Weighted anytime search: new schemes for optimization over graphical models

Natalia Flerova · Radu Marinescu · Rina Dechter

Abstract Weighted 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 best-first and weighted depth-first Branch and Bound search schemes are competitive energy-minimization anytime optimization algorithms. Weighted 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 depth-first search has not been studied for graphical models. We report on a significant empirical evaluation, demonstrating the potential of both weighted 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 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 class of weighted heuristic search schemes is in broadening the applicability
of best-first search, converting it into an anytime scheme, by using the weight as a parameter controlling the trade-off between time, memory and accuracy. The common idea is to have multiple executions of a weighted scheme, gradually reducing the weight along some schedule. A valuable byproduct of these schemes is that the weight offers a suboptimality bound on the ratio of the generated solution cost and an optimal one.

In this paper we investigate the potential of weighted heuristic search to 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 anytime behavior. They are therefore rarely used for graphical models. What we show in this paper is that weighted heuristics can be used to make the best-first search scheme effective, useful and competitive for graphical models as well. The following paragraphs elaborate.

The most common search scheme for combinatorial optimization tasks, such as MAP / MPE or Weighted CSP [Marinescu and Dechter (2009a)], is depth-first Branch-and-Bound. Its use for finding both exact and approximate solutions was extensively studied in recent years [Kask and Dechter (2001), Marinescu and Dechter (2009a), Marinescu and Dechter (2009b), Otten and Dechter (2011), de Givry et al (2006)]. Meanwhile, and as already mentioned, best-first search algorithms, though known to be more effective in bounding the search space [Dechter and Pearl (1985)], are seldom considered for graphical models due to their inherently large memory requirements and their inability to provide any solution before termination. Furthermore, one of best-first’s most attractive features, avoiding the exploration of unbounded paths, seems irrelevant since solutions are of equal depth (i.e., the number of variables).

In contrast, in path-finding domains, 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)]. The idea is to inflate the heuristic function guiding the search by a constant factor of \( w > 1 \), making the heuristic inadmissible, while still guaranteeing a solution cost within a factor of \( w \) from the optimal and typically yielding faster search. If the (non-optimal) solution is found quickly, the search for a better solution may resume. Several anytime weighted 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 is in extending and evaluating these ideas of weighted search to 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)], which are, respectively a best-first and depth-first Branch and Bound algorithms over AND/OR search spaces developed for graphical models. Both AOBF and AOBB traverse the context minimal AND/OR graph guided by an admissible and consistent mini-bucket heuristic [Dechter and Rish (2003), Kask and Dechter (2001), Dechter and Mateescu (2007)]. We focus on a variant called Breadth-Rotating AND/OR Branch and Bound (BRAOBB) [Otten and Dechter (2011)] as the main competing scheme since this version of AOBB was
Weighted Search in Graphical Models

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

extended to yield good anytime behavior over AND/OR search space. BRAOBB was instrumental in winning the 2011 Probabilistic Inference Challenge\(^1\) in all optimization categories.

We explored a variety of weighted schemes. The two best-first schemes that emerged as most promising after an extensive preliminary empirical evaluation were wAOBF and wRAOBF. Both apply weighted best-first search iteratively while decreasing \(w\). wAOBF starts afresh at each iteration, while wRAOBF reuses search efforts from previous iterations, extending ideas presented in Anytime Repairing A* (ARA*) [Likhachev et al (2003)]. Our empirical analysis revealed that weighted best-first search algorithms can be favorable compared with BRAOBB on a significant number of instances from a variety of domains, both in terms of anytime performance and as bounding algorithms with suboptimality guarantees.

While we were motivated by applying weighted heuristics to best-first scheme primarily, we also explored the benefit of using weights for depth-first search, yielding schemes wAOBB and wBRAOBB. In a sense the weights facilitate an alternative anytime approach to existing Branch and Bound schemes and most importantly equip those schemes with the desirable suboptimality bounds. The empirical evaluation revealed the potential of these weighted Branch and Bounds schemes. On a considerable number of instances they outperformed both BRAOBB and the anytime weighted best-first algorithms, while providing suboptimality solution guarantees, which BRAOBB lacks.

Overall, our experiments highlight the power of schemes that are based on the weighted search, both best-first and depth-first Branch and Bound, and thus enrich the set of solvers that should be considered for solving optimization over graphical models alongside BRAOBB, possibly jointly used within a portfolio scheme [Huberman et al (1997)].

Additionally, our theoretical contribution lies in analysing the properties of the search space explored by weighted best-first search, in particular, introducing a notion of focused search and deriving the optimal value of the weight that a) yields a greedy search with least loss of accuracy; b) when computed over an arbitrary solution path provides a guarantee on the solution accuracy.

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 best-first search and reason about values of the weights that make this exploration efficient. Section 4 presents our extension of anytime weighted 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 best-first (5.2), reports on our evaluation of different weight policies (5.3) and compares the anytime performances of our two anytime weighted best-first schemes against the baseline of BRAOBB (5.4). Section 6 introduces the two anytime weighted depth-first Branch and Bound schemes (6.1) and presents their empirical evaluation (6.2). Section 7 summarizes and concludes.

\(^1\) http://www.cs.huji.ac.il/project/PASCAL/realBoard.php
Best-First Search (BFS) maintains a graph of explored paths and a frontier of OPEN nodes. It chooses from OPEN a node with lowest value of an evaluation function \( f(n) \), expands it, and places its child nodes in OPEN. The most popular variant, A*, uses 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 \( r \forall n \ h(n) \leq h^*(n) \).

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 \( C \leq w \cdot C^* \). 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: \( f(n') = g(n') + w \cdot h(n') \leq w \cdot (g(n') + h(n')) \). 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 \( \mathcal{M} = (X, D, F, \otimes) \), where \( X = \{X_1, \ldots, X_d\} \) is a set of variables and \( D = \{D_1, \ldots, D_d\} \) is the set of their finite domains of values. \( F = \{f_1, \ldots, f_k\} \) is a set of non-negative real-valued functions defined on subsets of variables \( x_S \subseteq X \), called scopes (i.e., \( \forall i \ f_i : x_S \rightarrow \mathbb{R}^+ \)). The set of function scopes implies a primal graph whose vertices are the variables and which includes an edge connecting any two variables that appear in the scope of the same function (e.g. 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 \( w^* \). For more details see [Kask et al (2005)]. The combination operator \( \otimes \in \{\prod, \Sigma\} \) defines the complete function represented by the graphical model \( \mathcal{M} \) as \( \mathcal{C}(X) = \otimes_{j=1}^D f_j(x_S) \).

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 optimizing configuration \( x^* \):

\[
C^* = C(x^*) = \max_{X} \prod_{j=1}^D f_j(x_S)
\] (1)
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:

$$x^* = \text{argmax}_x \prod_{j=1}^{r} f_j(x_{S_j})$$  \hspace{1cm} (2)

Bucket Elimination (BE) [Dechter (1999), Bertele and Brioschi (1972)] solves the 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 parameterized by the $i$-bound, is time and space exponential in $i$ only. When $i$ is large enough (i.e., $i \geq w^*$), MBE coincides with full BE. Mini-bucket elimination is often used to generate heuristics for both Best-First and Depth-First Branch and Bound search over graphical models [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).

