Noname manuscript No.
(will be inserted by the editor)

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

Natalia Flerova · **Radu Marinescu** · **Rina Dechter**

the date of receipt and acceptance should be inserted later

Abstract Weighted heuristic search (best-first or depth-first) refers to search with a heuristic function multiplied by a constant w [Pohl (1970)]. The paper shows for the first time that for graphical models optimization queries 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 depth-first branch and bound algorithms as approximation anytime schemes (that have suboptimality 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

1 Introduction

The idea of weighing the heuristic evaluation function by a fixed (or varying) constant when guiding search, is well known [Pohl (1970)]. 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,

Natalia Flerova
University of California Irvine, USA
E-mail: nflerova@uci.edu

Radu Marinescu
IBM Research – Ireland
E-mail: radu.marinescu@ie.ibm.com

Rina Dechter
University of California Irvine, USA
E-mail: dechter@uci.edu

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 suboptimality 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 lack of anytime behaviour, 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 path-finding domain, where solution length varies (e.g., planning), best-first search and especially its popular variant A* [Hart et al (1968)] 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*) [Pohl (1970)]. Several anytime weighted heuristic best-first search schemes were proposed in the context of path-finding problems in the past decade [Hansen and Zhou (2007); Likhachev et al (2003); Van Den Berg et al (2011); Richter et al (2010); Thayer and Ruml (2010); Richter et al (2010)].

Our contribution.

We extend and evaluate weighted heuristic search for graphical models. As a basis we used AND/OR Best First search (AOBF,[Marinescu and Dechter (2009b)]) and AND/OR Branch and Bound search (AOBB, [Marinescu and Dechter (2009a)]), both described in Section 2.3. We compare against a variant called Breadth-Rotating AND/OR Branch and Bound (BRAOBB [Otten and Dechter (2011)]). As already mentioned, BRAOBB (under the name daoopt) was instrumental in winning the 2011 Probabilistic Inference Challenge[1] in all optimization categories. This algorithm also won second place for 20 minutes and 1 hour time bounds in MAP category and first place for all time bounds in MMAP category in UAI Inference Challenge 2014[2]. We also compare our schemes against traditional depth-first branch and bound (DFBB) and A* exploring 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 decreasing w. wAOBF starts afresh at each iteration, while wR-AOBF reuses search efforts from previous iterations, extending ideas presented in Anytime Repairing A* (ARA*) [Likhachev et al (2003)]. 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 for depth-first search, resulting in wAOBB and wBRAOBB schemes. The weights facilitate an alternative anytime approach and most importantly equip those schemes with suboptimality guarantees. Our empirical evaluation showed that for many instances our algorithms yielded best results.

To explain the behaviour 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

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

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

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., [Hutter et al (2005); Neveu et al (2004); Fontaine et al (2013); Sontag et al (2012); Wang and Daphne (2013)] remain beyond the scope of our consideration, as do a number of exact methods, developed for weighted CSP problems, e.g., [Lecoutre et al (2012); Delisle and Bacchus (2013)]. 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 [Cabon et al (1998)], 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 includes the overview of methodology used (5.1), shows 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 [Nillson (1980)], 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 with 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 to go from n to a goal node $h^*(n)$. For a minimization task, $h(n)$ is *admissible* if $\forall n\ h(n) \leq h^*(n)$.

Weighted A Search* (WA*) [Pohl (1970)] 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 behaviour, 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 optimal solution's cost. Such solution is called w-optimal.

Formally, after [Pohl (1970)]:

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 [Pearl (1984)] 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 $h(t) = 0$, $C \leq w \cdot C^*$. $\square$

2.2 Graphical models

A *graphical model* is a tuple $\mathscr{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \bigotimes \rangle$, 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., $\forall i \; f_i : \mathbf{X}_{\mathbf{S}_i} \to \mathbb{R}^+$). 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. Figure 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., Figure 1b) and has a certain *induced width* w^* (not to be confused with weight w). For more details see [Kask et al (2005)]. The combination operator $\bigotimes \in \{\prod, \sum\}$ defines the complete function represented by the graphical model $\mathscr{M}$ as $\mathscr{C}(\mathbf{X}) = \bigotimes_{j=1}^{r} f_j(\mathbf{X}_{\mathbf{S}_j})$.

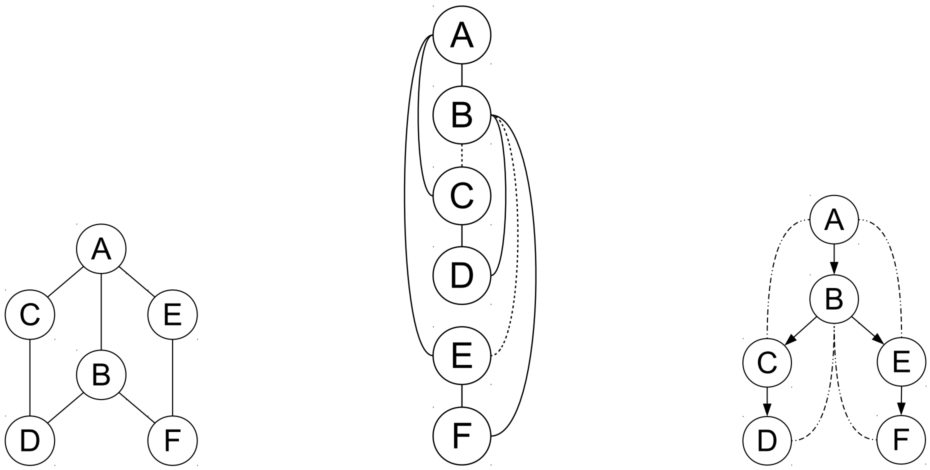
The most common optimization task is known as the *most probable explanation* (MPE) or *maximum a posteriori* (MAP), in which we would like to compute the optimal value C^* and/or its optimizing configuration x^*:

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

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

The MPE/MAP task is often converted into log-space and solved as an *energy minimization* (min-sum) problem. This is also known as the Weighted CSP (WCSP) problem [Marinescu and Dechter (2009a)] and is defined as follows:

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



(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 pseudotree, and resulting AND/OR search graph with AOBB pruning example.

Bucket Elimination (BE) [Dechter (1999), Bertele and Brioschi (1972)] 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) [Dechter and Rish (2003)] 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 parametrized 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 [Kask and Dechter (1999a), Kask and Dechter (1999b)].

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 [Dechter and Mateescu (2007)]. A *pseudo tree* of the primal graph defines the search space and captures problem decomposition (e.g., Figure 1c).

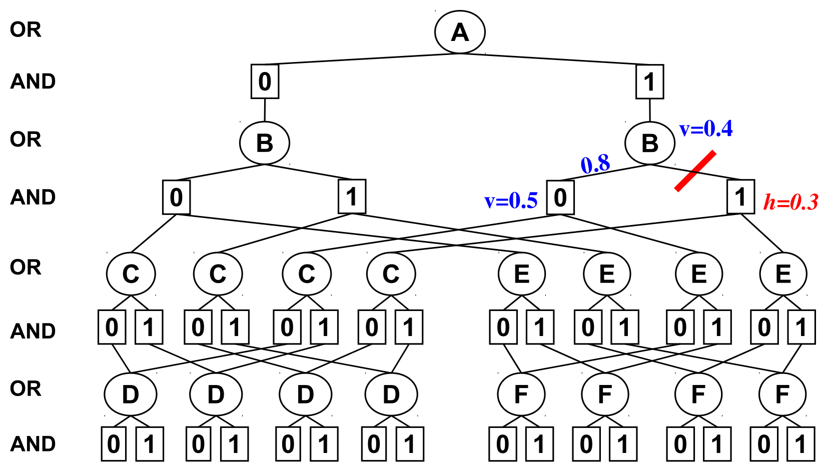


Fig. 2 Context-minimal AND/OR search graph with AOBB pruning example.

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

Given a graphical model $\mathscr{M} = (\mathbf{X},\mathbf{D},\mathbf{F})$ with primal graph G and a pseudo tree $\mathscr{T}$ of G, the *AND/OR search tree* $S_{\mathscr{T}}$ based on $\mathscr{T}$ has alternating levels of OR and AND nodes. Its structure is based on the underlying pseudo tree. The root node of $S_{\mathscr{T}}$ is an OR node labelled by the root of $\mathscr{T}$. 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 $\mathscr{T}$, 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* [Dechter and Mateescu (2007)]. Merging all context-mergeable nodes yields the *context minimal AND/OR search graph*, denoted by $C_{\mathscr{T}}$ (e.g., Figure 2). The size of the context minimal AND/OR graph is exponential in the induced width of G along a depth-first traversal of $\mathscr{T}$ [Dechter and Mateescu (2007)].

A *solution tree* T of $C_{\mathscr{T}}$ is a subtree such that: (1) it contains the root node of $C_{\mathscr{T}}$; (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., nodes 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 $C_{\mathscr{T}}$ 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 by minimization and summation, for OR and AND nodes, respectively [Dechter and Mateescu (2007)].

We next provide an overview 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 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 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* [Nillson (1980)] that explores the AND/OR context-minimal search graph. It was developed by [Marinescu and Dechter (2009a)].

AOBF, described by Algorithm 1 for min-sum task, maintains the explicated part of the context minimal AND/OR search graph $\mathscr{G}$ 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 explored search graph $\mathscr{G}$, with a bottom-up cost revision step (lines 16-25), 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 bottom-up phase OR nodes that have at least one solved child and 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 19). Following the backward step, a new best partial solution tree T^* is recomputed (line 25). 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 AOBF does not maintain explicit OPEN and CLOSED lists as traditional OR search schemes do. Same is true for all AND/OR algorithms discussed in this paper.

Theorem 2 (Marinescu and Dechter (2009b)) *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. 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.*

AND/OR Branch and Bound (AOBB). The AND/OR Branch and Bound [Marinescu and Dechter (2009a)] 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 4-17) with a backward cost revision (or propagation) step (lines 19-29) 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 higher than 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.

