the anytime behavior with any other heuristics. We have recently provided some initial empirical evaluation on the impact of the weighted heuristic schemes for cost-shifting relaxation schemes that augment the mini-bucket scheme, introduced in [14]. Our initial findings can be found in [33].

Acknowledgments This work was sponsored in part by NSF grants IIS-1065618 and IIS-1254071, and by the United States Air Force under Contract No. FA8750-14-C-0011 under the DARPA PPAML program.

Appendix A: Weighted heuristic BF: summaries by scatter diagrams

In this section we include additional scatter plots showing in Fig. 19 the comparison between wAOBF and wR-AOBF for two time bound and two levels of heuristic strength and, in Figs. 20, 21, 22 and 23 the comparison between wAOBF and BRAOBB for three values of i-bounds and three time bounds for each instance. For completeness, we also include some of the plots previously presented in Section 5.4.3.

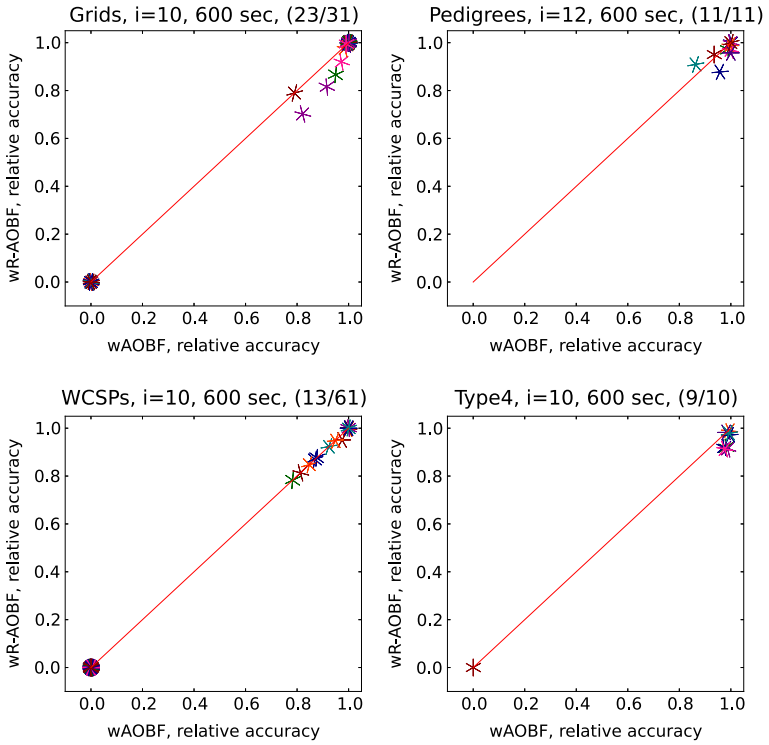


Fig. 19 wAOBF vs wR-AOBF, all benchmarks: comparison of relative accuracy at 600 sec. Each marker represents a single instance. Memory limit 4 GB, time limit 1 hour. In parenthesis (X/Y): X - # instances, for which at least one algorithm found a solution, Y - total # instances, MBE heuristic