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

Given a graphical model $M = (X,D,F)$ with primal graph $G$ and a pseudo tree $T$ of $G$, the AND/OR search tree $S_T$ based on $T$ has alternating levels of OR and AND nodes. Its structure is based on the underlying pseudo tree. The root node of $S_T$ is an OR node labeled by the root of $T$. The children of an OR node $\langle X_i \rangle$ are AND nodes labeled 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 labeled with the children of $X_i$ in $T$, 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 $C_T$ (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 $T$ [Dechter and Mateescu (2007)].

A solution tree $T$ of $C_T$ is a subtree such that: (1) it contains the root node of $C_T$; (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 $C_T$ 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* [Nilsson (1980)] that explores the AND/OR context-minimal search graph. It was developed by [Marinescu and Dechter (2009a)].

AOBF, described by Algorithm 1, maintains the explicated part of the context minimal AND/OR search graph and also keeps track of the current best partial solution tree $T^*$. AOBF interleaves iteratively a top-down node expansion step (lines 4-5), selecting a non-terminal tip node of $T^*$ and generating its children in explored search graph $G$, with a bottom-up cost revision step (lines 6-7), updating the values of the internal nodes based on the children’s values. The algorithm also marks the arc to the best AND child of an OR node through which the minimum is achieved (line 7). Following the backward step, a new best partial solution tree $T^*$ is recomputed (line 8). AOBF terminates when there are no more nodes to expand (all tip nodes in $T^*$ are terminal). At this point $T^*$ is the optimal solution with cost $v(s)$, where $s$ is the root node of the search space.

**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. It is described in Algorithm 2. A node $n$ will be pruned if the current upper bound exceeds a heuristic lower bound on the minimal solution extending the conditioned subproblem at $n$. The algorithm interleaves forward node expansion with a backward cost revision or propagation step 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. 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.

**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)].
3 Theoretical insights to weighted best-first search

Before we describe the main contribution of our paper we will give some intuition for the expected behaviour from weighted best-first search. 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. As was discussed in [Wilt and Ruml (2012)], for some domains greedy search can be less efficient than A*. In this section we explore the interplay between the weight \( w \) and heuristic function \( h \) and their impact on the explored search space.

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.

**Definition 2 (focused search space)** An explored search space is focused along a path \( \pi \), 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* using \( w_h \), and that is value depends on the costs of the arcs on the solution paths and on the heuristic values along the path.

**Proposition 1** Let \( \pi \) be a solution path in a rooted search space. Let arc \((n,n')\) be such that \( f(n) > f(n') \). If \( n \) is ever expanded by an A* search 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 favor 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 \( f(n'') \leq f(n') \).
Algorithm 2: AOBB($w, h, UB = \infty$)

Input: A graphical model $\mathcal{M} = (X, D, F)$, pseudo tree $\mathcal{T}$ rooted at $X_1$; weight $w$ (default value 1), heuristic $h$;

Output: Optimal solution to $\mathcal{M}$

1. Create root OR node $s$ labelled by $X_1$ and let stack of created but not expanded nodes OPEN = \{s\};
2. Initialize $v(s) = \infty$ and best partial solution tree rooted in $s$ $T^*(s) = \emptyset$;

while OPEN $\neq \emptyset$ do
3. Select top node $n$ on OPEN.

//EXPAND
5. Expand node $n$:
6. if $n$ is OR node labelled $X_j$ then
7. foreach $x_i \in D(X_j)$ do
8. add AND child $n' = (X_i, x_i)$ to list of successors of $n$ initialize $v(n') = 0$, best partial solution tree rooted in $n$ $T^*(n') = \emptyset$
9. if $n$ is AND node labelled $(X_j, x_j)$ then
10. foreach OR ancestor $m$ of $n$ do
11. evaluate partial solution tree rooted in $m$, based on heuristic $w \cdot h$, assign its cost to $f(m)$
12. if evaluated partial solution is not better than current upper bound at $m$ (i.e. $f(m) \geq v(m)$) then
13. prune the subtree below the current tip node $n$
14. else
15. foreach successor $X_j$ of $X_j \in \mathcal{T}$ do
16. add OR child $n' = X_j$ to list of successors of $n$ initialize $v(n') = \infty$, best partial solution tree rooted in $n$ $T^*(n') = \emptyset$
17. add successors of $n$ on top of OPEN
18. //PROPOGATE

while list of successors of node $n$ is not empty do
19. if node $n$ is the root node then
20. return solution: $v(n), T^*(n)$
21. else
22. update ancestors of $n$, AND and OR nodes $p$, bottom up:
23. if $p$ is AND node then
24. $v(p) = v(p) + v(n)$, $T^*(p) = T^*(p) \cup T^*(n)$
25. else if $p$ is OR node then
26. if $v(p) > (c(p, n) + v(n))$ then
27. $T^*(p) = T^*(p) \cup (x_i, X_j)$
28. if the new value of better than the old one
29. remove $n$ from the list of successors of $p$
30. move one level up: $p = n$

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 $f(n) \geq f(n')$. However, node $q$ has smaller depth than $n$, otherwise it would have been expanded before $n$ (as they have the same $f$ value), and thus smaller depth than $n'$, which is not expanded yet and thus is the descendant of $n$. Either way, node $q$ will not be expanded before $n'$. □
In the following we assume that the algorithms we consider always break ties in favor of deeper nodes.

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

From Proposition 1 it immediately follows:

**Theorem 2** Given a solution path \( \pi \) along which evaluation function \( f \) is monotonically non-increasing, then the search space is focused along path \( \pi \).

We will next show that this focused search property can be achieved by \( W^* \) using certain weights value of \( w \). We denote by \( k(n, n') \) the cost of the arc from node \( n \) to its child \( n' \).

**Definition 4 (the h-weight of an arc)** We denote

\[
    w_h(n, n') = \frac{k(n, n')}{h(n) - h(n')}
\]

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

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

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

\[
    w_h(\pi) = \max_{(n, n') \in \pi} w_h(n, n') = \max_{(n, n') \in \pi} \frac{k(n, n')}{h(n) - h(n')}
\]

**Theorem 3** Given a solution \( \pi \) in a search graph and a heuristic function \( h \) such that \( w_h(\pi) \) is well defined, then, if \( w > w_h(\pi) \), \( W^* \) using \( w \) yields a focused search along \( \pi \).

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

\[
    w \geq \frac{k(n, n')}{h(n) - h(n')}
\]

or, equivalently,

\[
    k(n, n') \leq w \cdot h(n) - w \cdot h(n')
\]

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

\[
    g_\pi(n) + k(n, n') + w \cdot h(n') \leq g_\pi(n) + w \cdot h(n)
\]

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 \( W^* \) is focused along \( \pi \) with this \( w \).
Clearly therefore,

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

**Corollary 2** When $h = h^*$, then on an optimal path $\pi$, \[ \frac{k(n,n')}{h(n) - h(n')} = 1. \] 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 4** Given a search space, and given an admissible heuristic function $h$ yielding $w_h(\pi)$ for every $\pi$. Let $\pi$ be a solution path from $s$ to $t$, satisfying:
1) for all arcs $(n,n') \in \pi$, $k(n,n') \geq 0$, and for one arc at least $k(n,n') > 0$
2) for all arcs $(n,n') \in \pi$, $h(n) - h(n') > 0$,

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

$$C_\pi \leq w_h(\pi) \cdot C^*$$

(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 $\pi$.