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 1: AOBF(h_i, w_0) exploring 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 $\mathscr{M}$
create root OR node s labelled by X_1 and let $\mathscr{G}$ (explored search graph) $= \{s\}$;
initialize $v(s) = w_0 \cdot h_i(s)$ and best partial solution tree T^* to $\mathscr{G}$;
while *s is not SOLVED* **do**
    select non-terminal tip node n in T^*. If there is no such node then **exit**;
    // expand node n
    if $n = X_i$ *is OR* **then**
        forall the $x_i \in D(X_i)$ **do**
            create AND child $n' = \langle X_i, x_i \rangle$;
            if *n' is TERMINAL* **then**
                mark n' SOLVED;
            $succ(n) \leftarrow succ(n) \cup n'$;
    else if $n = \langle X_i, x_i \rangle$ *is AND* **then**
        forall the *successor* X_j *of* X_i *in* $\mathscr{T}$ **do**
            create OR child $n' = X_j$;
            $succ(n) \leftarrow succ(n) \cup n'$;
    initialize $v(n') = w_0 \cdot h_i(n')$ for all new nodes;
    add new nodes to the explores search space graph $\mathscr{G} \leftarrow \mathscr{G} \cup succ(n)$;
    // update n and its AND and OR ancestors in $\mathscr{G}$, bottom-up
    repeat
        if *n is OR node* **then**
            $v(n) = \min_{k \in succ(n)}(c(n,k) + v(k))$;
            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);
            mark n as SOLVED if its best marked successor is solved;
        else if *n is AND node* **then**
            $v(n) = \sum_{k \in succ(n)} v(k)$;
            mark all arcs to the successors;
            mark n as SOLVED if all its children are SOLVED;
        $n \leftarrow p$; //p is a parent of n in $\mathscr{G}$
    until *n is not root node s*;
    recompute T^* by following marked arcs from the root s;
return $\langle v(s), T^* \rangle$;

Theorem 3 (Marinescu and Dechter (2009b)) *The AND/OR Branch and Bound search algorithm with no caching(traversing an AND/OR search tree has worst case time and space complexity of $O(n \cdot k^h)$, 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.*

Breadth Rotating AND/OR Branch and Bound (BRAOBB). To remedy the relatively poor anytime behaviour of AOBB, the *Breadth-Rotating AND/OR Branch and Bound* algorithm has been introduced recently by [Otten and Dechter (2011)]. The basic idea is to rotate through different subproblems in a breadth-first manner. Empirically, BRAOBB finds the first suboptimal solution significantly faster than plain AOBB [Otten and Dechter (2011)].

Algorithm 2: AOBB(h_i, w_0, $UB = \infty$) exploring 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;
Output: Optimal solution to $\mathscr{M}$

create root OR node s labelled by X_1 and let stack of created but not expanded nodes OPEN $= \{s\}$;
initialize $v(s) = \infty$ and best partial solution tree rooted in s $T^*(s) = \emptyset$; $UB = \infty$;
while *OPEN* $\neq \emptyset$ **do**
  select top node n on OPEN.
  //EXPAND
  if *n is OR node labelled X_i* **then**
    foreach $x_i \in D(X_i)$ **do**
      //Expand node n:
      add AND child $n' = \langle X_i, x_i \rangle$ to list of successors of n;
      initialize $v(n') = 0$, best partial solution tree rooted in n $T^*(n') = \emptyset$;
  if *n is AND node labelled $\langle X_i, x_i \rangle$* **then**
    foreach *OR ancestor k of n* **do**
      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
      if *evaluated partial solution is not better than current upper bound at k (e.g. $f(k) \geq v(k)$ for minimization* **then**
        prune the subtree below the current tip node n;
      else
        foreach *successor X_j of $X_i \in \mathscr{T}$* **do**
          add OR child $n' = X_j$ to list of successors of n;
          initialize $v(n') = \infty$, best partial solution tree rooted in n $T^*(n') = \emptyset$;
  add successors of n on top of OPEN;
  //PROPAGATE
  //Only propagate if all children are evaluated and the final v are determined
  while *list of successors of node n is empty* **do**
    if *node n is the root node* **then**
      return *solution: $v(n), T^*(n)$;*
    else
      //update ancestors of n, AND and OR nodes p, bottom up:
      if *p is AND node* **then**
        $v(p) = v(p) + v(n)$, $T^*(p) = T^*(p) \cup T^*(n)$;
      else if *p is OR node* **then**
        if *the new value of better than the old one, e.g. $v(p) > (c(p,n) + v(n))$ for minimization* **then**
          $v(p) = c(p,n) + v(n)$, $T^*(p) = T^*(p) \cup \langle x_i, X_i \rangle$;
    remove n from the list of successors of p;
    move one level up: $n \leftarrow p$;

3 Some properties of weighted heuristic search

In this section we explore the interplay between the weight w and 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 on. In general, however, such behaviour is not guaranteed [Wilt and Ruml (2012)]. 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 explore the subtree rooted at the current node representing a partial solution path. This leads us to the following definition.

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 $f(T_n')$;
if $succ(n) == \emptyset$ **then**
  return $h(n) \cdot w$;
else
  if *n is an AND node* **then**
    let $k_1, \ldots, k_l$ be the OR children of n;
    return $\sum_{i=1}^{l}$ `evalPartialSolutionTree`(T_{k_i}', $h(k_i)$, w);
  else if *n is an OR node* **then**
    let k be the AND child of n;
    return $w(n,k)$ + `evalPartialSolutionTree`(T_k', $h(k)$, w);

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'' belong to the subgraph rooted at n , and c) under the weaker condition that $f(n) \geq f(n')$, parts a) and b) still holds 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 (which after expansion of n include 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 favour 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, then the search space is focused along path π.*

We will next show that this focused search property can be 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 π in a search graph and a heuristic function h such that $w_h(\pi)$ is well defined, then, if $w > w_h(\pi)$, 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 $w > w_h(\pi)$ for each solution path π, then WA* performs a greedy search, assuming ties are broken in favour of deeper nodes.*

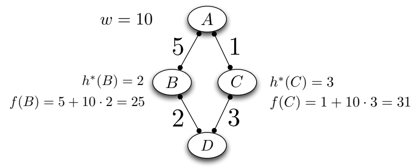


Fig. 3 WA* with the exact heuristic

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.*

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 Figure 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 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 one arc at least $c(n,n') > 0$*
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.

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 2 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 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 worse case the bound may be loose.

It is interesting to note that,

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

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

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

□

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 analysing a number of existing weighted 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 AOBF

The fixed-weighted 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 [Chakrabarti et al (1987)], but it is adapted to the specifics of AOBF. Clearly, if $h_i(n)$ is admissible, which is the case for mini-bucket

Algorithm 4: wAOBF(w_0, h_i)

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

heuristics, the cost of the solution discovered by weighted AOBF is w-optimal, same as is known for WA* [Pohl (1970)] and WAO* [Chakrabarti et al (1987)].

Consider an example problem with 4 binary variables in Figure 4 that we will use to illustrate the work of our algorithms. Figure 4a shows the primal graph. We assume 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 Figure 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 AOBF with weight $w = 10$ (on the right). The numbers in 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 AOBF discovers a suboptimal 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.

4.2 Iterative Weighted AOBF (wAOBF)

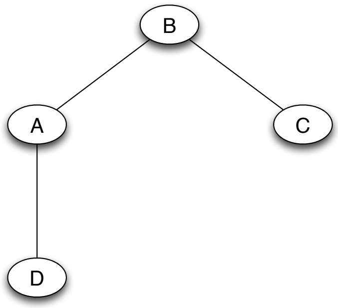
Since Weighted heuristic AOBF yields w-optimal solutions, it can be extended to an anytime scheme **wAOBF** (Algorithm 4), decreasing the weight from one iteration to the next. This approach is identical to the Restarting Weighted A* by Richter et al (2010) applied to AOBF.

Theorem 7 *Worst case time and space complexity of a single iteration of wAOBF 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 problem.*

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 weight decreasing policy. □

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*



(a) Primal graph.

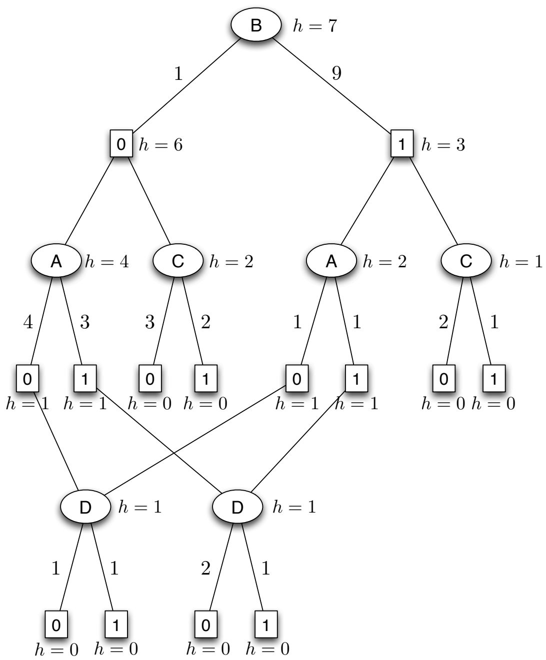
$f(A,B)$	A	B
4	0	0
1	0	1
3	1	0
1	1	1

$f(B,C)$	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.

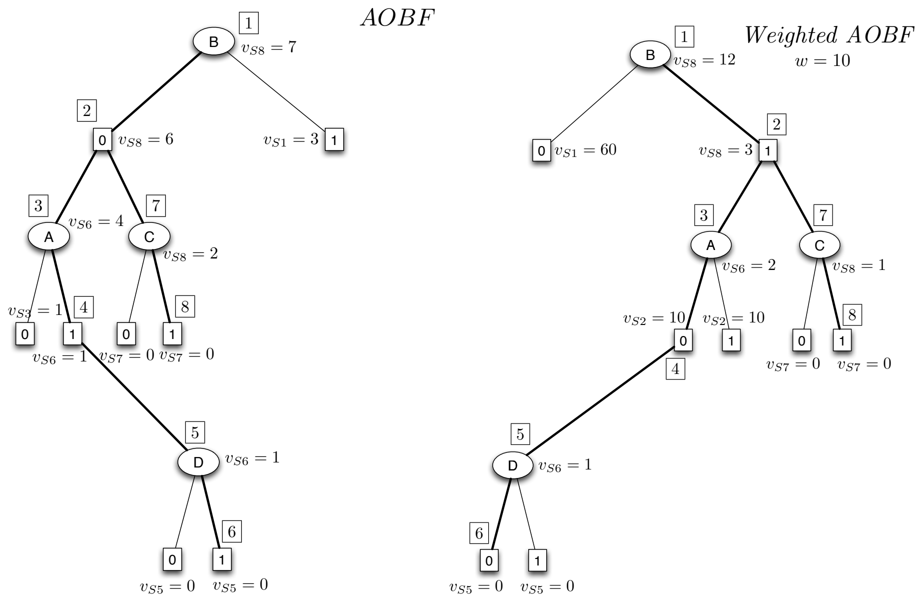


(c) Context-minimal weighted AND/OR graph.

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

Repairing A (ARA*)* algorithm [Likhachev et al (2003)] 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



(a) (b)

Fig. 5 Search graphs explored by (a) AOBF, (b) Weighted AOBF, w=10. Boxed numbers 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.

Algorithm 5: wR-AOBF(h_i, w_0)

Input: A graphical model $\mathscr{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$; pseudo tree $\mathscr{T}$ rooted at X_1; heuristic h_i for i-bound=i; initial weight w_0, weight update schedule S

Output: Set of suboptimal 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^* to $\mathscr{G}$;
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** $\mathscr{C}$;

---

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 9).

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 *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 problem. The worst case time complexity can be loosely bounded by $O(2 \cdot n \cdot k^{w^*})$. Total number of iteration depends on chosen starting weight parameters.*

Proof In 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 required book-keeping.