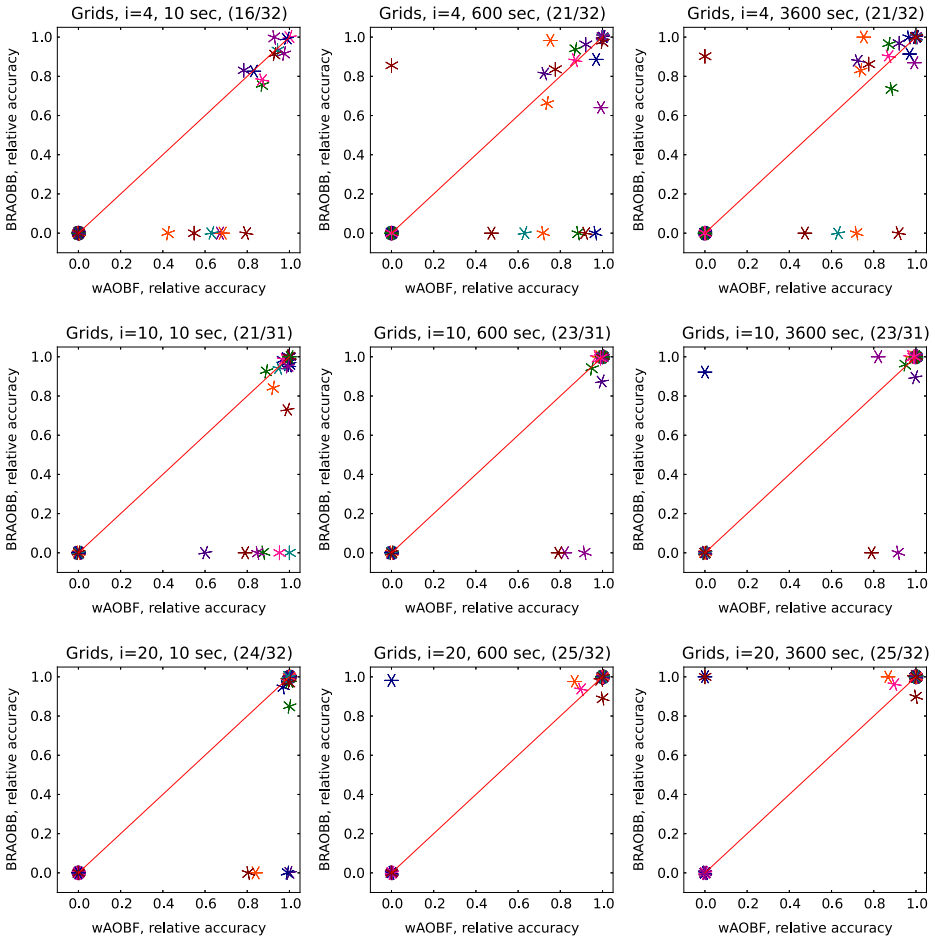


Fig. 20 wAOBF vs BRAOBB on Grids: comparison of relative accuracy at times 10, 600 and 3600 sec. Each row - a single time bound. Each marker represents a single instance. Memory limit 4 GB, time limit 1 hour. In parenthesis (X/Y): X - # instances, for which at least one algorithm found a solution, Y - total # instances, MBE heuristic

Appendix B: Comparing weighted heuristic search schemes against BRAOBB

Tables 5 and 6 present the summary of the results for wAOBF, wR-AOBF and wAOBB compared against BRAOBB. For each benchmark, for five time bounds and for four i-bounds, we show the percentages of the instances for which a particular weighted heuristic algorithm found a more accurate solution than BRAOBB ($\%X$), and for which the algorithm found the solution of the same cost as BRAOBB ($\%Y$). We also show the number of instances solved by each algorithm (N).