Since for any arc $(n,n')$ on path $\pi$, 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,
\[ w_h(\pi) \cdot h(s) \leq w_h(\pi) \cdot h^*(s) = w_h(\pi) \cdot C^* \]
We get from the above 2 inequalities that
\[ w_h(\pi) \cdot C^* \geq C_\pi \]

In the extreme we can use $w^* = \max_\pi w_h(\pi)$ 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 $w^*$ 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 $h(n, n') > 0$, then $w_h(\pi) \geq 1$.

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

It is easy to conclude from Theorem 4 that

**Proposition 3** For every $\pi$, and under the conditions of Theorem 4
\[ C_\pi \geq C^* \geq \frac{C_\pi}{w_h(\pi)} \]
and therefore,
\[ \min_\pi \{ C_\pi \} \geq C^* \geq \max_\pi \{ \frac{C_\pi}{w_h(\pi)} \} \]

In summary, the above analysis provides some intuition as to why weighted 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. In this paper we explore the impact the weights and shed some light on the connection with the heuristic’s accuracy within the context of anytime search. There is clearly room for further exploration on the potential of the relationship $C^* \geq \frac{C_\pi}{w_h(\pi)}$, and the potential of Proposition 3 to yield to any algorithmic advance. We leave this for future work.

4 Tailoring weighted BFS to graphical models

As mentioned, weighted best-first search is a popular framework which was investigated extensively in both planning-related and general search literature. After analyzing 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).
Weighted Search in Graphical Models

Algorithm 3: \texttt{wAOBF}(w_0, h_i)

\textbf{Input}: A graphical model \texttt{M} = \langle X, D, F \rangle; heuristic \(h_i\) calculated with i-bound \(i\); initial weight \(w_0\), weight update schedule \(S\)

\textbf{Output}: Set of suboptimal solutions \(C\)

1. Initialize \(w = w_0\) and let \(\mathcal{C} \leftarrow \emptyset\);
2. while \(w \geq 1\) do
3. \(\langle C_w, T^* \rangle \leftarrow \texttt{AOBF}(w \cdot h_i)\);
4. \(\mathcal{C} \leftarrow \mathcal{C} \cup \{\langle w, C_w, T^* \rangle\}\);
5. Decrease weight \(w\) according to schedule \(S\);
6. return \(\mathcal{C}\);

Algorithm 4: \texttt{wR-AOBF}(w_0, h_i)

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

\textbf{Output}: Set of suboptimal solutions \(C\)

1. Initialize \(w = w_0\) and let \(\mathcal{C} \leftarrow \emptyset\);
2. Create root OR node labeled by \(X_1\) and let \(\mathcal{G} = \{s\}\);
3. Initialize \(v(s) = w \cdot h_i(s)\) and best partial solution tree \(T^*\) to \(\mathcal{G}\);
4. while \(w \geq 1\) do
5. Expand and update nodes in \(\mathcal{G}\) using \texttt{AOBF}(w \cdot h_i) search with heuristic function \(w \cdot h_i\);
6. If \(T^*\) has no more tip nodes then \(\mathcal{C} \leftarrow \mathcal{C} \cup \{\langle w, C_w, T^* \rangle\}\);
7. Decrease weight \(w\) according to schedule \(S\);
8. For all leaf nodes in \(n \in \mathcal{G}\), update \(v(n) = w \cdot h_i(n)\). Update the values of all nodes in \(\mathcal{G}\) using the values of their successors. Mark best successor of each OR node.
9. Recalculate \(T^*\) following the marked arcs;
10. return \(\mathcal{C}\);

**Weighted AOBF.** The fixed-weighted version of the AOBF algorithm is obtained by multiplying mini-bucket heuristic function with 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 basically identical to WAO*, an algorithm introduced previously by Chakrabarti et al (1987), but it is adapted to the specifics of AOBF. Clearly, if \(h_i(n)\) is admissible, which is always true for mini-bucket heuristics, the cost of the solution discovered by weighted AOBF is bounded by a factor of \(w\) from the optimal one, same as for WA* and WAO*.

**Iterative Weighted AOBF (wAOBF).** Since the accuracy of Weighted AOBF is bounded by the weight \(w\), it is common to extend the algorithm to an anytime scheme, which we call \texttt{wAOBF} (Algorithm 3). The algorithm executes Weighted AOBF iteratively, decreasing the weight at each iteration according to a given schedule. This approach is identical to the Restarting Weighted A* by Richter et al (2010) applied to AOBF. It results in a series of solutions, each with a suboptimality factor equal to the weight \(w\).

**Anytime Repairing AOBF (wR-AOBF).** Running each search iteration from scratch seems redundant, since the same search subspace might be explored multiple times. To remedy this problem we introduce Anytime Repairing AOBF (Algorithm 4), denoted by \texttt{wR-AOBF}. This algorithm is an extension of the Anytime 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 runs of the algorithm 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 the OPEN list before starting a new iteration.

Extending the idea of recomputing the heuristic evaluation function for each weight update to the AND/OR search space is fairly straightforward. Since AOBF does not maintain explicit OPEN and CLOSED lists, \textit{wR-AOBF} keeps track of the partially explored AND/OR search graph, and after each weight update it performs a bottom-up update of all the node values starting from the leaf nodes (whose $h$-values are multiplied by the new weight) and continuing towards the root node (line 8). During this phase, the algorithm also marks the best AND successor of each OR node in the search graph. These markings are used to recompute the best partial solution tree $T'$. Then, the search resumes in the usual manner by expanding a tip node of $T'$ (line 9).

Like ARA*, \textit{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.

5 Empirical evaluation of weighted BFS

In the first part of our empirical evaluation we focus on the two weighted best-first schemes. We have conducted a number of experiments solving one of the most common optimization problems classes in graphical models: finding the Most Probable Explanation in Bayesian networks.

5.1 Overview and methodology

We evaluate the two weighted best-first algorithms described in Section 4: \textit{wAOBF} and \textit{wR-AOBF}. Additionally we contrast their performance with 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 \footnote{http://www.cs.hju.ac.il/project/PASCAL/realBoard.php}. 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. All schemes use the mini-bucket heuristics [Dechter and Rish (2003)], whose strength is controlled by a parameter $i$-bound. Higher $i$-bounds typically yield more accurate heuristics but take more time and space, which is exponential in the $i$-bound. We used 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.

We evaluated the algorithms on 4 benchmarks, which include randomly generated binary grids, pedigree networks from from UAI 2008 competition\footnote{http://graphmod.ics.uci.edu/group/Repository}, Type4 genetic networks, and Weighted CSP problems used in 2011 Probabilistic Inference Challenge \footnote{http://www.cs.hju.ac.il/project/PASCAL/archive/mpe.tgz}. In random...
grid networks, the nodes are arranged in an \( N \times N \) square and the functions are defined over pairs of variables and are generated uniformly randomly. The pedigree and type4 instances come from the domain of genetic linkage analysis and are associated with the task of haplotyping. The Weighted CSP networks benchmark includes graph colouring problems, SPOT5 networks and other Weighted CSP domains. Table 1 describes the benchmark parameters: \# inst - number of instances, \( n \) - number of variables, \( k \) - maximum domain size, \( w^* \) - induced width, \( h_T \) - pseudotree height.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># inst</th>
<th>( n )</th>
<th>( k )</th>
<th>( w^* )</th>
<th>( h_T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pedigrees</td>
<td>11</td>
<td>581-1006</td>
<td>3-7</td>
<td>16-39</td>
<td>52-104</td>
</tr>
<tr>
<td>Grids</td>
<td>32</td>
<td>144-2500</td>
<td>2-2</td>
<td>15-90</td>
<td>48-283</td>
</tr>
<tr>
<td>WCSP</td>
<td>56</td>
<td>25-1057</td>
<td>2-100</td>
<td>5-287</td>
<td>11-337</td>
</tr>
<tr>
<td>type4</td>
<td>10</td>
<td>3907-8186</td>
<td>5-5</td>
<td>21-32</td>
<td>319-825</td>
</tr>
</tbody>
</table>

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

In all our experiments for each anytime solution by an algorithm we record its cost, CPU time in seconds and the corresponding weight (for weighted schemes). For uniformity we consider all problems as solving the maximization-product task, also known as Most Probable Explanation problem (MPE). Therefore the computed anytime costs returned by the algorithms are lower bounds on the optimal solutions.