The time complexity comprises of time it takes to expand nodes and 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, though 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. 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 behaviour 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 [Otten and Dechter (2011)], 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 [Hart et al (1968)] and depth-first branch and bound (DFBB) [Lawler and Wood (1966)], both exploring an OR search tree, and against Stochastic Local Search (SLS) [Hutter et al (2005); Kask and Dechter (1999c)]. 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 variable ordering, obtained using well-known MinFill ordering heuristic [Kjærulff (1990)]. 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.

All schemes use the Mini-Bucket Elimination heuristic, described in Section 2.3, with 10 i-bounds, ranging from 2 to 20. However, for some hard problems computing 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.

[3] http://www.cs.huji.ac.il/project/PASCAL/realBoard.php
[4] http://www.hlt.utdallas.edu/ vgogate/uai14-competition/leaders.html

Benchmark	# inst	n	k	w^*	h_T
Pedigrees	11	581-1006	3-7	16-39	52-104
Grids	32	144-2500	2-2	15-90	48-283
WCSP	56	25-1057	2-100	5-287	11-337
Type4	10	3907-8186	5-5	21-32	319-625

Table 1 Benchmark parameters: # inst - number of instances, n - number of variables, k - domain size, w^* - induced width, h_T - pseudotree height.

We evaluated the algorithms on 4 benchmarks: Weighted CSPs (WCSP), Binary grids, Pedigree networks and Type4 genetic networks. The **pedigrees 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 [Fishelson and Geiger (2002)]. 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** 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 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 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, analyse the choice of weight decreasing policy, and study the anytime behaviour of the schemes and the interaction between heuristic strength and the weight.

5.2 The impact of weights on the weighted 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 behaviour prevalent over each benchmark. Following

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

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	—

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 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.

the names and parameters of each instance, the table is vertically split into two blocks, corresponding to two i-bounds. In the second 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, $i = 10$).

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 run time 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=16, for $w = 2.828$ weighted AOBF's runtime 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=10, BRAOBB takes 54.34 seconds and weighted AOBF - 54.88. On some WCSP instances, such as myciel5g_3.wcsp, i=2, BRAOBB is clearly superior, finding an optimal solution within the time limit, while weighted 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 behaviour. For example, for grid 75-18-5, i=6, we see that 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 (strong 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=16 weighted AOBF runtime for $w = 1.033$ is 34.66 seconds as opposed to 1082.02 seconds by BRAOBB. Observe that often the actual results are far more accurate than the bound suggests. In particular, in a few of the cases, the optimal solution is obtained with $w > 1$. For example, see grid 75-18-5, i-bound=20, $w = 1.003$. Sometimes exact AOBF with $w = 1$ is faster than BRAOBB.

Impact of heuristic strength. The i-bound parameter allows to flexibly control the strength of 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 Mini-Bucket Elimination algorithm is exponential in the i-bound. Thus we are interested to understand how the heuristic strength influences the behaviour 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 more accurate heuristic comes at too high a price. For example, for pedigree37 weighted 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=16. One of the examples to the contrary, where the higher i-bound is beneficial, is grid 90-21-5, where weighted 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 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, 2 orders of magnitude more. On the other hand, for the same instance weighted AOBF with weight $w = 1.033$ has much smaller difference in performance for the two i-bounds. weighted AOBF terminates in 17.65 seconds for $i = 20$ and in 41.38 seconds for $i = 6$. This may suggest that wAOBF could be preferable when the i-bound is small relative to the problem's induced width.

Overall, weighted AOBF solves some hard problems that are infeasible for the exact scheme and often yields solutions with 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., Hansen and Zhou (2007), Likhachev et al (2003). 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 behaviour, since the weight rapidly approaches 1.0 and much fewer intermediate solutions are found. This could potentially allow the schemes to produce the exact solution fast, 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 a) to explore the schemes' behaviour on a large range of weights; b) to make the search focused enough initially to solve harder instances, known to be infeasible for regular BF 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 behaviour 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.

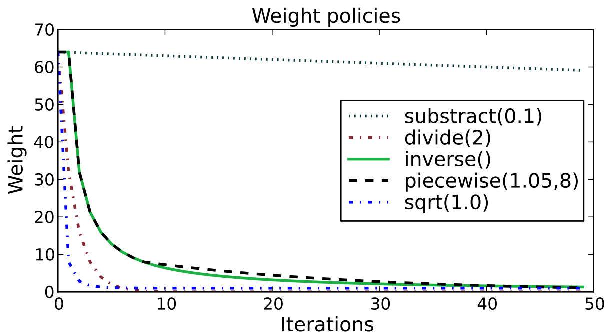


Fig. 6 The dependency of the weight value on iteration index according to considered weight policies, showing first 50 iterations, starting weight w_0= 64.

Figure 6 illustrates the weight changes during the first 50 iterations according to the considered policies. We use the parameter values that proved to be more 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 an i-bound from mid-range, 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 performances of 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, only the ones that yielded the best performance are presented. The starting weight is 64 and $w!$ denotes the weight at the time of algorithms termination. The behaviour depicted here was quite typical across instances and i-bounds. In this set of experiments the memory limit was 2 GB, with time limit of 1 hour and MBE heuristic was used.

We observe in Figure 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 coloured graph online). This can be seen, for example, on grid instance 75-23-5 and on type4b_120_17. 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 in 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 instance 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.

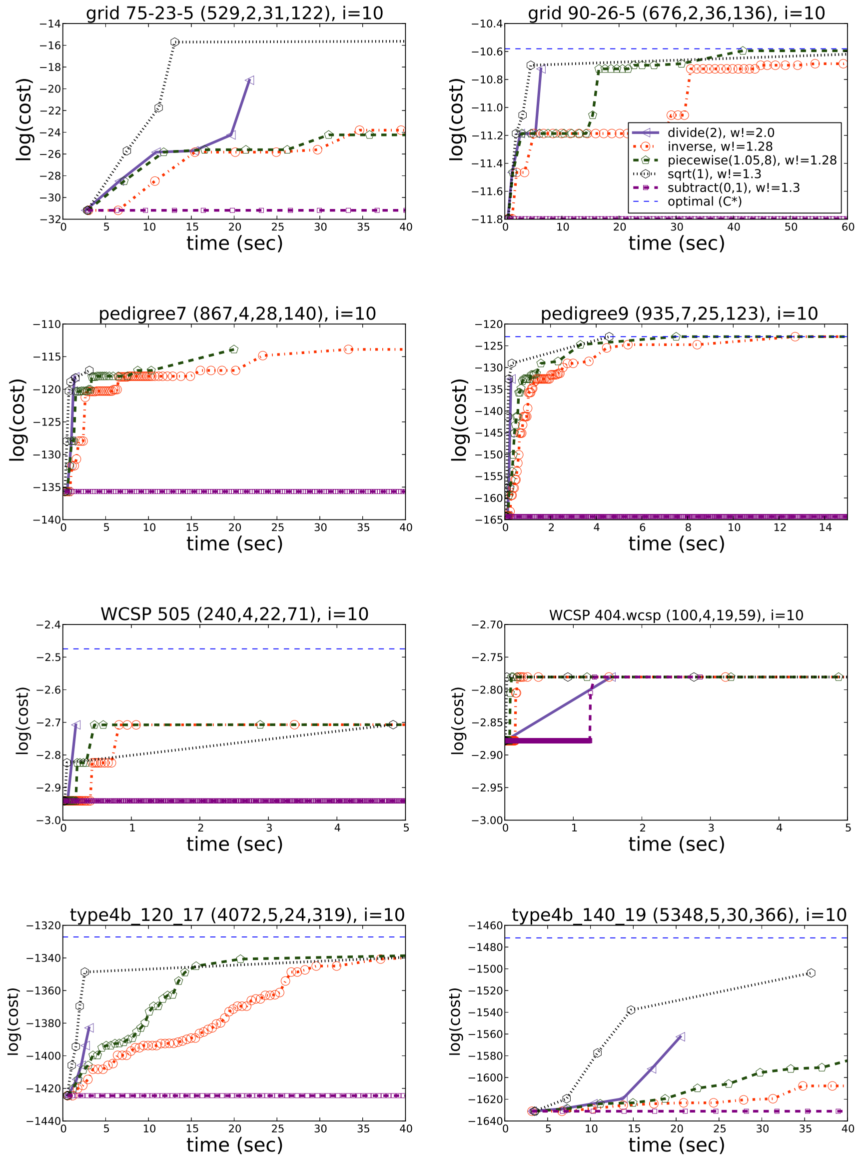


Fig. 7 wAOBF: solution cost (on logarithmic scale) vs time (sec) for different weight policies, starting weight = 64. Instance parameters are in format (n,k,w^*,h_T), where n - number of variables, k - max. domain size, w^* - induced width, h_T - pseudo-tree height. Time limit - 1 hour, memory limit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

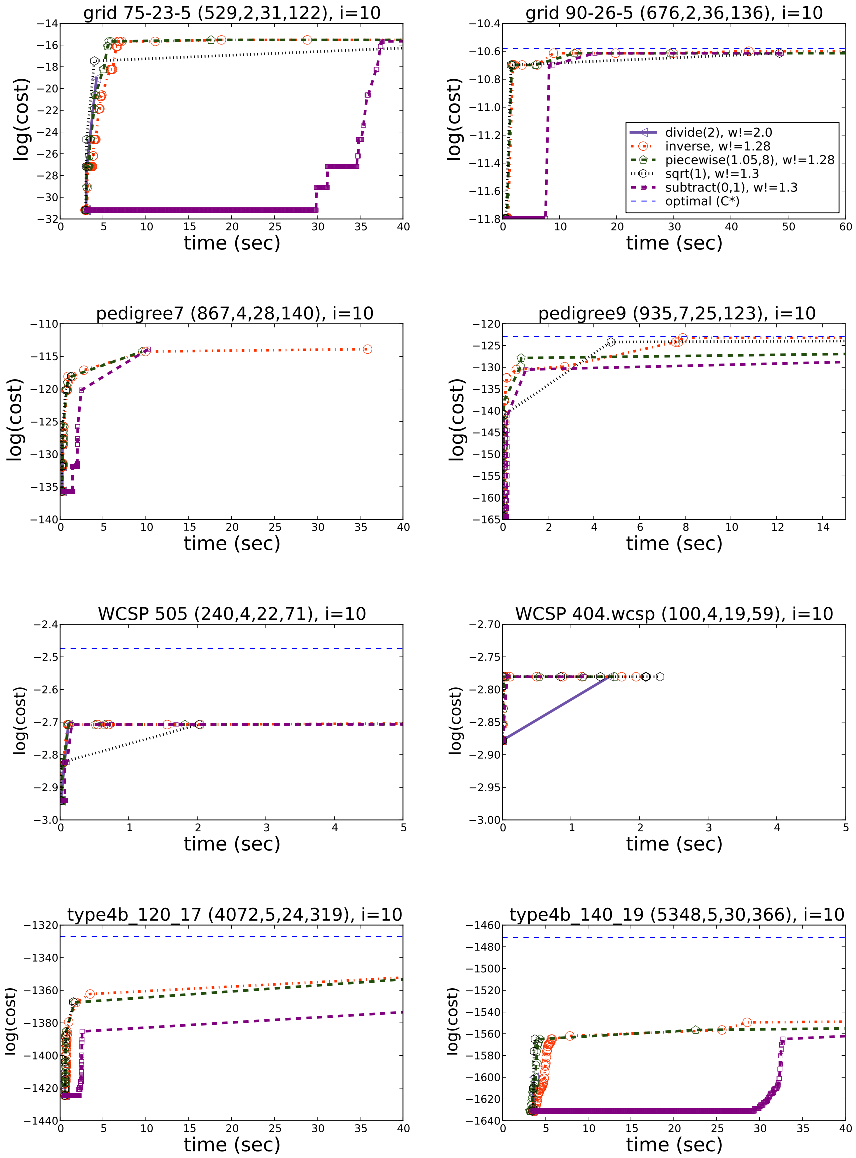


Fig. 8 wR-AOBF: solution cost (on logarithmic scale) vs time (sec) for different weight policies, starting weight = 64. Instance parameters are in format (n,k,w^*,h_T), where n - number of variables, k - max. domain size, w^* - induced width, h_T - pseudo-tree height. Time limit - 1 hour, memory limit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

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.

