Weighted Best First Search for MAP

Natalia Flerova
University of California Irvine

Radu Marinescu
IBM Research - Ireland

Rina Dechter
University of California Irvine

Abstract

The paper considers Weighted Best First (WBF) search schemes, popular for path-finding domain, as approximations and as anytime schemes for the MAP task. We demonstrate empirically the ability of these schemes to effectively provide approximations with guaranteed suboptimality and also show that as anytime schemes they can be competitive on some benchmarks with one of the best state-of-the-art scheme, Depth-First Branch-and-Bound.

Introduction

The most common search scheme for combinatorial optimization tasks, such as MAP/MPE or Weighted CSP, 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 2009b; Otten and Dechter 2011; de Givry, Schiex, and Verfaillie 2006). Meanwhile, 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, Nilsson, and Raphael 1968) is clearly favored. 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 in the past decade (Hansen and Zhou 2007; Likhachev, Gordon, and Thrun 2003; van den Berg et al. 2011; Richter, Thayer, and Ruml 2010).

In our work we extended the above methods to graphical models and investigated their potential empirically. We used AND/OR Best First search (AOBF) (Marinescu and Dechter 2009b), a best-first algorithm developed for AND/OR search spaces over graphical models, as a basis. AOBF explores the context minimal AND/OR graph in a best-first manner, guided by the admissible and consistent mini-bucket heuristic (Dechter and Rish 2003; Kask and Dechter 2001; Dechter and Mateescu 2007).

After exploring a variety of approaches and following extensive empirical analysis (including two non-parametric algorithms that interleave depth- and best-first exploration), the two schemes that emerged as most promising were those running WA* iteratively while decreasing w. One (wAOBF) starts from scratch at each iteration and the other (wR-AOBF) reuses search efforts from previous iterations, extending ideas of Anytime Repairing A* (ARA*) (Likhachev, Gordon, and Thrun 2003).

We report on a comprehensive empirical evaluation of the two candidate algorithms (which ran multiple time with multiple heuristics) on over 100 instances from 4 different benchmarks, evaluating their performance both as approximation and anytime schemes. We compared against Breadth-Rotating AND/OR Branch-and-Bound (BRAOBB) (Otten and Dechter 2011), a state-of-the-art anytime Depth-First Branch-and-Bound which won the 2011 Probabilistic Inference Challenge[1] in all optimization categories.

Our empirical analysis revealed that best-first search schemes can be used effectively for optimization over graphical models. Specifically: 1. Weighted BFS provides an effective scheme for generating approximate solutions with upfront guaranteed level of sub-optimality, 2. Weighted BFS can be made into effective anytime schemes and on some benchmarks even outperforms one of the best state of the art scheme which is based on depth-first search.

Background

Best-first search (BFS): 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 $g(n)$

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

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. For a minimization task, $h(n)$ is *admissible* if $\forall n\ h(n) \leq h^*(n)$.

Weighted A* search (WA*): WA* 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 (Pohl 1970).

A **Graphical model** is a tuple $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F}, \prod)$, where **F** is a set of real-valued local functions over subsets of discrete variables **X**, called scopes, with finite domains **D**. The common optimization task is to find $\max_{\mathbf{X}} \prod_i f_i$ called **MAP** or **MPE**, though the same algorithms can easily be used to solve the Weighted CSP problem: $\min_{\mathbf{X}} \sum_i f_i$. The set of function scopes defines a *primal graph* (see example in Figure 1a) and, given an ordering of the variables, yields an *induced graph* (e.g., Figure 1b) with a certain *induced width*. For detail see (e.g. (Kask et al. 2005)).

AND/OR search spaces allow capturing the dependencies of the underlying graphical model during search (Dechter and Mateescu 2007). The AND/OR search space is defined using a *pseudotree* of the primal graph $G = (X, E)$ (e.g., Figure 1c), which captures problem decomposition. It is a directed rooted tree $\mathcal{T} = (X, E')$, such that every arc of G not included in E' is a back-arc in $\mathcal{T}$, namely it connects a node in $\mathcal{T}$ to its ancestor. The associated **AND/OR search tree** consists of alternating levels of OR and AND guided by the pseudotree structure. The children of an OR node $\langle X_i \rangle$ are *AND nodes* labelled with assignments $\langle X_i, x_i \rangle$, and the children of an AND node $\langle X_i, x_i \rangle$ are OR nodes labelled with the children of X_i in the pseudotree $\mathcal{T}$. They root conditionally independent subproblems. The edges of the AND/OR tree are annotated by values derived from the input potentials and finding the optimal-cost solution subtree solves the stated optimization task.