In our empirical evaluation we will address the impact of the following aspects on the performance of the proposed algorithms:

1. The impact of the weight on the solution accuracy and runtime
2. The choice of weight decreasing policy
3. The anytime behaviour of the schemes
4. The interaction between heuristic strength and the multiplicative weight

5.2 The impact of weights on the weighted AOBF performance

One of the most valuable qualities of weighted 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 of the solution.

In this section we demonstrate the impact of the weight on the solution accuracy and runtime of weighted AND/OR Best First (AOBF) search. In order to do that we run \( w \)AOBF and we consider its 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 heuristic \( h_i \), having i-bound equal to \( i \), multiplied by the current value of weight \( w_j \). The solution costs and runtimes of each iterations are independent of each other and depend on the current weight.
<table>
<thead>
<tr>
<th>Instance ((n, k, w^*, h_f))</th>
<th>BRAOBB (C^*) (\log(\text{cost})) time</th>
<th>AOBF((w)) weights (1.00)</th>
<th>BRAOBB (C^*) (\log(\text{cost})) time</th>
<th>AOBF((w)) weights (1.00)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time (sec)</td>
<td>log(cost)</td>
<td>time (sec)</td>
<td>log(cost)</td>
</tr>
<tr>
<td>50-16.5 ((256, 2, 21, 79))</td>
<td>2601.46</td>
<td>-16.916</td>
<td>7.16</td>
<td>7.01</td>
</tr>
<tr>
<td></td>
<td>0.16</td>
<td>-21.095</td>
<td>-16.916</td>
<td>-16.916</td>
</tr>
<tr>
<td>50-17.5 ((289, 2, 23, 77))</td>
<td>1335.44</td>
<td>0.05</td>
<td>9.42</td>
<td>9.44</td>
</tr>
<tr>
<td></td>
<td>-17.759</td>
<td>-23.496</td>
<td>-17.759</td>
<td>-17.759</td>
</tr>
<tr>
<td>75-18-5 ((324, 2, 24, 85))</td>
<td>590.72</td>
<td>0.42</td>
<td>13.52</td>
<td>13.95</td>
</tr>
<tr>
<td></td>
<td>-8.911</td>
<td>-10.931</td>
<td>-8.911</td>
<td>-8.911</td>
</tr>
<tr>
<td>75-20-5 ((400, 2, 27, 99))</td>
<td>17.8</td>
<td>1.78</td>
<td>22.52</td>
<td>24.96</td>
</tr>
<tr>
<td></td>
<td>-16.282</td>
<td>-12.72</td>
<td>-14.067</td>
<td>-12.72</td>
</tr>
<tr>
<td>90-21-5 ((441, 2, 28, 106))</td>
<td>187.75</td>
<td>1.13</td>
<td>17.01</td>
<td>17.32</td>
</tr>
<tr>
<td></td>
<td>-7.638</td>
<td>-8.871</td>
<td>-7.638</td>
<td>-7.638</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pedigrees</th>
<th>1-bound=6</th>
<th>1-bound=10</th>
</tr>
</thead>
<tbody>
<tr>
<td>pedgree9 ((935, 7, 27, 100))</td>
<td>time out</td>
<td>1082.02</td>
</tr>
<tr>
<td></td>
<td>-117.178</td>
<td>-122.904</td>
</tr>
<tr>
<td>pedgree13 ((883, 3, 32, 102))</td>
<td>time out</td>
<td>388.36</td>
</tr>
<tr>
<td></td>
<td>-88.563</td>
<td>-144.882</td>
</tr>
<tr>
<td>pedgree37 ((726, 5, 20, 72))</td>
<td>4.36</td>
<td>0.18</td>
</tr>
<tr>
<td></td>
<td>-144.882</td>
<td>-144.882</td>
</tr>
<tr>
<td>pedgree39 ((953, 5, 20, 77))</td>
<td>time out</td>
<td>4.3</td>
</tr>
<tr>
<td></td>
<td>-174.304</td>
<td>-155.608</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WCSP</th>
<th>1-bound=2</th>
<th>1-bound=10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1502.wcsp ((209, 4, 5, 11))</td>
<td>time out</td>
<td>1563.44</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>-1.258</td>
</tr>
<tr>
<td>42.wcsp ((190, 4, 26, 72))</td>
<td>time out</td>
<td>54.34</td>
</tr>
<tr>
<td></td>
<td>-144.882</td>
<td>-144.882</td>
</tr>
<tr>
<td>bwt3ac.wcsp ((45, 11, 16, 27))</td>
<td>2.47</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>-0.561</td>
<td>-1.85</td>
</tr>
<tr>
<td>capmo5.wcsp ((200, 100, 100, 100))</td>
<td>time out</td>
<td>24.04</td>
</tr>
<tr>
<td></td>
<td>1.18</td>
<td>-0.262</td>
</tr>
<tr>
<td>myciel5g2.wcsp ((47, 3, 19, 24))</td>
<td>2661.91</td>
<td>5.02</td>
</tr>
<tr>
<td></td>
<td>-64.0</td>
<td>-1109.91</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type4</th>
<th>1-bound=6</th>
<th>1-bound=16</th>
</tr>
</thead>
<tbody>
<tr>
<td>type4b_10019 ((3938, 5, 29, 354))</td>
<td>time out</td>
<td>33.32</td>
</tr>
<tr>
<td></td>
<td>5.02</td>
<td>-1099.91</td>
</tr>
<tr>
<td>type4b_23017 ((4072, 5, 24, 319))</td>
<td>time out</td>
<td>26.06</td>
</tr>
<tr>
<td></td>
<td>4.16</td>
<td>-1483.588</td>
</tr>
<tr>
<td>type4b_36019 ((5348, 5, 30, 366))</td>
<td>time out</td>
<td>44.94</td>
</tr>
<tr>
<td></td>
<td>7.28</td>
<td>-1765.403</td>
</tr>
<tr>
<td>type4b_50014 ((5804, 5, 32, 522))</td>
<td>time out</td>
<td>38.22</td>
</tr>
<tr>
<td></td>
<td>16.15</td>
<td>-2007.388</td>
</tr>
<tr>
<td>type4b_17023 ((5590, 5, 21, 427))</td>
<td>time out</td>
<td>18.62</td>
</tr>
<tr>
<td></td>
<td>9.65</td>
<td>-2191.859</td>
</tr>
</tbody>
</table>

Table 2 Runtime (sec) and cost obtained by AOBF\((w, h)\) for selected \(w\), and by BRAOBB (that finds \(C^*\) - optimal cost). Instance parameters: \(n\) - number of variables, \(k\) - max domain size, \(w^*\) - induced width, \(h_f\) - pseudo tree height. "—" - running out of memory. 4 GB memory limit, 1 hour time limit.