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=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=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=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=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

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

5.4 Anytime behaviour 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 of OR search tree (DFBB) and Stochastic Local Search (SLS). We ran each scheme on all instances from the same 4 benchmarks with MBE heuristic using the i-bound ranging from 2 to 20. We recorded the solutions at different time points, up until either the optimal solution was found or until the algorithm ran out of

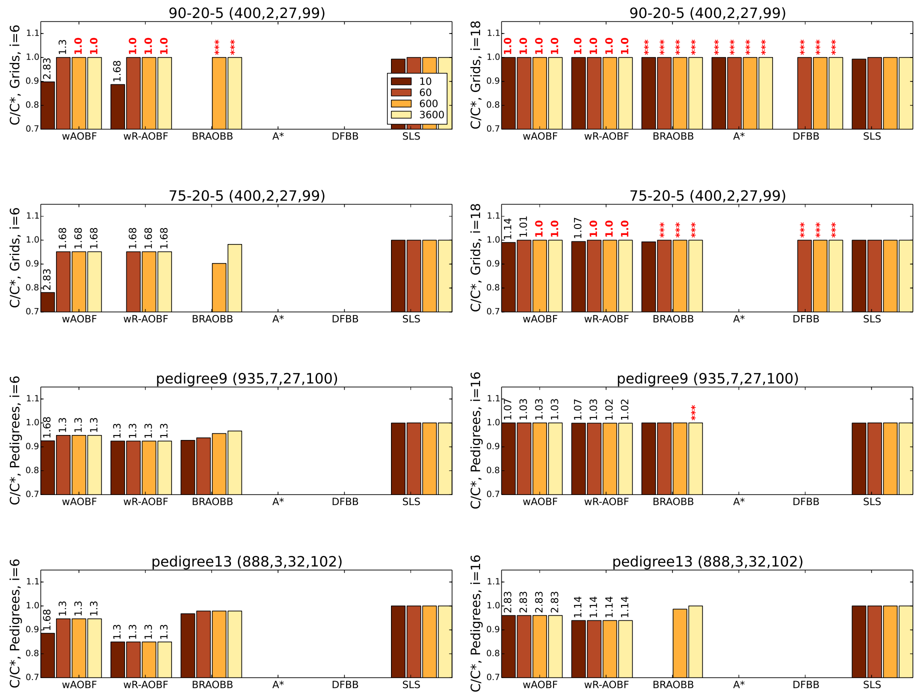


Fig. 9 Ratio of the cost obtained by some time point (10, 60, 600 and 3600 sec) and max cost (optimal, if known, otherwise - best cost found for the problem by any of the schemes). Corresponding weight - above the bars. '***' indicated proven solution optimality. Instance parameters are in format (n,k,w^*,h_T), 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.

4 GB of memory or the time cut off of 3600 seconds 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 the Table 3 and in bar charts in Figures 9 and 10. Then we provide summaries over all instances using bar charts in Figures 11-14 and scatterplots in Figure 15.

5.4.1 wAOBF vs wR-AOBF

Table 3 shows solution cost and corresponding weight by wAOBF and wR-AOBF for 2 selected instances from each benchmark, for medium i-bound, for several time 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 type4b_190_20, for 600 or 3600 seconds.

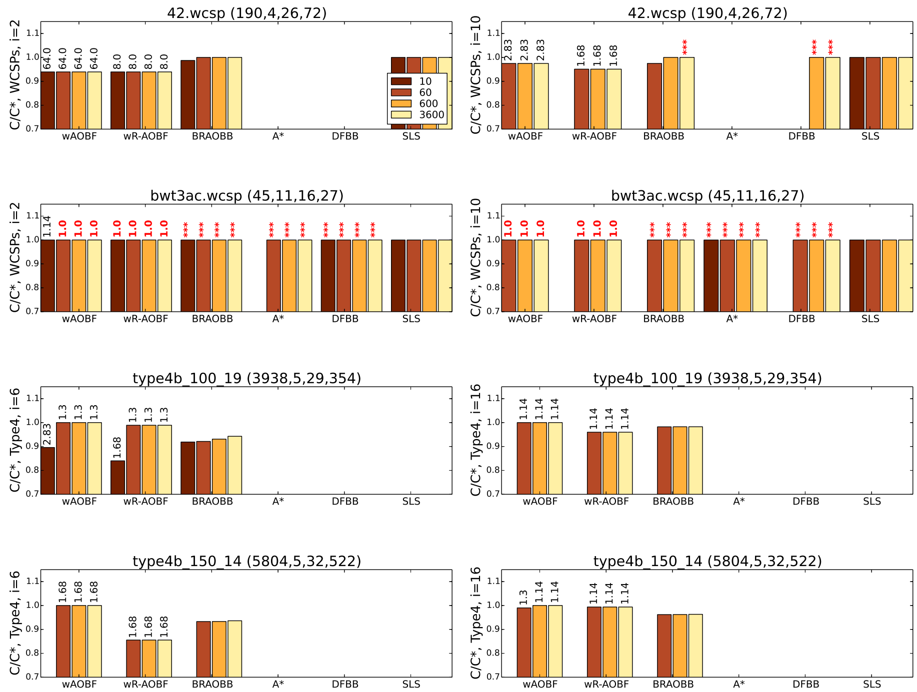


Fig. 10 Ratio of the cost obtained by some time point (10, 60, 600 and 3600 sec) and max cost (optimal, if known, otherwise - best cost found for the problem by any of the schemes). Corresponding weight - above the bars. '***' indicated proven solution optimality. Instance parameters are in format (n,k,w^*,h_T), 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.

Figures 9 and **10** present the anytime performance of wAOBF, wR-AOBF, BRAOBB, A*, DFBB and SLS for representative instances using bar charts. Figure 9 shows results for Grids and Pedigrees, Figure 10 - for WCSPs and Type4. 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 the weight corresponding to the returned solution, where $w = 1$ is shown in red. The symbol '***' above bars indicate proven solution optimality.

In these figures we again observe that wR-AOBF has worse anytime behaviour than wAOBF. For example, in Figure 9 for pedigree9 and pedigree13, i=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 already expanded nodes and to update their evaluation function 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 schemes compared against several state of the art anytime algorithms. We observe that the domineering scheme varies from benchmark to benchmark. In **Figure 9** we see that wAOBF is superior on the Grid instances when the heuristic is weak. For example, for grid 75-20-5, $i = 6$ 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 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 any solution for a small number of instances and only for highly accurate heuristics, e.g. 90-20-5, i=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=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 many of the Pedigrees. A* and DFBB are usually inferior even when the i-bound is high, e.g. pedigree13, i=16.

Figure 10 shows that the weighted heuristic best-first schemes are superior to all other schemes for Type4 instances shown, for all levels of heuristic strengths. Interestingly, on most of 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 schemes provide tight w-optimality guarantees in many cases. For example, on the two Type4 instances presented in Figure 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. On WCSPs, however, wAOBF and wR-AOBF are less effective. For example, both are inferior to BRAOBB and SLS on 42.wcsp instance, for both 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, Marinescu and Dechter (2005, 2007) for more comparisons between AND/OR and OR search). Additionally, A* is not an inherently anytime. Stochastic Local Search, though often finds accurate solutions fast, is not a complete algorithm and thus is outside 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 BROABB

We now show data summarizing the results from all the instances. **Figures 11-14** provide summaries of our experimental results in the form of bar charts. Note that the summaries include also a weighted 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 Figures 11-14 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 our benchmarks the weighted heuristic schemes dominated, finding costs of better accuracy within the time limits. This superior behaviour on Grids and Type4 benchmarks was mostly consistent across time bounds and heuristics of various strengths and can be observed in Figures 11 and 13. Specifically, for certain i-bounds and time bounds wAOBF returned more accurate costs than BRAOBB on up to 68% of the Grid instances and on 90-100% of the Type4 instances. Interestingly, for Type4 benchmark the weighted schemes almost never found the same solution costs as BRAOBB, as is indicated by the zeros in the plots on the right side of Figure 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 Type4 benchmark and for small i-bound, e.g. for i=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 more successful of the two weighted heuristic best-first schemes, wAOBF, and BRAOBB. Each plot corresponds to a specific time point. The x-axis coordinate gives the relative accuracy of the solution obtained by wAOBF, the y-axis coordinate - the relative accuracy of BRAOBB. As for the bar charts, the accuracy is defined relative to the optimal cost, if known, or the maximum cost available. 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 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 weak heuristic because it yields greater diversity in solution accuracy by wAOBF and BRAOBB. Typically, for strong heuristics the algorithms yields solutions having similar costs. Appendix B includes additional scatter diagrams showing results for 3 i-bounds and 3 time bounds per benchmark, along with scatter plots comparing wAOBF and wR-AOBF.