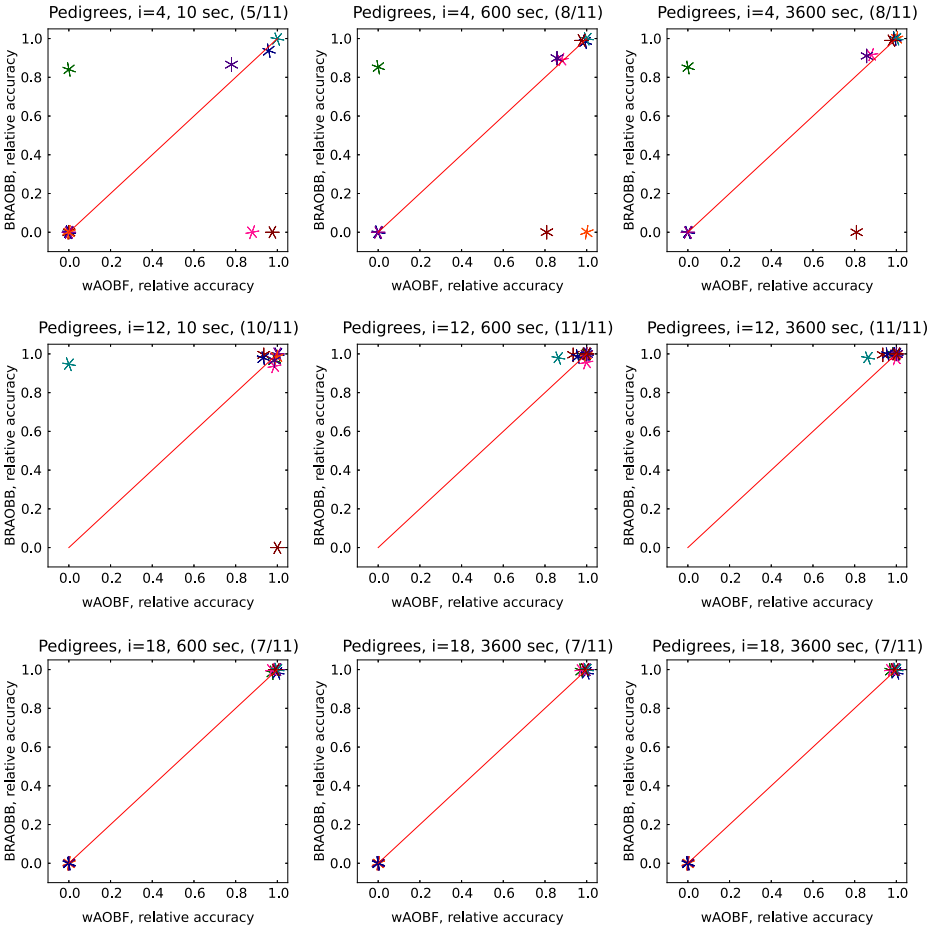


Fig. 21 wAOBF vs BRAOBB on Pedigrees. comparison of relative accuracy at times 10, 600 and 3600 sec. Each row - a single time bound. Each marker represents a single instance. Memory limit 4 GB, time limit 1 hour. In parenthesis (X/Y): X - # instances, for which at least one algorithm found a solution, Y - total # instances, MBE heuristic