Table 2 shows the results for weighted AOBF\((w, h)\) for selected values of weight \((n=2.828, 1.033, 1.00)\), for several selected instances representative of the behaviour preva-
lent over each benchmark. The leftmost column shows the names and parameters of each instance. The remainder of 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 "—" throughout the table 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.

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 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’ runtime are often quite similar. For example, for wrt3ac.wcsp, i=10, BRAOBB takes 54.34 seconds and weighted AOBF - 54.88. On some WCSP instances, such as myciel5g3.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 time 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.

Overall, weighted AOBF manages to generate solutions to some hard problems that are infeasible for the exact scheme and often manages to find accurate solutions with reasonably tight bounds considerably faster than the optimal solutions obtained by BRAOBB or exact AOBF.

### 5.3 Identifying good weight policy

How should we choose the starting weight value and weight decreasing policy? Previous works on weighted 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.]. In our preliminary experiments we wanted to gain some understanding on
the influence of the weight decreasing policy on the performance of our weighted Best First schemes. We experimented with 5 different policies. We considered only fixed weights at each iteration.

We chose the starting weight \( w_0 \) to be equal to 64. The initial weight value needed 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.

The first two policies we considered were \textit{subtract}, which decreases the weight by a fixed quantity, and \textit{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.

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.

Overall, we evaluated the following five policies, each for several values of parameters. Given the parameters \( k \) and \( d \):

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

Figure 4 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: \textit{subtract}(\( k = 0.1 \)), \textit{divide}(\( k = 2 \)), \textit{inverse}(), \textit{piecewise}(\( k = 1.05, d = 8 \)), \textit{sqrt}(\( k = 1.0 \)).

Figure 4 The dependency of the weight value on iteration index according to considered weight policies, showing first 50 iterations, starting weight \( w_0 = 64 \).
Fig. 5 wAOBF: solution log-cost vs time (sec) for different weight policies, starting weight = 64. Instance parameters are in format (n,k,w∗,hT), where n - number of variables, k - max. domain size, w∗ - induced width, hT - pseudo-tree height. Time limit - 1 hour, memory limit - 2 GB.

Figures 5 and 6 show the anytime performance of the weight scheduling schemes, namely how the solution cost changes as a function of time in seconds. We plot the solution cost on logarithmic scale. Figure 5 displays the results for wAOBF, for 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 6 shows analogous results for wR-AOBF, on the same instances.
Fig. 6 wR-AOBF: solution log-cost 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.

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.

We observe in Figure 5 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 instances 75-23-5 and 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 SPOT5 instance 505 the results are more varied or the difference in negligible.

Figure 6 depicts the same information for wR-AOBF. The variance between the result 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 408. However, there are still many instances, for which sqrt policy is as good, for example, pedigree7.

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

5.4 Anytime behaviour of weighted BFS

We now turn to our main focus which is evaluating the anytime performance of our two iterative weighted best-first schemes wAOBF and wR-AOBF and comparing against the anytime depth-first Branch and Bound algorithm BRAOBB, known to have good anytime behaviour. We did not compare against other types of anytime schemes like stochastic local search because they are not guaranteed to achieve an optimal solution when given enough time, and also because some comparison was already carried out with BRAOBB [Otten and Dechter (2012)].

We ran each scheme on instances from the same 4 benchmarks, using mini-bucket heuristics with i-bound ranging from 2 to 20. The algorithms yielded solutions at different points in time, up until either the optimal solution was found or until the algorithm ran out of 4 GB of memory or the time cut off of 3600 seconds was reached. When comparing two anytime algorithms, we consider one to be superior to another if it: 1) discovers the initial solution faster and 2) for a fixed time it returned a solution of better accuracy.

The results are summarized in bar charts displaying a selected set of individual instances (Figures 7, 8), in scatter diagrams (Figures 9-13) and in Tables 3 and 4.

5.4.1 Comparing wAOBF vs wR-AOBF

Comparison on nodes expanded. Table 3 reports solution cost, weight and number of nodes expanded by wAOBF and wR-AOBF for 2 selected instances from each benchmark, for medium i-bound.

\[\text{http://www.cs.huji.ac.il/project/PASCAL/realBoard.php}\]
<table>
<thead>
<tr>
<th>Instance</th>
<th>Algorithm</th>
<th>Time bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>log(cost)</td>
<td>log(cost)</td>
</tr>
<tr>
<td></td>
<td>weight</td>
<td>weight</td>
</tr>
<tr>
<td></td>
<td>nodes</td>
<td>nodes</td>
</tr>
<tr>
<td>Grids, (i=10)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>75-16-5 (256,2,21,73)</td>
<td>wAOBF</td>
<td>-8.1262</td>
</tr>
<tr>
<td></td>
<td>wr-AOBF</td>
<td>-8.0642</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.7490</td>
</tr>
<tr>
<td>75-26-5 (676,2,36,129)</td>
<td>wAOBF</td>
<td>-24.3951</td>
</tr>
<tr>
<td></td>
<td>wR-AOBF</td>
<td>2.8284</td>
</tr>
<tr>
<td></td>
<td></td>
<td>266458</td>
</tr>
<tr>
<td>Pedigrees, (i=10)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>wR-AOBF</td>
<td>-114.4256</td>
</tr>
<tr>
<td></td>
<td></td>
<td>180035</td>
</tr>
<tr>
<td>pedigree41 (885,5,33,100)</td>
<td>wAOBF</td>
<td>-123.6391</td>
</tr>
<tr>
<td></td>
<td>wR-AOBF</td>
<td>-123.6391</td>
</tr>
<tr>
<td></td>
<td></td>
<td>45784</td>
</tr>
<tr>
<td>WCSPs, (i=10)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>408.wcsp (200,4,34,87)</td>
<td>wAOBF</td>
<td>-2.6798</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12908</td>
</tr>
<tr>
<td>capmo5.wcsp (200,100,100,100)</td>
<td>wAOBF</td>
<td>-0.2622</td>
</tr>
<tr>
<td></td>
<td>wR-AOBF</td>
<td>-0.2622</td>
</tr>
<tr>
<td></td>
<td></td>
<td>47517</td>
</tr>
<tr>
<td>WCSPs, (i=10)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type4, (i=10)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type4b,150 (5804,5,32,522)</td>
<td>wAOBF</td>
<td>-1698.1897</td>
</tr>
<tr>
<td></td>
<td>wR-AOBF</td>
<td>-1698.1897</td>
</tr>
<tr>
<td></td>
<td></td>
<td>173853</td>
</tr>
<tr>
<td></td>
<td></td>
<td>233884</td>
</tr>
</tbody>
</table>

Table 3 Solution cost, weight and number of expanded nodes 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.

As theory suggests, we observe that wR-AOBF almost always expands less nodes than wAOBF, often by orders of magnitude. For example, for type4b,150,14, 60 seconds wAOBF expands 743927 nodes, while wR-AOBF - just 32217. There are a few exceptions, for ex-
Fig. 7 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. 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.