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

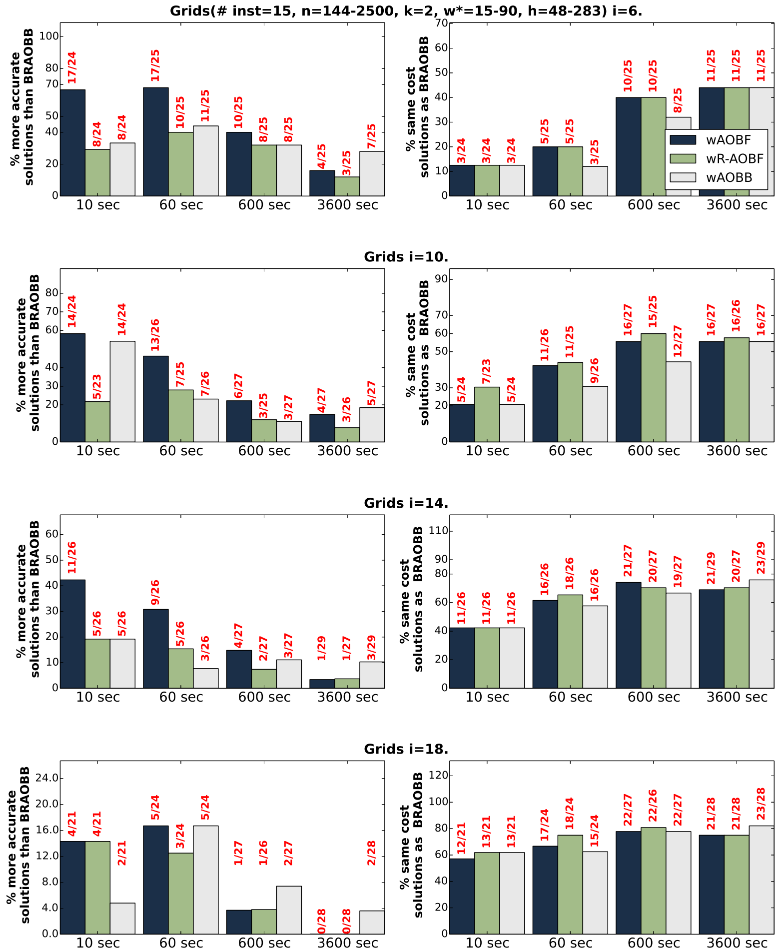


Fig. 11 Grids: percentage of instances for which each algorithm found solutions of more accurate cost than BRAOBB at a specific time bound (left), percentage of instances for which algorithm is tied with BRAOBB, i.e. found solution of equal cost (right). Above bars - number of instances where algorithm 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.

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 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 larger and more effective pruning of the space.

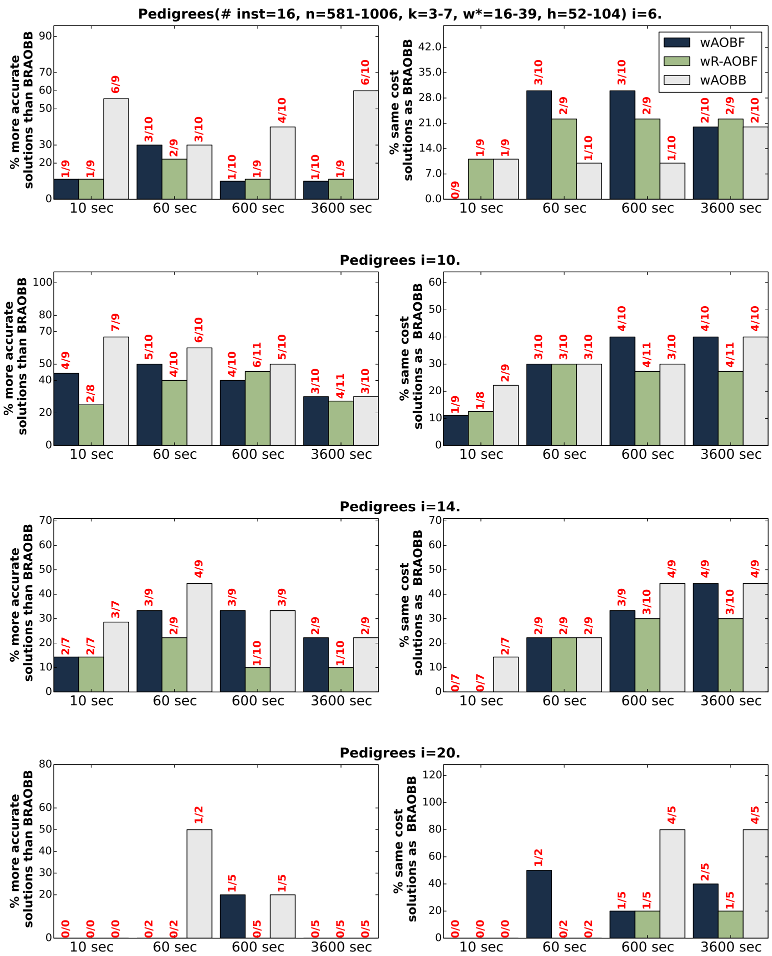


Fig. 12 Pedigrees: percentage of instances for which each algorithm found solutions of more accurate cost than BRAOBB at a specific time bound (left), percentage of instances for which algorithm is tied with BRAOBB, i.e. found solution of equal cost (right). Above bars - number of instances where algorithm 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.

Therefore, in this section we extend the depth-first branch and bound algorithms AOBB and BRAOBB to weighted 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 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 version of AOBB that uses the mini-bucket heuristic h_i having i-bound=i, multi-

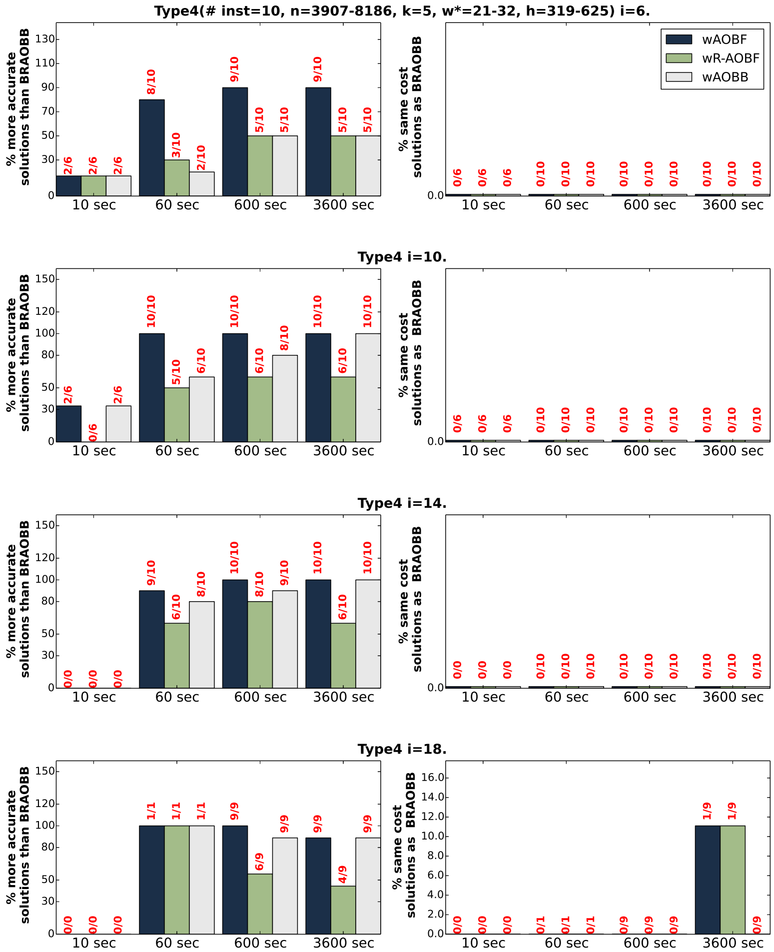


Fig. 13 Type4: percentage of instances for which each algorithm found solutions of more accurate cost than BRAOBB at a specific time bound (left), percentage of instances for which algorithm is tied with BRAOBB, i.e. found solution of equal cost (right). Above bars - number of instances where algorithm 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.

plied 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

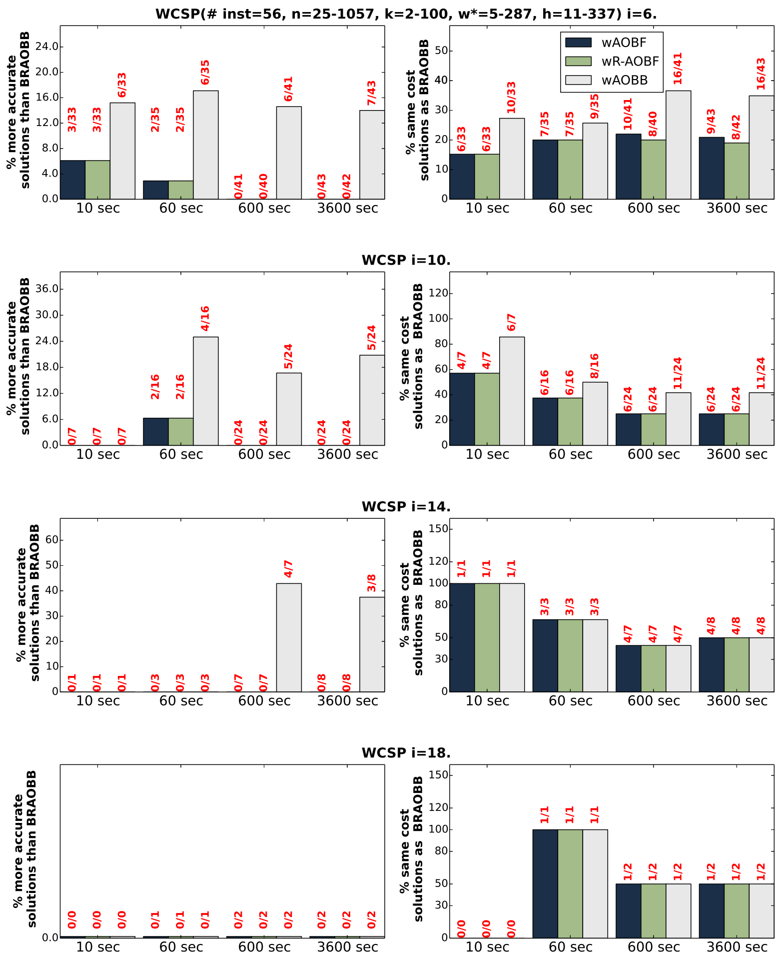


Fig. 14 WCSP: percentage of instances for which each algorithm found solutions of more accurate cost than BRAOBB at a specific time bound (left), percentage of instances for which algorithm is tied with BRAOBB, i.e. found solution of equal cost (right). Above bars - number of instances where algorithm 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.