Given a pseudotree $\mathcal{T}$ of height h_T, the size of the AND/OR search tree based on $\mathcal{T}$ is $O(n \cdot k^{h_T})$, where k bounds the variables' domain size. The context-minimal AND/OR search graph has size $O(n \cdot k^w)$, where w is the induced width of the problem graph along a depth-first traversal of $\mathcal{T}$ (Dechter and Mateescu 2007).

Graph-based merging of identical subproblems yields the *context minimal AND/OR search graph* (e.g. Figure 1d) which has size $O(Nk^{w^*})$, where w^* is the induced width of G along a depth first order of $\mathcal{T}$, N is the number of variables and k bounds the domain sizes.

AND/OR Best First search (AOBF): The state-of-the-art version of A* that explores the AND/OR search spaces is the AND/OR Best-First (AOBF) algorithm (Nilsson 1982) that utilizes the mini-bucket heuristic which is admissible and consistent (Dechter and Rish 2003). AOBF described by Algorithm 1 is a variant of AO* (Nilsson 1980). It maintains the explicated part of the context minimal AND/OR search graph. At the same time AOBF keeps track of the current best partial solution tree T^*.

Algorithm 1: AOBF(w, h)

Data: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, pseudo tree $\mathcal{T}$ rooted at X_1; weight w (default value 1), heuristic h;
Result: Optimal solution to $\mathcal{M}$

1 Create root OR node s labelled by X_1 and let $\mathcal{G} = \{s\}$;
2 Initialize $v(s) = w \cdot h(s)$ and best partial solution tree T^* to $\mathcal{G}$;
3 **while** *true* **do**
4 Select non-terminal tip node n in T^*. If there is no such node then **exit**;
5 Expand node n: if $n = X_i$ is OR then for each $x_i \in D(X_i)$ add AND child $n' = \langle X_i, x_i \rangle$ to $\mathcal{G}$; if $n = \langle X_i, x_i \rangle$ is AND then for each successor X_j of X_i in $\mathcal{T}$ add OR child $n' = X_j$ to $\mathcal{G}$; Initialize $q(n') = h(n')$ for all new nodes;
6 Update ancestors AND and OR nodes p of n in $\mathcal{G}$, bottom-up as: if p is OR then $v(p) = \min_{m \in succ(p)}(c(p, m) + v(m))$; else if p is AND then $v(p) = \sum_{m \in succ(p)} v(m)$;
7 Mark best successor m of OR ancestors p, such that $m = \arg\min_{m \in succ(p)}(c(p, m) + v(m))$ maintaining marked successor if still best;
8 Recompute T^* by following marked arcs from the root s;
9 **return** $\langle v(s), T^* \rangle$;

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 $\mathcal{G}$, 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 $v(s)$.