One might expect wR-AOBF to be superior to wAOBF, since the former is specifically designed to avoid wastefully re-expanding the previously explored nodes in the search space and thus is supposed to be more efficient. However, in practice, looking at Fig. 7 and 8 we

...
observe that wR-AOBF has worse anytime behaviour than the simpler scheme wAOBF for about half of the instances. For example, in Fig. 7 for the pedigree9 and pedigree13, i=6, wAOBF finds more accurate solutions for the same time bounds. This can be explained by the large overhead at each iteration of wR-AOBF, due to the need to keep track of already expanded nodes and to update their evaluation function as the weight changes. As a result, despite the lower number of nodes expanded, each iteration of wR-AOBF is typically not much faster or is even slower than an iteration by wAOBF.

Comparing the suboptimality bounds. When observing the weights depicted for particular time points, we see several trends. Often wR-AOBF reaches $w = 1$ faster than wAOBF, (see for 75-16-5, i=6 in Fig. 7 or WCSP problem bwt3ac, i=2 in Fig. 8, for example). However, when $w > 1$, wAOBF usually finds a more accurate solution at the same time limit (e.g., 75-18-5, i=6, for $w = 1.3$ at 10 sec). Moreover, in many cases the anytime weighted best-first schemes find solutions of optimal costs even when $w > 1$, (e.g., pedigree9, i=16).

5.4.2 Comparing wAOBF and wR-AOBF against BRAOBB

The performance of wAOBF and wR-AOBF compared to BRAOBB varies a lot from benchmark to benchmark. Figure 7 illustrates the superiority of the anytime weighted best-first schemes, especially wAOBF, on the Grid whenever weak heuristics were used. For example, for grid 75-16-5, i = 6 within 10 seconds wAOBF finds a considerably more accurate
solution than the other two schemes. On many Pedigrees BRAOBB is superior for weak heuristics, e.g. pedigree9, i=6. When the heuristics are stronger the performance is similar for all algorithms’ on Grids and many Pedigrees.

Figure 8 shows that for Type4 instances the anytime weighted best-first is superior for both heuristics strengths. WCSP instances are hard for anytime weighted best-first schemes. For example, both wAOBF and wR-AOBF are inferior to BRAOBB on capmo2 instance, for both i-bounds.

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 values of the 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 best-first schemes when the value of the i-bound is considerably smaller than the induced width of the problem.

Consider again the results in Table 2. Comparing the results across i-bounds for the same algorithm and the same weight, we see that stronger heuristics do not yield improved runtime in all cases. One of the examples of the higher i-bound being beneficial is the performance of BRAOBB on grid 50-17-5 where it finds the optimal solution in 1335.42 seconds when i-bound=6, but requires 9.42 seconds only, when the i-bound=20, which is much closer to the induced width of the problem along the chosen ordering, equal to 23. Similarly, wAOBF takes 41.38 seconds for grid 90-21-5 for \( w = 1.033 \) when i-bound=6, but only 17.65 sec-

Fig. 9 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.
onds, when the i-bound=20. However we observe many other instances where more accurate heuristic come at too high a price. For example, for pedigree37 wAOBF finds a $w\text{-optimal}$ solution with $w = 2.828$ in 0.08 seconds for $i$-bound=6, but takes 388.96 seconds for $i=16$.

When comparing different heuristic strengths across algorithms and all weights in Table 2, we observe that often wAOBF 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 wAOBF with weight $w = 1.033$ has much smaller difference in performance for the two $i$-bounds. wAOBF 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.
Likewise in Figures 7 and 8 we observe that for a number of problems the difference between the anytime performance across i-bounds is more prominent for BRAOBB than for wAOBF and wR-AOBF, (e.g., pedigree13 in Fig. 7).

5.4.3 Summaries by scatter diagrams.

Figures 9 to Figure 13 summarize the experimental results using scatter diagrams. Figure 9 compares wAOBF against wR-AOBF, Figures 10-13 - wAOBF against BRAOBB. We omit explicit comparison between wR-AOBF and BRAOBB, since the results are very similar to the ones shown in Figure 9. Since here we are interested only in comparing the relatively
performance of the search algorithms, we do not account in these plots for the heuristic calculation time, which is the same for all schemes.

Figure 9 (wAOBF vs wR-AOBF) shows 4 scatterplots displaying results for the time bound 600 seconds and a single medium i-bound. These time cut-off and i-bound are representative the behaviour of the algorithms across time and i-bounds. Each plot corresponds to one of our benchmarks. 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 all figures each scatter plot shows the relative accuracy of the two schemes for a particular time bound for each benchmarks. As in the bar charts in Figures 7 and 8, the accuracy is defined in relation to the optimal cost, if known, or the maximum cost available. Each marker’s x-axis coordinate corresponds to the relative accuracy of the solution.
Fig. 13  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.

obtained by one of the schemes, for example by wAOBF in case of Figure 9. The y-axis coordinate corresponds to the relative accuracy of the other schemes, for example wR-AOBF for Figure 9. Values equal to 1.0 are preferable.

wAOBF vs wR-AOBF (Figure 9). We see once again that the simpler scheme wAOBF is often more successful than more sophisticated wR-AOBF. wAOBF dominates on Grids and Type4, while on WCSPs and Pedigrees both algorithms perform almost equally well. Such behaviour is quite consistent across the i-bounds.

wAOBF vs BRAOBB (Figure 10-13). Figures 10-13 present the results comparing wAOBF with BRAOBB in more details. Each row shows a particular i-bound and contains three plots, displaying results at 10, 600 and 3600 seconds. The behaviour of the algorithms,
while being quite consistent for different time bound, varies a lot across benchmarks, just as
we saw previously in Figures 7 and 8. The results speak for themselves.
Overall, we again observe that on Type4 and many Grid instances, especially for short
time bounds, wAOBF is superior, on Pedigrees both schemes show mostly similar results
and BRAOBB dominates on WCSPs.

6 Weighted Depth-First Branch and Bound (BB) for Graphical Models

The primary reason for using weighted heuristics in the context of best-first search is to con-
vert them into memory effective anytime scheme and to get a solution with some bounded
guarantee. Since depth-first search Branch and Bound schemes are inherently anytime, the
idea of using weighted search may seem irrelevant. Still, Branch and Bound schemes do not
provide any guarantees when terminating early. So sheer curiosity coupled with the benefit
of having guaranteed bounds intrigued us into exploring the principle of weighted 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.
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 eval-
uate their performance in much the same way we did for the weighted best-first search
algorithms.

6.1 Weighted Branch and Bound schemes

The extension of AOBB to weighted 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 $h_i$ calculated with $i$-bounds=$i$
in a pre-processing step, multiplied by the weight $w_0$, and an initial upper bound equal to
$UB$.