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 pruned relative to the weighted evaluation function). Therefore,

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

Therefore (as $w \geq 1$) and from Equation 6

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

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

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

□

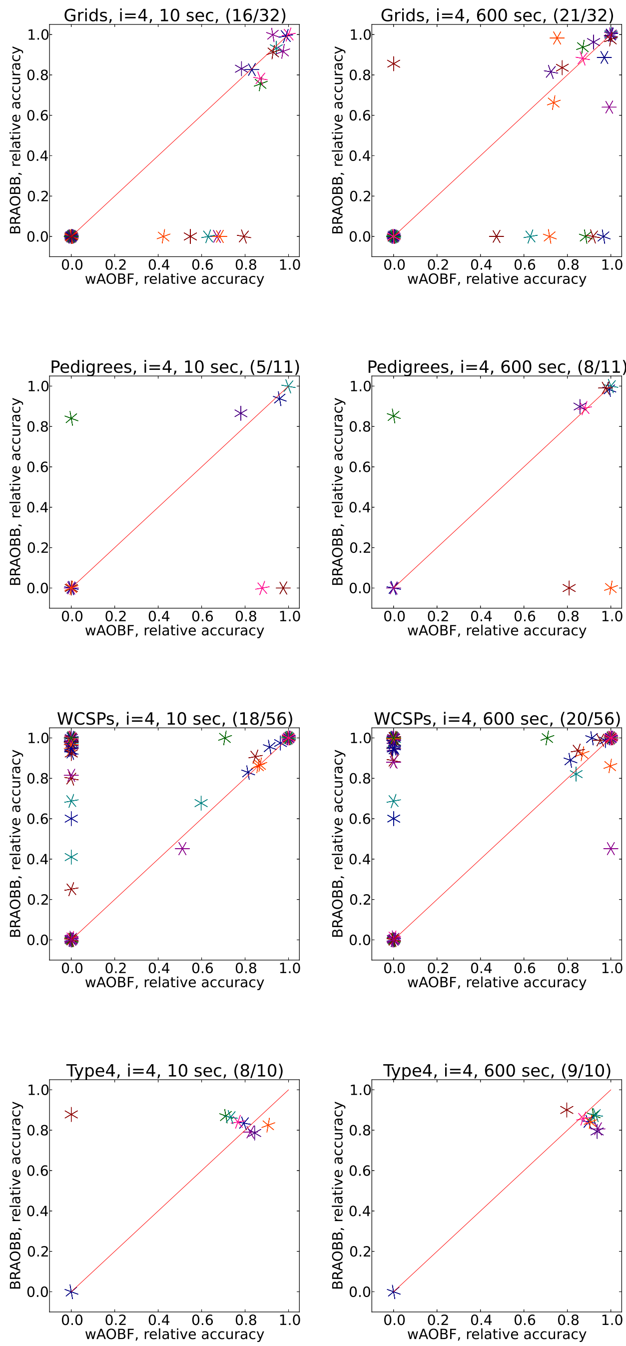


Fig. 15 wAOBF vs BRAOBB: comparison of relative accuracy at times 10, 3600 sec. Each row - a single time bound. Each marker represents a single instance. Memory limit 4 GB, time limit 1 hour. In parenthesis (X/Y): X - # instances, for which at least one algorithm found a solution, Y - total # instances,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

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). However, there are some inherent differences between these two schemes and wAOBF, explained next.

Iterative Weighted 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 cost of the solution of AOBB(h_i,w_j,UB_j) C_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.*

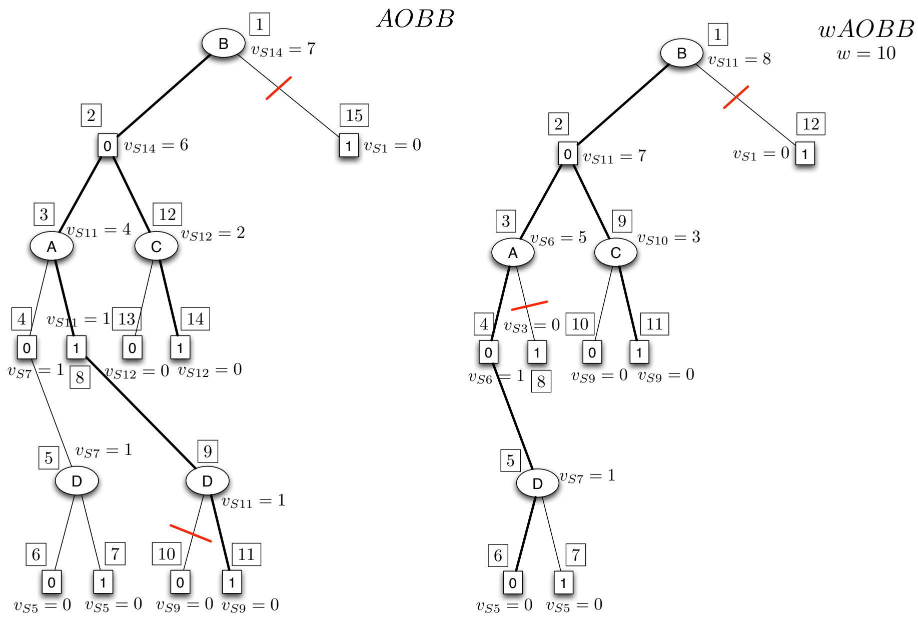
Proof The solution cost C_j with which AOBB(h_i,w_j,UB_j) terminates at iteration j is bounded: $C_{w_j} \leq w_j \cdot C^*$. The upper bound used for pruning at iteration $j > 1$ is equal to the cost of the solution on the previous iteration ($UB_j = C_{j-1}$). No intermediate solutions worse than this 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 (on the left) and by the first iteration of wAOBB with weight $w = 10$ (on the right), when solving the problem from Figure 4. In this example AOBB explores 15 nodes in order to find an optimal solution. wAOBB explores 12 nodes, discovering a suboptimal solution with cost $C = 8$ and assignment $\mathbf{x} = \{A = 0, B = 0, C = 1, D = 0\}$.

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

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

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. □



(a) (b)

Fig. 16 Search graphs explored by (a) AOBB, (b) First iteration of wAOBB, w=10. Boxed numbers 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.

Algorithm 6: wAOBB(w_0, h_i)

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

Initialize $j = 1$, $UB_j = \infty$, $w_j = w_0$, weight update schedule S and let $\mathscr{C} \leftarrow \emptyset$;
while $w_j >= 1$ **do**
  while *AOBB(h_i,w_j, UB_j) not terminated* **do**
    run AOBB(h_i,w_j, UB_j)
    if *AOBB found an intermediate solution* C'_j **then**
      output the solution bounded by the weight of previous iteration: $\mathscr{C} \leftarrow \mathscr{C} \cup \{\langle w_{j-1}, C'_j \rangle\}$
  output the solution with which AOBB terminated, bounded by the current weight: $\mathscr{C} \leftarrow \mathscr{C} \cup \{\langle w_j, C_j \rangle\}$
  Decrease weight w according to schedule S;
  $UB \leftarrow C_j$
  return $\mathscr{C}$

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 com-

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 / -1489.57 96.93 / -1489.57	— / — — / — — / —	— / — — / — — / —

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 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.

pared 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.

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

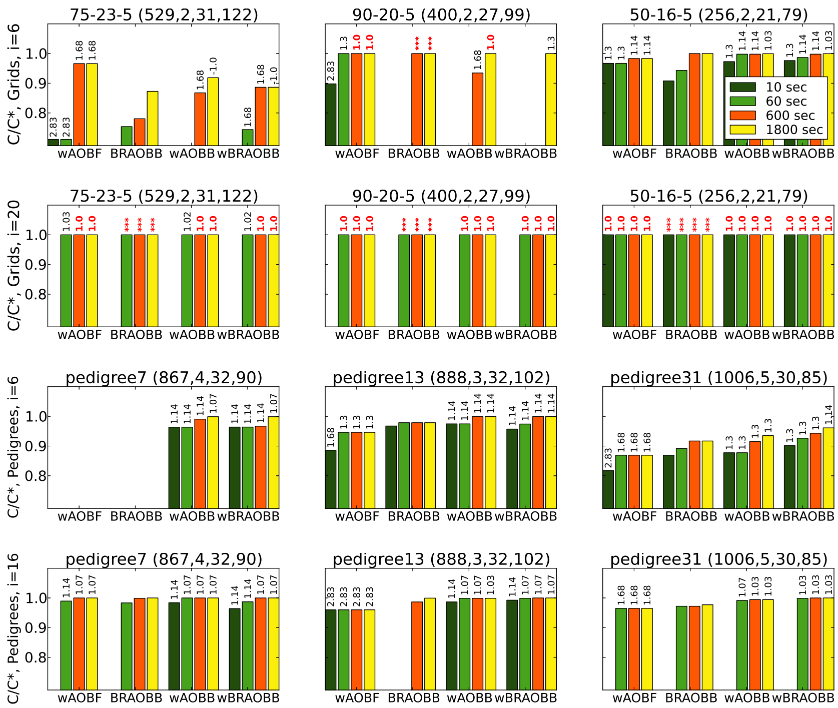


Fig. 17 Ratio of the cost obtained by some time point (10, 60, 600 and 1800 sec) and max cost. Max. cost = optimal, if known, otherwise = best cost found for the problem. Corresponding weight - above the bars. The cases where BRAOBB proved solution optimality is indicated by '***' above bars. In red - 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.

solution starting from a particular weight, e.g. $w = 2.8284$, 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 cost by BRAOBB at termination. The time equal to 3600 seconds for BRAOBB signifies that it failed to report the optimal solution within the time bound and we then report the best solution found.

Comparing with weighted heuristic best-first search. Based on Table 4 we see that time-wise none of the schemes dominates for a given weight. For example, 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 suboptimality. 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 suboptimality bounds, yielding significant time savings compared to finding optimal solutions by competing BRAOBB. The weighted branch and bound schemes are more memory

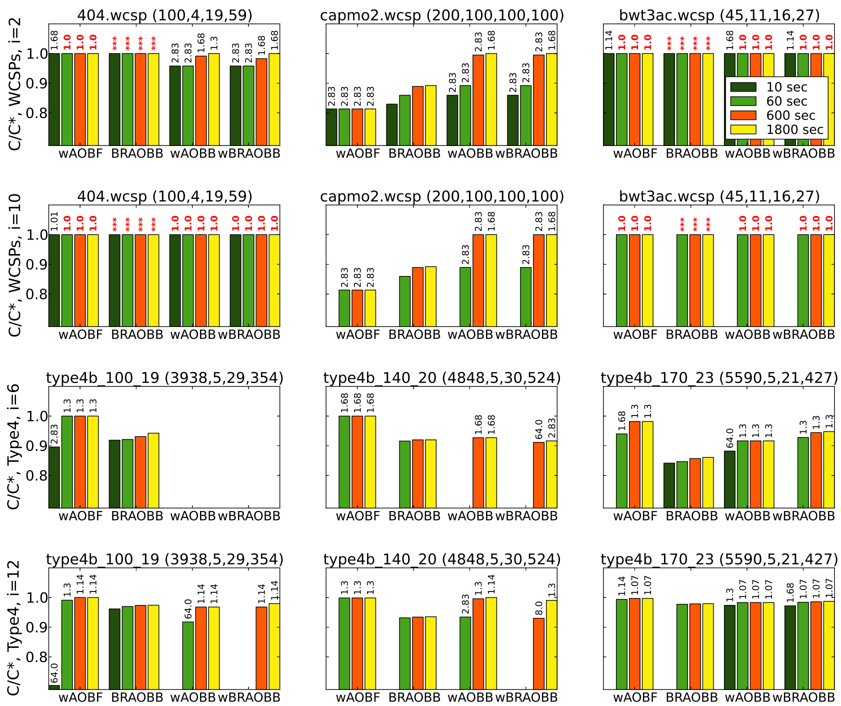


Fig. 18 Ratio of the cost obtained by some time point (10, 60, 600 and 1800 sec) and max cost. Max. cost = optimal, if known, otherwise = best cost found for the problem. Corresponding weight - above the bars. The cases where BRAOBB proved solution optimality is indicated by '***' above bars. In red - 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.