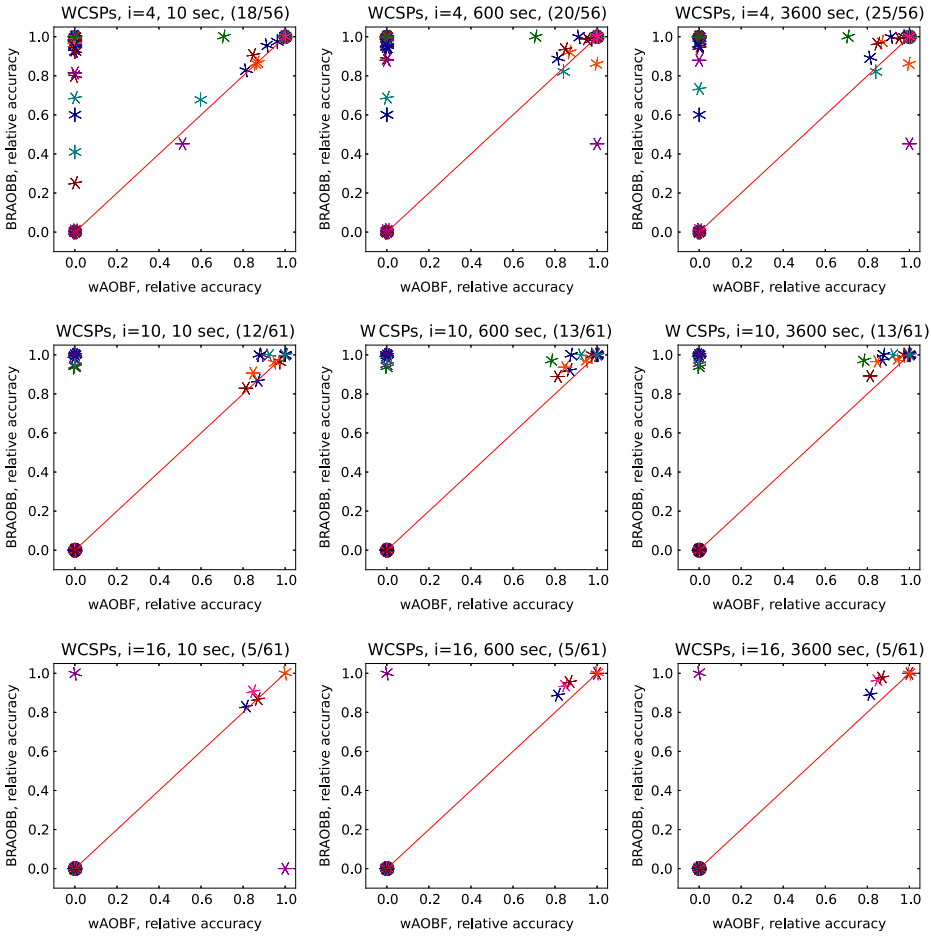


Fig. 22 wAOBF vs BRAOBB on WCSPs: comparison of relative accuracy at times 10, 600 and 3600 sec. Each row - a single time bound. Each marker is a single instance. Memory limit 4 GB, time limit 1 hour. In parenthesis (X/Y): X - # instances, for which at least one algorithm found a solution, Y - total # instances, MBE heuristic

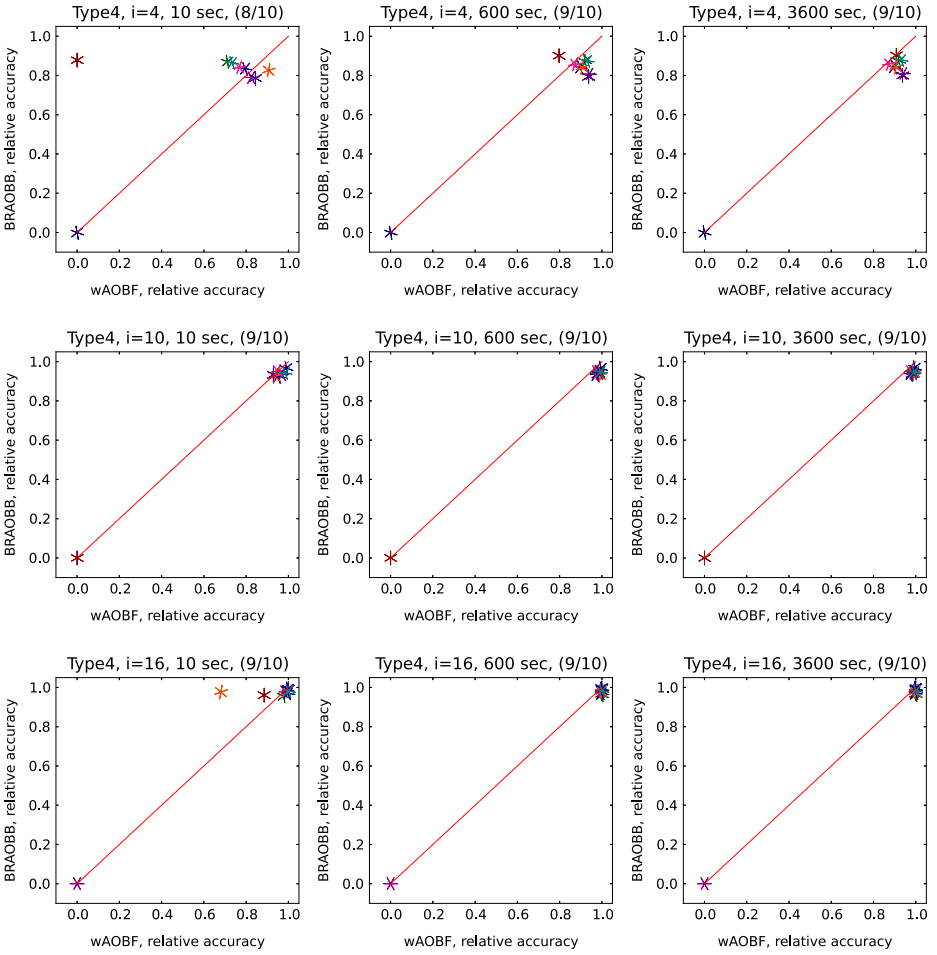


Fig. 23 wAOBF vs BRAOBB on Type4: comparison of relative accuracy at times 10, 600 and 3600 sec. Each row - a single time bound. Each marker is a single instance. Memory limit 4 GB, time limit 1 hour. In parenthesis (X/Y): X - # instances, for which at least one algorithm found a solution, Y - total # instances, MBE heuristic