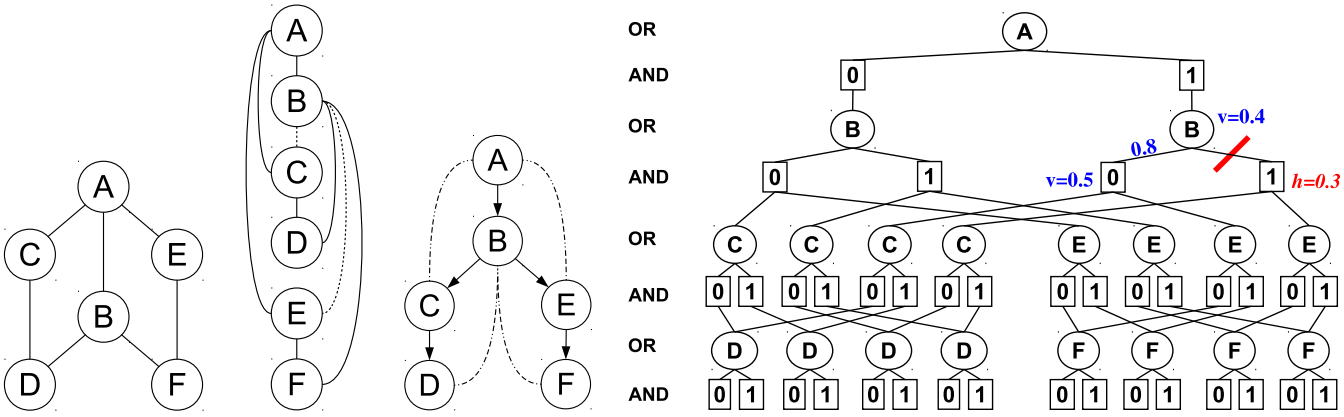
Anytime AND/OR Branch and Bound (BRAOBB): Depth First AND/OR Branch-and-Bound (AOBB) (Marinescu and Dechter 2009a) was shown to be a powerful search scheme for graphical models. The algorithm, however, lacks a proper anytime behavior: at each AND node all but one independent child subproblems have to be solved completely, before the last one is even considered. *Breadth-Rotating AND/OR Branch-and-Bound* (BRAOBB) (Otten and Dechter 2011) remedies this deficiency, rotating through different subproblems in a breadth-first manner. Empirically, BRAOBB finds the first suboptimal solution significantly faster than plain AOBB (Otten and Dechter 2011).

Adapting Anytime Best First Search for Graphical Models

This section describes the anytime Best First algorithms that we adapt to AND/OR search space over graphical models.

Iterative Weighted AOBF (wAOBF): The fixed-weighted version of the AOBF algorithm is obtained by inflating the mini-bucket heuristic function with a weight $w > 1$ (i.e., substituting $h(n)$ by $w \cdot h(n)$). This scheme is basically identical to WAO*, an algorithm introduced previously by (Chakrabarti, Ghose, and De Sarkar 1987), but it is adapted to the specifics of AOBF.

Clearly, like WA* and WAO*, if $h(n)$ is admissible,



(a) Primal graph. (b) Induced graph. (c) Pseudotree. (d) Context-minimal AND/OR search graph with AOBB pruning example.

Figure 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.

Algorithm 2: wAOBF(w_0, h)

Data: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$; heuristic h; initial weight w_0
Result: Set of suboptimal solutions $\mathcal{C}$

Initialize $w = w_0$, weight update schedule S and let $\mathcal{C} \leftarrow \emptyset$;
while $w >= 1$ **do**
  $\langle C_w, T^*_w \rangle \leftarrow$ AOBF($w \cdot h$);
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{\langle w, C_w, T^*_w \rangle\}$;
  Decrease weight w according to schedule S;
return $\mathcal{C}$;

the cost of the solution discovered by weighted version of AOBF is bounded by a factor of w from the optimal one.

Since the accuracy of Weighted AOBF is bounded by w, it is natural to extend it to an anytime scheme, called **wAOBF** (Algorithm 2), that executes Weighted AOBF iteratively, decreasing the weight at each iteration. This approach, similar to the Restarting Weighted A* by (Richter, Thayer, and Ruml 2010), results in a series of solutions, each with a suboptimality factor equal to w.

Anytime Repairing AOBF (wR-AOBF): Running each search iteration from scratch is wasteful, since the same search subspace might be explored multiple times. To remedy this problem Anytime Repairing AOBF (**wR-AOBF**, Algorithm 3) extends *Anytime Repairing A* (ARA*)* algorithm (Likhachev, Gordon, and Thrun 2003) to AND/OR search spaces over graphical models. The original ARA* algorithm utilizes the results of previous steps by recomputing the evaluation functions of the nodes with each weight change, re-using inherited OPEN and CLOSED lists and by keeping track of already expanded nodes whose evaluation function changed between iterations and re-inserting them back to OPEN list before starting a new iteration.

The extension of ARA* to AND/OR search space is not straightforward, since AOBF does not maintain explicit OPEN and CLOSED lists. Instead, at each iteration, **wR-AOBF** keeps track of the partially explored AND/OR graph and, after decreasing w, it performs a bottom-up update of all node values starting from the leaf nodes (whose h-values are inflated with the new weight) and continuing upwards towards the root node. Then, the search continues with the newly identified best partial solution tree. Like ARA*, **wR-AOBF** provides the same guarantees with respect to the quality of the suboptimal solutions found.