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 behaviour 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 Figures 9 and 10 in the previous section). Figure 17 shows the results for Grids and Pedigrees, while Figure 18 presents WCSPs and Type4 instances.

Grids (Figure 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=6. However, there are exceptions (e.g., grid 50-16-5, i=6).

Pedigrees: both weighted heuristic branch and bound schemes are often the best (e.g. pedigree13, i=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=6.

WCSPs (Figure 18): The wAOBB and wBRAOBB perform better than BRAOBB and wAOBF on such 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 mostly dominates over the branch and bound schemes, included the weighted heuristic ones. However, for larger i-bounds on Type4 weighted heuristic branch and bound schemes can find good solutions, sometimes even providing tighter suboptimality guarantees than wAOBF. For example, for type4b_140_20, i=12, for 1800 sec the bound by wAOBF is $w = 1.3$ while for wAOBB it is $w = 1.14$. BRAOBB is inferior for this benchmark.

We turn now to Figures 11-14 in order to summarize the performance of weighted heuristic depth-first branch and bound search algorithms, concentrating on wAOBB as a slightly better of 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=10, wAOBB finds solutions of higher accuracy than BRAOBB on 23.1% of instances, while for i=18, superiority is 16.7%. The two schemes are tied on 82.1% of instances for Grids, i=18, 3600 seconds, and in general, when the i-bound is high, the difference between the solution costs reported by various algorithms diminishes.

We conclude from these figures that the wAOBB can definitely be superior to wAOBF on 2 out of 4 benchmarks (Pedigrees and WCSPs) for many problems, and can find solution 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 w). Alternatively, when run in anytime fashion, whenever stopped they generate the best solution encountered thus far and a weight w bounding its suboptimality.

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 AND/OR best-first search scheme AOBF and AND/OR depth-first branch and bound scheme AOBB (see [Marinescu and Dechter (2009b)]). This resulted in two anytime weighted heuristic best-first schemes, wAOBF and wR-AOBF, and two weighted depth-first, wAOBB and wBRAOBB. We evaluated these algorithms against each other and against several competing schemes, and especially against the state of the art BRAOBB [Otten and Dechter (2011)], on a large variety of instances from 4 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 [Dechter and Rish (2003)], 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 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 behaviour than wR-AOBF and produced more accurate solutions in a comparable time on many instances (see Figures 11-15).
- **Performance varied per benchmark:** On two out of four benchmarks wAOBF and wR-AOBF dominated over BRAOBB, more often finding costs of better accuracy within

the time limit. This good behaviour on Grids and Type4 benchmarks was mostly consistent across heuristic strengths and time bounds. The weighted heuristic DFS branch and bound schemes wAOBB and wBRAOBB were better than wAOBF and BRAOBB on Pedigrees and, especially, 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 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 becomes 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 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 un-weighted schemes BRAOBB or AOBF. 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 [Xu et al (2008)] and PBP [Gerevini et al (2009)].

In that 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 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 bound on the SLS solution.

The potential impact of weights on various heuristics. While we evaluated the weighted 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 behaviour with any other heuristics. We have recently provided some initial empirical evaluation of the impact of the weighted schemes for cost-shifting relaxation schemes that augment the mini-bucket scheme, introduced in [Ihler et al (2012)]. Our initial findings are presented in [Sharma et al (2014)].

Acknowledgement

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.

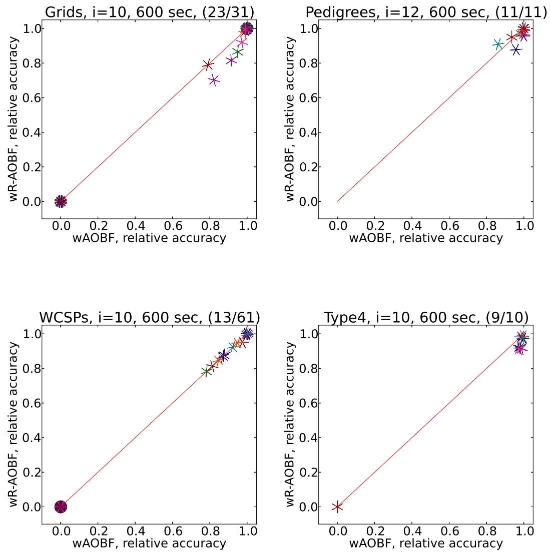


Fig. 19 wAOBF vs wR-AOBF, all benchmarks: comparison of relative accuracy at 600 sec. Each marker represents a single instance. Memory limit 4 GB, time limit 1 hour. In parenthesis (X/Y): X - # instances, for which at least one algorithm found a solution, Y - total # instances, MBE heuristic.

A Weighted heuristic BF: summaries by scatter diagrams

In this section we include additional scatter plots, showing in Figure 19 the comparison between wAOBF and wR-AOBF for two time bound and two levels of heuristic strength and in Figures 20-20 the comparison between wAOBF and BRAOBB for 3 values of i-bounds and 3 time bounds for each instance. For completeness we include also some of the plots previously presented in Section 5.4.3.

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 5 time bounds and for 4 i-bounds we show the percentages of the instances for which a particular weighted 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).

References

Bertele U, Brioschi F (1972) Nonserial dynamic programming. Academic Press, Inc.

Cabon B, De Givry S, Verfaillie G (1998) Anytime lower bounds for constraint violation minimization problems. In: Principles and Practice of Constraint ProgrammingCP98, Springer, pp 117–131

Chakrabarti P, Ghose S, De Sarkar S (1987) Admissibility of AO* when heuristics overestimate. Artificial Intelligence 34(1):97–113

Dechter R (1999) Bucket elimination: A unifying framework for reasoning. Artificial Intelligence 113(1):41–85

Dechter R, Mateescu R (2007) AND/OR search spaces for graphical models. Artificial Intelligence 171(2-3):73–106

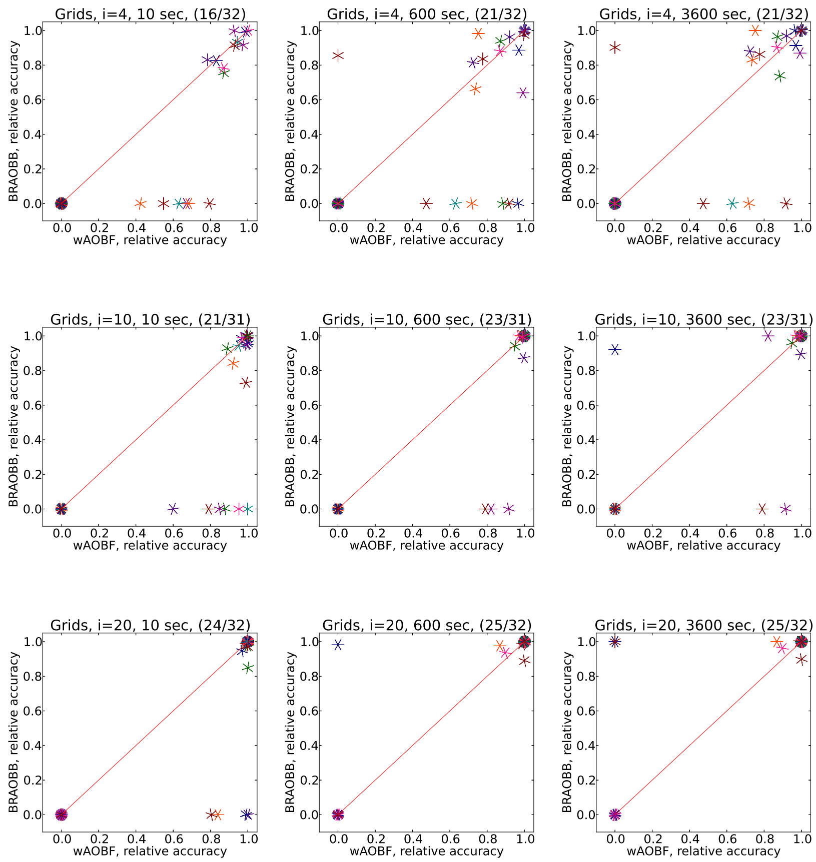


Fig. 20 wAOBF vs BRAOBB on Grids: comparison of relative accuracy at times 10, 600 and 3600 sec. Each row - a single time bound. Each marker represents a single instance. Memory limit 4 GB, time limit 1 hour. In parenthesis (X/Y): X - # instances, for which at least one algorithm found a solution, Y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

Dechter R, Rish I (2003) Mini-buckets: A general scheme for bounded inference. Journal of the ACM 50(2):107–153

Delisle E, Bacchus F (2013) Solving Weighted CSPs by successive relaxations. In: Principles and Practice of Constraint Programming, Springer, pp 273–281

Fishelson M, Geiger D (2002) Exact genetic linkage computations for general pedigrees. In: International Conference on Intelligent Systems for Molecular Biology (ISMB), pp 189–198

Fontaine M, Loudni S, Boizumault P (2013) Exploiting tree decomposition for guiding neighborhoods exploration for vns. RAIRO-Operations Research 47(02):91–123

Gerevini A, Saetti A, Vallati M (2009) An automatically configurable portfolio-based planner with macro-actions: Pbp. In: International Conference on Automated Planning and Scheduling

Hansen E, Zhou R (2007) Anytime heuristic search. Journal of Artificial Intelligence Research 28(1):267–297

Hart P, Nilsson N, Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. IEEE Trans on Systems Science and Cybernetics 4(2):100–107

Hutter F, Hoos HH, Stützle T (2005) Efficient stochastic local search for MPE solving. In: International Joint Conference on Artificial Intelligence, pp 169–174