Table 5 X% - percentage of instances for which each algorithm is better than BRAOBB at a specific time bound, Y% - percentage of instances for which algorithm ties with BRAOBB, N - number of instances for which at least one of algorithms found a solution

I-bound	Algorithm	Time bounds				
		10	30	60	600	3600
		X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N
		Grids (# inst=32, n=144-2500, k=2, w^*=15-90, h_T=48-283)				
i=6	wAOBF	66.7 / 12.5 / 24	64.0 / 16.0 / 25	68.0 / 20.0 / 25	40.0 / 40.0 / 25	16.0 / 44.0 / 25
	wR-AOBF	29.2 / 12.5 / 24	45.8 / 16.7 / 24	40.0 / 20.0 / 25	32.0 / 40.0 / 25	12.0 / 44.0 / 25
	wAOBB	33.3 / 12.5 / 24	40.0 / 12.0 / 25	44.0 / 12.0 / 25	32.0 / 32.0 / 25	28.0 / 44.0 / 25
i=10	wAOBF	58.3 / 20.8 / 24	56.0 / 32.0 / 25	46.2 / 42.3 / 26	22.2 / 55.6 / 27	14.8 / 55.6 / 27
	wR-AOBF	21.7 / 30.4 / 23	37.5 / 33.3 / 24	28.0 / 44.0 / 25	12.0 / 60.0 / 25	7.7 / 57.7 / 26
	wAOBB	54.2 / 20.8 / 24	44.0 / 24.0 / 25	23.1 / 30.8 / 26	11.1 / 44.4 / 27	18.5 / 55.6 / 27
i=14	wAOBF	42.3 / 42.3 / 26	38.5 / 46.2 / 26	30.8 / 61.5 / 26	14.8 / 74.1 / 27	3.4 / 69.0 / 29
	wR-AOBF	19.2 / 42.3 / 26	19.2 / 50.0 / 26	15.4 / 65.4 / 26	7.4 / 70.4 / 27	3.7 / 70.4 / 27
	wAOBB	19.2 / 42.3 / 26	19.2 / 46.2 / 26	7.7 / 57.7 / 26	11.1 / 66.7 / 27	10.3 / 75.9 / 29
i=18	wAOBF	14.3 / 57.1 / 21	16.7 / 66.7 / 24	16.7 / 66.7 / 24	3.7 / 77.8 / 27	0.0 / 75.0 / 28
	wR-AOBF	14.3 / 61.9 / 21	12.5 / 75.0 / 24	12.5 / 75.0 / 24	3.8 / 80.8 / 26	0.0 / 75.0 / 28
	wAOBB	4.8 / 61.9 / 21	8.3 / 58.3 / 24	16.7 / 62.5 / 24	7.4 / 77.8 / 27	3.6 / 82.1 / 28
		Pedigrees (# inst=11, n=581-1006, k=3-7, w^*=16-39, h_T=52-104)				
i=6	wAOBF	11.1 / 0.0 / 9	11.1 / 11.1 / 9	30.0 / 30.0 / 10	10.0 / 30.0 / 10	10.0 / 20.0 / 10
	wR-AOBF	11.1 / 11.1 / 9	11.1 / 22.2 / 9	22.2 / 22.2 / 9	11.1 / 22.2 / 9	11.1 / 22.2 / 9
	wAOBB	55.6 / 11.1 / 9	22.2 / 11.1 / 9	30.0 / 10.0 / 10	40.0 / 10.0 / 10	60.0 / 20.0 / 10
i=10	wAOBF	44.4 / 11.1 / 9	50.0 / 30.0 / 10	50.0 / 30.0 / 10	40.0 / 40.0 / 10	30.0 / 40.0 / 10
	wR-AOBF	25.0 / 12.5 / 8	44.4 / 22.2 / 9	40.0 / 30.0 / 10	45.5 / 27.3 / 11	27.3 / 27.3 / 11
	wAOBB	66.7 / 22.2 / 9	60.0 / 20.0 / 10	60.0 / 30.0 / 10	50.0 / 30.0 / 10	30.0 / 40.0 / 10
i=14	wAOBF	14.3 / 0.0 / 7	28.6 / 14.3 / 7	33.3 / 22.2 / 9	33.3 / 33.3 / 9	22.2 / 44.4 / 9
	wR-AOBF	14.3 / 0.0 / 7	14.3 / 14.3 / 7	22.2 / 22.2 / 9	10.0 / 30.0 / 10	10.0 / 30.0 / 10
	wAOBB	28.6 / 14.3 / 7	42.9 / 14.3 / 7	44.4 / 22.2 / 9	33.3 / 44.4 / 9	22.2 / 44.4 / 9
i=20	wAOBF	0 / 0 / 0	0 / 0 / 0	0.0 / 50.0 / 2	20.0 / 20.0 / 5	0.0 / 40.0 / 5
	wR-AOBF	0 / 0 / 0	0 / 0 / 0	0.0 / 0.0 / 2	0.0 / 20.0 / 5	0.0 / 20.0 / 5
	wAOBB	0 / 0 / 0	0 / 0 / 0	50.0 / 0.0 / 2	20.0 / 80.0 / 5	0.0 / 80.0 / 5

inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo tree height. 4 GB memory, 1 hour time limit, MBE heuristic

Table 6 X% - percentage of instances for which each algorithm is the better than BRAOBB at a specific time bound, Y% - percentage of instances for which algorithm ties with BRAOBB, N - number of instances for which at least one of algorithms found a solution

I-bound	Algorithm	Time bounds				
		10	30	60	600	3600
		X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N
		WCSP (# inst=56, n=25-1057, k=2-100, w^*=5-287, h_T=11-337)				
i=6	wAOBF	6.1 / 15.2 / 33	3.0 / 18.2 / 33	2.9 / 20.0 / 35	0.0 / 22.0 / 41	0.0 / 20.9 / 43
	wR-AOBF	6.1 / 15.2 / 33	3.0 / 18.2 / 33	2.9 / 20.0 / 35	0.0 / 20.0 / 40	0.0 / 19.0 / 42
	wAOBB	15.2 / 27.3 / 33	15.2 / 27.3 / 33	17.1 / 25.7 / 35	14.6 / 36.6 / 41	14.0 / 34.9 / 43
i=10	wAOBF	0.0 / 57.1 / 7	7.1 / 28.6 / 14	6.3 / 37.5 / 16	0.0 / 25.0 / 24	0.0 / 25.0 / 24
	wR-AOBF	0.0 / 57.1 / 7	7.1 / 28.6 / 14	6.3 / 37.5 / 16	0.0 / 25.0 / 24	0.0 / 25.0 / 24
	wAOBB	0.0 / 85.7 / 7	28.6 / 50.0 / 14	25.0 / 50.0 / 16	16.7 / 41.7 / 24	20.8 / 41.7 / 24
i=14	wAOBF	0.0 / 100.0 / 1	0.0 / 50.0 / 2	0.0 / 66.7 / 3	0.0 / 42.9 / 7	0.0 / 50.0 / 8
	wR-AOBF	0.0 / 100.0 / 1	0.0 / 50.0 / 2	0.0 / 66.7 / 3	0.0 / 42.9 / 7	0.0 / 50.0 / 8
	wAOBB	0.0 / 100.0 / 1	0.0 / 50.0 / 2	0.0 / 66.7 / 3	42.9 / 42.9 / 7	37.5 / 50.0 / 8
i=18	wAOBF	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 50.0 / 2	0.0 / 50.0 / 2
	wR-AOBF	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 50.0 / 2	0.0 / 50.0 / 2
	wAOBB	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 50.0 / 2	0.0 / 50.0 / 2
		Type4 (# inst=10, n=3907-8186, k=5, w^*=21-32, h_T=319-625)				
i=6	wAOBF	16.7 / 0.0 / 6	44.4 / 0.0 / 9	80.0 / 0.0 / 10	90.0 / 0.0 / 10	90.0 / 0.0 / 10
	wR-AOBF	16.7 / 0.0 / 6	33.3 / 0.0 / 9	30.0 / 0.0 / 10	50.0 / 0.0 / 10	50.0 / 0.0 / 10
	wAOBB	16.7 / 0.0 / 6	11.1 / 0.0 / 9	20.0 / 0.0 / 10	50.0 / 0.0 / 10	50.0 / 0.0 / 10
i=10	wAOBF	33.3 / 0.0 / 6	100.0 / 0.0 / 9	100.0 / 0.0 / 10	100.0 / 0.0 / 10	100.0 / 0.0 / 10
	wR-AOBF	0.0 / 0.0 / 6	44.4 / 0.0 / 9	50.0 / 0.0 / 10	60.0 / 0.0 / 10	60.0 / 0.0 / 10
	wAOBB	33.3 / 0.0 / 6	44.4 / 0.0 / 9	60.0 / 0.0 / 10	80.0 / 0.0 / 10	100.0 / 0.0 / 10
i=14	wAOBF	0 / 0 / 0	71.4 / 0.0 / 7	90.0 / 0.0 / 10	100.0 / 0.0 / 10	100.0 / 0.0 / 10
	wR-AOBF	0 / 0 / 0	57.1 / 0.0 / 7	60.0 / 0.0 / 10	80.0 / 0.0 / 10	60.0 / 0.0 / 10
	wAOBB	0 / 0 / 0	71.4 / 0.0 / 7	80.0 / 0.0 / 10	90.0 / 0.0 / 10	100.0 / 0.0 / 10
i=18	wAOBF	0 / 0 / 0	0 / 0 / 0	100.0 / 0.0 / 1	100.0 / 0.0 / 9	88.9 / 11.1 / 9
	wR-AOBF	0 / 0 / 0	0 / 0 / 0	100.0 / 0.0 / 1	55.6 / 0.0 / 9	44.4 / 11.1 / 9
	wAOBB	0 / 0 / 0	0 / 0 / 0	100.0 / 0.0 / 1	88.9 / 0.0 / 9	88.9 / 0.0 / 9

inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo tree height. 4 GB memory, 1 hour time limit, MBE heuristic

References

1. Bertele, U., Brioschi, F.: Nonserial dynamic programming. Academic (1972)
2. Cabon, B., De Givry, S., Verfaillie, G.: Anytime lower bounds for constraint violation minimization problems. In: International conference on principles and practice of constraint programming (CP), pp. 117–131 (1998)
3. Chakrabarti, P.P., Ghose, S., DeSarkar, S.: Admissibility of ao* when heuristics overestimate. Artif. Intell. **34**(1), 97–113 (1987)
4. Dechter, R.: Bucket elimination: a unifying framework for reasoning. Artif. Intell. **113**(1), 41–85 (1999)

5. Dechter, R., Mateescu, R.: AND/OR search spaces for graphical models. Artif. Intell. **171**(2–3), 73–106 (2007)
6. Dechter, R., Rish, I.: Mini-buckets: a general scheme for bounded inference. J. ACM **50**(2), 107–153 (2003)
7. Delisle, E., Bacchus, F.: Solving Weighted CSPs by successive relaxations. In: International conference on principles and practice of constraint programming (CP), pp. 273–281 (2013)
8. Fishelson, M., Geiger, D.: Exact genetic linkage computations for general pedigrees. In: International Conference on Intelligent Systems for Molecular Biology (ISMB), pp. 189–198 (2002)
9. Fontaine, M., Loudni, S., Boizumault, P.: Exploiting tree decomposition for guiding neighborhoods exploration for VNS. RAIRO - Oper. Res. **47**(02), 91–123 (2013)
10. Gerevini, A., Saetti, A., Vallati, M.: An automatically configurable portfolio-based planner with macro-actions: PbP. In: International conference on automated planning and scheduling (ICAPS), pp. 350–353 (2009)
11. Hansen, E., Zhou, R.: Anytime heuristic search. J. Artif. Intell. Res. **28**(1), 267–297 (2007)
12. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans Syst. Sci. Cybern. **4**(2), 100–107 (1968)
13. Hutter, F., Hoos, H.H., Stützle, T.: Efficient stochastic local search for MPE solving. In: International joint conference on artificial intelligence (IJCAI), pp. 169–174 (2005)
14. Ihler, A., Flerova, N., Dechter, R., Otten, L.: Join-graph based cost-shifting schemes. In: Uncertainty in artificial intelligence (UAI), pp. 397–406 (2012)
15. Kask, K., Dechter, R.: Branch and bound with mini-bucket heuristics. In: International joint conference on artificial intelligence (IJCAI), pp. 426–433 (1999a)
16. Kask, K., Dechter, R.: Mini-bucket heuristics for improved search. In: Uncertainty in artificial intelligence (UAI), pp. 314–323 (1999b)
17. Kask, K., Dechter, R.: Stochastic local search for Bayesian networks. In: Workshop on AI and statistics (AISTATS), pp. 113–122 (1999c)
18. Kask, K., Dechter, R., Larrosa, J., Dechter, A.: Unifying cluster-tree decompositions for automated reasoning. Artif. Intell. **166**(1–2), 165–193 (2005)
19. Kjærulff, U.: Triangulation of graphs–algorithms giving small total state space. Tech. rep., Department of Mathematics and Computer Science, Aalborg University, Denmark (1990)
20. Lawler, E., Wood, D.: Branch-and-bound methods: a survey. Oper. Res. **14**(4), 699–719 (1966)