It is easy to show that:

**Theorem 5** If the heuristic $h_i(n)$ is admissible, algorithm AOBB($h_i, w_0 > 1, UB = \infty$) (and
similarly BRAOBB) terminates with a solution $\pi$, whose cost $C_\pi$ 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:
$C_1 \geq \cdots \geq C_i \geq \cdots \geq C_\pi$, where $C_\pi$ is the returned solution. If $\pi$ is not optimal, (otherwise the
claim is trivially proved) there exists an optimal solution $\pi^*$ which must have been pruned by
the algorithm. Let $n$ be the last node on $\pi^*$ that was generated and which was pruned. Since
the heuristic $h$ is admissible, the un-weighted evaluation function of $f$ along $\pi^*$ satisfies that

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

(6)

Let $C_i$ be the solution cost used to prune $n$ (namely it pruned relative to the weighted eval-
uation 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
\[
C_\pi \leq w \cdot C^*.
\]

We present two iterative weighted 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 of wAOBB (Algorithm 5) it executes \( \text{AOBB}(h_i,w_0,UB=\infty) \), namely AOBB with heuristic \( h_i \cdot w_0 \) and with default upper bound equal to infinity. The algorithm does no pruning until it discovers its first solution. Then the upper bound is set to the current best solution cost. At termination of the first iteration, it 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 \( \text{AOBB}(h_i,w_j,UB_j) \) 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 \( \text{AOBB}(h_i,w_j,UB_j) \) \( C_j \) 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 \( \text{AOBB}(h_i,w_j,UB_j) \) are guaranteed to have costs within the factor of \( w_{j-1} \) from the optimal.

**Proof** The solution cost \( C_j \) with which \( \text{AOBB}(h_i,w_j,UB_j) \) terminates at iteration \( j \) is bounded: \( C_j \leq w_j \cdot C^* \). 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 \( C'_j \leq w_{j-1} \cdot C^* \).

**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.

### 6.2 Empirical evaluation of weighted BB

We carry out an empirical evaluation of the two weighted 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 Branch and Bound schemes against each other and with wAOBF, the superior of the weighted best-first schemes. We also compared against the state-of-the-art anytime BRAOBB. All algorithms use the same mini-bucket heuristic scheme. We use the same \( \sqrt{r}(1.0) \) policy and starting weight equal to 64 as we did for the weighted best-first schemes in Section 5.

In our empirical evaluation we will study the following facets of the weighted Branch and Bound algorithms’ behaviour:

1. Weighted Branch and Bound as \( w \)-optimal approximate schemes
2. The anytime performance compared to weighted best-first schemes and BRAOBB
Algorithm 5: wAOBB\( (w_0, h_i) \)

**Input:** A graphical model \( \mathcal{M} = (X, D, F) \); heuristic \( h_i \) obtained with \( i \)-bound \( t \); initial weight \( w_0 \)

**Output:** \( C \) - a set of suboptimal solutions \( C_i \), each with a bound \( w \)

1. Initialize \( j = 1, UB_j = \infty, w_j = w_0 \); weight update schedule \( S \) and let \( C \leftarrow \emptyset \);
2. while \( w_j > 1 \) do
   3. while \( AOB(h_i, w_j, UB_j) \) not terminated do
   4. run \( AOB(h_i, w_j, UB_j) \)
   5. if \( AOB \) found an intermediate solution \( C_i \) then
   6. output the solution with which \( AOBB \) terminated, bounded by the current weight:
      \[ C \leftarrow C \cup \{C_j\} \]
   7. Decrease weight \( w \) according to schedule \( S \);
   8. \( UB \leftarrow C_j \)
   9. return \( C \)

![Fig. 14](image_url)

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.

6.2.1 Weighted Branch and Bound as approximation

Table 4 reports the runtime in seconds required for each weighted scheme to produce a solution of a particular guaranteed suboptimality level and the corresponding solution cost for two selected instances, for a relatively strong heuristic. For each algorithm we report...
<table>
<thead>
<tr>
<th>Instance</th>
<th>BRAOBB</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.8284</td>
<td>1.2968</td>
<td>1.0330</td>
</tr>
<tr>
<td>wAOBF</td>
<td>wAOBF</td>
<td>wAOBF</td>
</tr>
<tr>
<td>wAOBB</td>
<td>wAOBB</td>
<td>wAOBB</td>
</tr>
<tr>
<td>wBRAOBB</td>
<td>wBRAOBB</td>
<td>wBRAOBB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>time / cost</th>
<th>time / cost</th>
<th>time / cost</th>
<th>time / cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.87 / -17.55</td>
<td>6.1 / -17.55</td>
<td>52.46 / -15.65</td>
<td></td>
</tr>
<tr>
<td>5.91 / -17.55</td>
<td>6.22 / -17.55</td>
<td>79.91 / -15.7</td>
<td></td>
</tr>
</tbody>
</table>

**Grids, I-bound=18**

| 75-25-5 | 3582.08 / -20.836 | 8.0 / -23.38  | 9.77 / -21.7  |                |                |

**Pedigrees, I-bound=18**

| pedigree9 | 220.34 / -122.904 | 12.74 / -123.06 | 13.63 / -123.2 |                |
| pedigree51| 3600 / -111.55    | 31.15 / -121.77 |                |                |

**WCSP, I-bound=6**

| capmo2.wcsp | 3600 / -0.28 | 2.52 / -72.0  | 7.8 / -64.0   | 37.63 / -64.0 |
| mycie5g.wcsp| 3600 / -0.28 | 2.52 / -72.0  | 7.8 / -64.0   | 37.63 / -64.0 |

**Type4, I-bound=18**

| type4b | 3600 / -1.332.18 | 80.49 / -334.93 | 81.24 / -329.59 | 84.98 / -327.6 |
|        | 3600 / -1.338.74 | 86.21 / -1438.24| 88.89 / -1386.34| 96.61 / -1512.32|

<table>
<thead>
<tr>
<th>time / cost</th>
<th>time / cost</th>
<th>time / cost</th>
<th>time / cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.87 / -17.55</td>
<td>6.1 / -17.55</td>
<td>52.46 / -15.65</td>
<td></td>
</tr>
<tr>
<td>5.91 / -17.55</td>
<td>6.22 / -17.55</td>
<td>79.91 / -15.7</td>
<td></td>
</tr>
</tbody>
</table>

**Table 4** Runtime (sec) and cost obtained by wAOBF, wAOBB and wBRAOBB for selected w, and by BRAOBB (that finds C⁰ - optimal cost). Instance parameters: n - number of variables, k - max domain size, w* - induced width, hT - pseudo tree height. "time out" - running out of time, "—" - running out of memory. 4 GB memory limit, 1 hour time limit.

The entire runtime required to reach the target weight (e.g. w = 2.8284). In this sense this table is different from Table 2, where we only reported the time it took the scheme to find the solution starting from a particular weight, e.g. 2.8284, not starting with initial weight w = 64. The difference is due to the fact that wAOBB and wBRAOBB use the results of previous iterations as upper bound and their iterations are not completely independent runs of AOB(hi,wj,UBj) and BRAOB(hi,wj,UBj), respectively.

We also report the runtime and the cost by BRAOBB at termination time. The time equal to 3600 seconds for BRAOBB signifies that it failed to report the optimal solution within the time bound and we report the best solution found.
Comparing with weighted best-first search. We compared wAOBF (top row in every cell) and the weighted Branch and Bound schemes (rows 2 and 3 in each cell). We see that in terms of time none of the schemes is consistently superior for a given weight. For example, on 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), we observe that weighted schemes often yield remarkable time savings compared to BRAOBB.

Overall, we observe that all weighted schemes can often provide good approximate solutions with tight suboptimality bounds, yielding significant time savings compared to finding optimal solutions by competing BRAOBB. Weighted Branch and Bound schemes are more memory efficient than wAOBF which is prone to running out of space. However, on the instances feasible for all three weighted schemes there is no clear dominance between them.

6.2.2 Anytime performance comparison

Figures 14 and 15 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 7 and 8 in the previous section). Figure 14 shows the results for Grids
and Pedigrees, while Figure 15 presents WCSPs and Type4 instances.