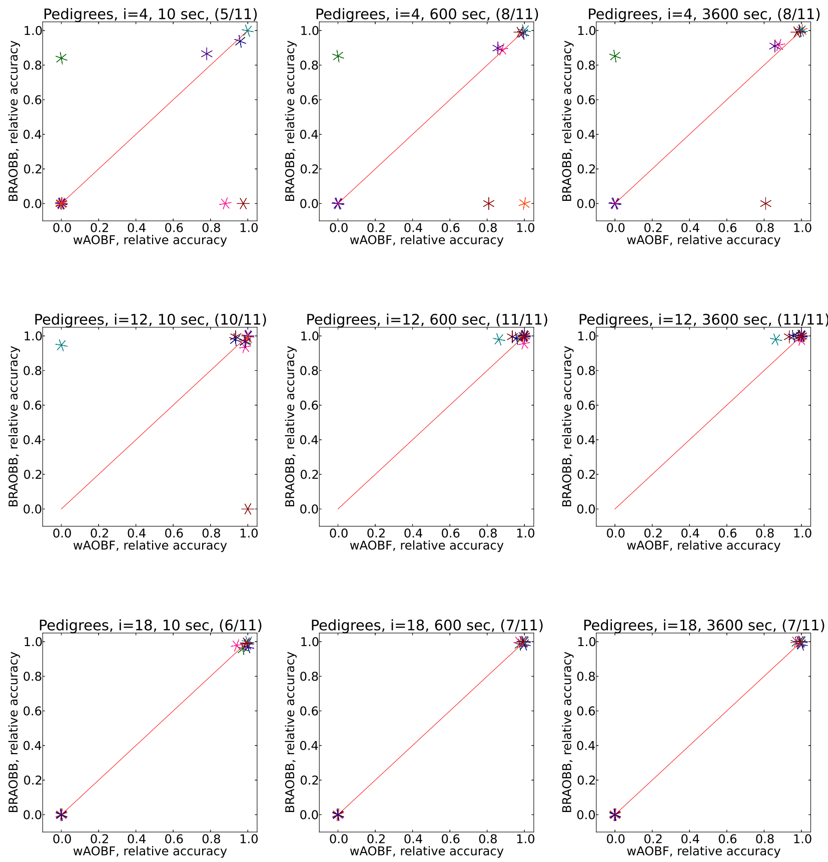


Fig. 21 wAOBF vs BRAOBB on Pedigrees. comparison of relative accuracy at times 10, 600 and 3600 sec. Each row - a single time bound. Each marker represents a single instance. Memory limit 4 GB, time limit 1 hour. In parenthesis (X/Y): X - # instances, for which at least one algorithm found a solution,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

Ihler AT, Flerova N, Dechter R, Otten L (2012) Join-graph based cost-shifting schemes. Uncertainty in Artificial Intelligence

Kask K, Dechter R (1999a) Branch and bound with mini-bucket heuristics. In: International Joint Conference on Artificial Intelligence, vol 99, pp 426–433

Kask K, Dechter R (1999b) Mini-bucket heuristics for improved search. In: Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence, Morgan Kaufmann Publishers Inc., pp 314–323

Kask K, Dechter R (1999c) Stochastic local search for bayesian networks. AI and Statistics

Kask K, Dechter R, Larrosa J, Dechter A (2005) Unifying cluster-tree decompositions for automated reasoning. Artificial Intelligence Journal

Kjærulff U (1990) Triangulation of graphs–algorithms giving small total state space. Tech Report R-90-09

Lawler EL, Wood DE (1966) Branch-and-bound methods: A survey. Operations research 14(4):699–719

Lecoutre C, Roussel O, Dehani DE (2012) WCSP integration of soft neighborhood substitutability. In: Principles and Practice of Constraint Programming, Springer, pp 406–421

Likhachev M, Gordon G, Thrun S (2003) ARA*: Anytime A* with provable bounds on sub-optimality. Neural Information Processing Systems 16

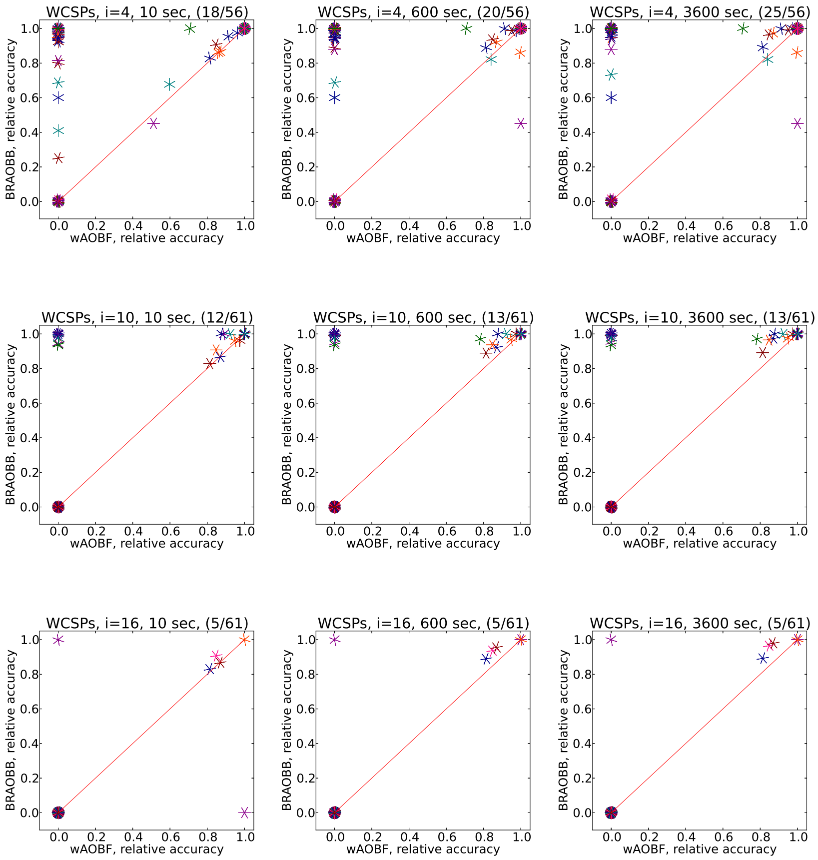


Fig. 22 wAOBF vs BRAOBB on WCSPs: comparison of relative accuracy at times 10, 600 and 3600 sec. Each row - a single time bound. Each marker is a single instance. Memory limit 4 GB, time limit 1 hour. In parenthesis (X/Y): X - # instances, for which at least one algorithm found a solution, Y - total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

Marinescu R, Dechter R (2005) AND/OR branch-and-bound for graphical models. In: International Joint Conference on Artificial Intelligence, Lawrence Erlbaum Associates Ltd., vol 19, p 224

Marinescu R, Dechter R (2007) Best-first AND/OR search for graphical models. In: Proceedings of the National Conference on Artificial Intelligence, Citeseer, vol 22, p 1171

Marinescu R, Dechter R (2009a) AND/OR Branch-and-Bound search for combinatorial optimization in graphical models. Artificial Intelligence 173(16-17):1457–1491

Marinescu R, Dechter R (2009b) Memory intensive AND/OR search for combinatorial optimization in graphical models. Artificial Intelligence 173(16-17):1492–1524

Neveu B, Trombettoni G, Glover F (2004) Id walk: A candidate list strategy with a simple diversification device. Principles and Practice of Constraint Programming–CP 3258:423–437

Nillson NJ (1980) Principles of Artificial Intelligence. Tioga, Palo Alto, CA

Otten L, Dechter R (2011) Anytime AND/OR depth first search for combinatorial optimization. In: International Symposium on Combinatorial Search

Pearl J (1984) Heuristics: Intelligent Search Strategies. Addison-Wesley

Pohl I (1970) Heuristic search viewed as path finding in a graph. Artif Intell 1(3-4):193–204

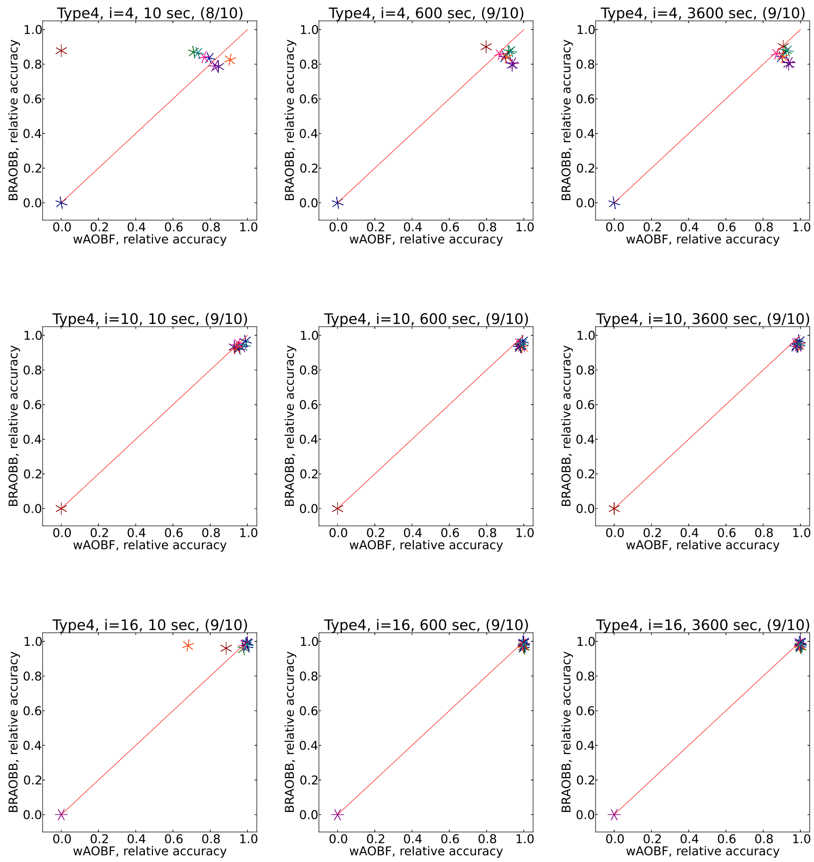


Fig. 23 wAOBF vs BRAOBB on Type4: comparison of relative accuracy at times 10, 600 and 3600 sec. Each row - a single time bound. Each marker is a single instance. Memory limit 4 GB, time limit 1 hour. In parenthesis (X/Y): X - # instances, for which at least one algorithm found a solution, Y - total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

Richter S, Thayer J, Ruml W (2010) The joy of forgetting: Faster anytime search via restarting. In: International Conference on Automated Planning and Scheduling, pp 137–144

Sharma P, Flerova N, Dechter R (2014) Empirical evaluation of weighted heuristic search with advanced mini-bucket heuristics for graphical models. Tech. rep., University of California Irvine

Sontag D, Choe DK, Li Y (2012) Efficiently searching for frustrated cycles in MAP inference. arXiv preprint arXiv:12104902

Thayer J, Ruml W (2010) Anytime heuristic search: Frameworks and algorithms. In: International Symposium on Combinatorial Search

Van Den Berg J, Shah R, Huang A, Goldberg K (2011) ANA*: Anytime nonparametric A*. In: Proceedings of Twenty-fifth AAAI Conference on Artificial Intelligence (AAAI-11)

Wang H, Daphne K (2013) Subproblem-tree calibration: A unified approach to max-product message passing. In: Proceedings of the 30th International Conference on Machine Learning (ICML-13), pp 190–198

Wilt CM, Ruml W (2012) When does weighted A* fail? In: International Symposium on Combinatorial Search

Xu L, Hutter F, Hoos HH, Leyton-Brown K (2008) SATzilla: Portfolio-based algorithm selection for SAT. J Artif Intell Res(JAIR) 32:565–606

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

Table 5 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. # 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.

.

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

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. # 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.

.