21. Lecoutre, C., Roussel, O., Dehani, D.E.: WCSP integration of soft neighborhood substitutability. In: International conference on principles and practice of constraint programming (CP), pp. 406–421 (2012)
22. Likhachev, M., Gordon, G., Thrun, S.: ARA*: Anytime A* with provable bounds on sub-optimality. In: Neural information processing systems (NIPS), p. 16 (2003)
23. Marinescu, R., Dechter, R.: AND/OR branch-and-bound for graphical models. In: International joint conference on artificial intelligence (IJCAI), pp. 224–229 (2005)
24. Marinescu, R., Dechter, R.: Best-first AND/OR search for graphical models. In: National conference on artificial intelligence (AAAI), pp. 1171–1176 (2007)
25. Marinescu, R., Dechter, R.: AND/OR branch-and-bound search for combinatorial optimization in graphical models. Artif. Intell. **173**(16–17), 1457–1491 (2009a)
26. Marinescu, R., Dechter, R.: Memory intensive AND/OR search for combinatorial optimization in graphical models. Artificial Intelligence **173**(16–17), 1492–1524 (2009b)
27. Neveu, B., Trombettoni, G., Glover, F.: Id walk: a candidate list strategy with a simple diversification device. In: International conference on principles and practice of constraint programming (CP), pp. 423–437 (2004)
28. Nillson, N.J.: Principles of artificial intelligence. Tioga, Palo Alto (1980)
29. Otten, L., Dechter, R.: Anytime AND/OR depth first search for combinatorial optimization. In: International symposium on combinatorial search (SoCS), pp. 117–124 (2011)
30. Pearl, J.: Heuristics: intelligent search strategies. Addison-Wesley (1984)
31. Pohl, I.: Heuristic search viewed as path finding in a graph. Artif. Intell. **1**(3–4), 193–204 (1970)
32. Richter, S., Thayer, J., Ruml, W.: The joy of forgetting: faster anytime search via restarting. In: International conference on automated planning and scheduling (ICAPS), pp. 137–144 (2010)
33. Sharma, P., Flerova, N., Dechter, R.: Empirical evaluation of weighted heuristic search with advanced mini-bucket heuristics for graphical models. Tech. rep., University of California Irvine (2014). URL http://www.ics.uci.edu/dechter/publications/r214.pdf
34. Sontag, D., Choe, D.K., Li, Y.: Efficiently searching for frustrated cycles in MAP inference. In: Uncertainty in artificial intelligence (UAI), pp. 795–804 (2012)
35. Thayer, J., Ruml, W.: Anytime heuristic search: frameworks and algorithms. In: International symposium on combinatorial search (SoCS), pp. 121–128 (2010)

36. Van Den Berg, J., Shah, R., Huang, A., Goldberg, K.: ANA*: anytime nonparametric A*. In: Conference on artificial intelligence (AAAI), pp. 105–111 (2011)
37. Wang, H., Daphne, K.: Subproblem-tree calibration: a unified approach to max-product message passing. In: International conference on machine learning (ICML), pp. 190–198 (2013)
38. Wilt, C.M., Ruml, W.: When does weighted A* fail? In: International symposium on combinatorial search (SoCS), pp. 137–144 (2012)
39. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla: portfolio-based algorithm selection for SAT. J. Artif. Intell. Res. (JAIR) **32**(1), 565–606 (2008)