Grids (Figure 14): we observe that wAOBF is typically superior to the Branch and
Bound algorithms, finding solutions faster and of better 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 where wAOBB and wBRAOBB are superior to wAOBF, (e.g., grid
50-16-5, i=6). BRAOBB often tends to find the least accurate solutions, only providing any
guarantees, where it proves solution optimality, (e.g., 90-20-5, i=10, 600 and 1800 seconds).

Pedigrees: both weighted Branch and Bound schemes are often superior (e.g. pedi-
gree13, 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 15): wAOBF is mostly inferior. The wAOBB and wBRAOBB perform
better than BRAOBB on many instances (e.g., capmo2.wcsp for all i-bounds), except for
a number of problems (not shown), for which only BRAOBB is the only scheme to return
solutions.

Type4: wAOBF mostly dominates over all Branch and Bound schemes. However, for
larger i-bounds on Type4 Weighted 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 and
wBRAOBB it is \(w = 1.14\). BRAOBB is inferior for this benchmark.

We conclude from these figures that wAOBF is quite competitive, but definitely do not
always dominate. On 2 out of 4 benchmarks (Pedigrees and WCSPs) weighted Branch and
Bound schemes are often superior.

7 Summary, Conclusion and Future work

The paper provides the first study of weighted best-first and weighted depth-first Branch and
Bound search for graphical models. They 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, where \(w\) is a parameter). Alternatively, when run in anytime fashion
they generate the best solution encountered thus far, as well as the weight \(w\) bounding the
suboptimality.

The idea of weighted best-first search is prevalent in the path-finding and planning com-
munities. In this paper we extended this idea to AND/OR Best First search scheme AOBF
for combinatorial optimization problems over graphical models [Marinescu and Dechter
(2009b)], yielding two anytime weighted schemes, wAOBF and wR-AOBF that were most
effective. We evaluated these algorithms against each other and against the state of the art
BRAOBB [Otten and Dechter (2011)], on a large number of instances from 4 benchmarks
while varying the heuristics strengths. We compared with weighted Branch and Bound
schemes as well.

Looking back at the questions we asked ourselves in Sections 5.1 and 6.2 we next high-
light the main trends observed in the experimental results. We also use the summarizing
Table 7 to reinforce some of these conclusions.

– Evaluating algorithms with weighted heuristic as \(w\)-optimal schemes. Weighted
AOBF, wAOBB and wBRAOBB yield \(w\)-optimal solutions. In some cases this is achieved
orders of magnitudes faster, even for small \(w\), than the time required to generate an exact
solution by BRAOBB or AOBF. Moreover, weighted schemes find \( w \)-optimal solutions with a small \( w \) for some hard instances infeasible (within the time limit) for the baseline algorithms, for example on Type4 benchmark.

- **Choosing the weight decreasing policy.** The \( \text{sqrt} \) policy emerged as the most promising out of the several evaluated weight decreasing policies, yielding the best anytime performance for \( w_{\text{AOBF}} \) and \( w_{\text{R-AOBF}} \). This policy decreases the weight quickly initially to facilitate fast improvement of solution and then fine tunes the solution by changing the weight slowly as it approaches 1.0.

- **Anytime \( w_{\text{AOBF}} \) vs anytime \( w_{\text{R-AOBF}} \).** Our experiments showed that, as expected, \( w_{\text{R-AOBF}} \) expand smaller portions of the search space. However, this often did not translate to an overall superior solution cost compared with \( w_{\text{AOBF}} \). We saw throughout that \( w_{\text{AOBF}} \) often finds solutions closer to optimal than \( w_{\text{R-AOBF}} \) does. Overall we observed that \( w_{\text{AOBF}} \) yielded a better trade-off between overhead, time and accuracy.

- **Anytime weighted best-first search vs anytime BRAOBB.** On two out of our four benchmarks the weighted schemes dominated, finding costs of better accuracy within the time limits. This superior behavior on Grids and Type4 benchmarks was mostly consistent across heuristic strengths and time bounds and can also be observed in Table 7. \( w_{\text{AOBF}} \) is superior to BRAOBB on half of our benchmarks. Specifically, on Grids it returns more accurate costs than BRAOBB on up to 68% of problems for certain i-bounds and time bounds, while on Type4 it is superior on 90% or all instances. Out of the two weighted best-first schemes, for most i-bounds and most time limits \( w_{\text{AOBF}} \) dominates over BRAOBB more often than \( w_{\text{R-AOBF}} \) does. This is especially obvious on Type4 benchmark when the i-bound is small, e.g. for \( i=6 \), 3600 seconds \( w_{\text{R-AOBF}} \) is superior to BRAOBB only on 50% of the instances and \( w_{\text{AOBF}} \) on 90%.

- **Anytime weighted Branch and Bound schemes vs \( w_{\text{AOBF}} \) and BRAOBB.** The weighted DFS schemes \( w_{\text{AOBB}} \) and \( w_{\text{BRAOBB}} \) were sometimes superior to \( w_{\text{AOBF}} \) and BRAOBB (e.g., some WCSPS and pedigrees). This superiority often corresponds to cases where \( w_{\text{AOBF}} \) ran out of memory, since those schemes are more memory efficient.

- **On the impact of the heuristic strength.** The weighted schemes tend to be superior to BRAOBB when there is a significant distance between the i-bound which characterizes the heuristic strength and the problem’s induced width (e.g., for Grids, 30 second, for \( i=10 \), \( w_{\text{AOBF}} \) yields better costs than BRAOBB on 56.0% of the instances, \( w_{\text{R-AOBF}} \) on 37.5% and \( w_{\text{AOBB}} \) on 44.0%, while for \( i=18 \), superiority is only on 16.7%, 12.5% and 8.3% respectively.)

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).

Since higher i-bounds often yield exact solutions, the weighted schemes are tied with BRAOBB on a significant percent of instances, e.g. Grids, \( i=18 \), 3600 seconds 75% for \( w_{\text{AOBF}} \) and \( w_{\text{R-AOBF}} \) and 82.1% for \( w_{\text{AOBB}} \).

To sum up, as could be expected, none of the schemes dominates. But our results show that weighted schemes can be quite powerful on a significant number of instances in the
context of anytime search. Since they also provide $w$-optimality guarantees they should be included as candidates solvers for optimization tasks over graphical models.

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. We also plan to automate the algorithm parameter selection based on benchmarks or problems, as well as enrich our set of schemes by considering, for example, dynamic weights.

An alternative to selecting a single scheme for a specific problem is combining our algorithms within a portfolio framework, known to be successful for such solvers as SATzilla [Xu et al (2008)] and PbP [Gerevini et al (2009)]. The question of portfolio building and scheduling also will be studied in the future.

Finally, in Section 3.23 provided some intuition regarding the interplay between the heuristic and the weights in yielding lower and upper bounds on the optimal costs, using a new quantity we called $h$-weight which is function of each path. When computed over an arbitrary solution path, it yields some lower bounds on the optimal costs. We are not sure to what extent these observation may lead to a useful bounds. We plan to empirically investigate the practical importance of such bounds in our future work.

**Acknowledgements** 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.
<table>
<thead>
<tr>
<th>I-bound</th>
<th>Algorithm</th>
<th>Time bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X% / Y% / N</td>
<td>X% / Y% / N</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>600</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3600</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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* - pseudo-tree height. 4 GB memory, 1 hour time limit.
References

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