Algorithm 3: wR-AOBF(w_0, h)

Data: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$; pseudo tree $\mathcal{T}$ rooted at X_1; heuristic h; initial weight w_0
Result: Set of suboptimal solutions $\mathcal{C}$

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

Experiments

We compare the performance of wAOBF, wR-AOBF and BRAOBB, exploring the same context minimal AND/OR graph. The search space is determined by a common variable ordering. All the algorithms 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 and take more time and space (exp(i)) to

Instance (n, k, w^*, h_T)	BRAOBB	AOBF(w, h) weights				BRAOBB	AOBF(w, h) weights			
		2.828	1.139	1.033	1.00		2.828	1.139	1.033	1.00
	time C^*	time cost	time cost	time cost	time cost	time cost	time cost	time cost	time cost	time cost
Grids	I-bound=6					I-bound=20				
50-17-5 (289, 2, 23, 77)	1335.44 -17.759	0.06 -23.496	35.99 **-17.759**	— —	— —	9.42 -17.759	0.02 -17.829	0.04 -17.829	0.05 **-17.759**	0.09 **-17.759**
50-18-5 (324, 2, 24, 84)	— —	0.04 -25.997	197.12 -21.843	— —	— —	14.59 -21.843	0.03 -22.71	0.17 -22.167	0.49 -22.025	2.51 **-21.843**
75-16-5 (256, 2, 21, 73)	47.1 -8.064	0.48 -9.514	19.09 -8.126	57.96 **-8.064**	201.91 **-8.064**	7.26 -8.064	0.01 **-8.064**	0.02 **-8.064**	0.03 **-8.064**	0.07 **-8.064**
75-18-5 (324, 2, 24, 85)	390.72 -8.911	1.13 -10.931	43.11 **-8.911**	177.19 **-8.911**	705.69 **-8.911**	13.52 -8.911	0.02 -9.078	0.04 -9.078	0.06 **-8.911**	0.19 **-8.911**
75-20-5 (400, 2, 27, 99)	— —	2.89 -16.282	— —	— —	— —	22.52 -12.72	0.06 -14.067	1.26 -12.843	10.24 **-12.72**	59.07 **-12.72**
90-21-5 (441, 2, 28, 106)	187.75 -7.658	2.84 -8.871	76.41 **-7.658**	158.23 **-7.658**	412.06 **-7.658**	17.01 -7.658	0.42 -9.476	0.89 **-7.658**	1.7 **-7.658**	5.02 **-7.658**
Pedigrees	I-bound=6					I-bound=16				
pedigree9 (935, 7, 27, 100)	— —	2.1 -137.178	— —	— —	— —	1082.02 -122.904	0.09 -133.063	0.27 -124.597	29.4 **-122.904**	— —
pedigree13 (888, 3, 32, 102)	— —	0.15 -88.563	— —	— —	— —	— —	0.23 -76.429	— —	— —	— —
pedigree37 (726, 5, 20, 72)	4.36 -144.882	0.05 -163.325	0.65 -145.784	7.67 -145.082	59.28 **-144.882**	388.36 -144.882	0.06 -155.259	0.12 -145.737	0.24 -145.341	1.08 **-144.882**
pedigree38 (581, 5, 16, 52)	204.0 -87.299	0.79 -96.213	1.34 -88.741	26.85 **-87.299**	327.07 **-87.299**	— —	— —	— —	— —	— —
pedigree39 (953, 5, 20, 77)	— —	0.13 -174.304	3.53 -158.743	— —	— —	4.34 -155.608	0.08 -162.381	0.15 **-155.608**	0.29 **-155.608**	3.14 **-155.608**
pedigree41 (885, 5, 33, 100)	— —	3.95 -131.831	47.5 -122.558	— —	— —	— —	1.48 -122.234	3.04 -121.731	— —	— —
WCSP	I-bound=2					I-bound=8				
29.wcsp (82, 4, 14, 23)	0.74 -1.604	0.52 -1.804	4.6 **-1.604**	7.63 **-1.604**	16.76 **-1.604**	0.76 -1.604	0.01 -1.804	0.03 **-1.604**	0.12 **-1.604**	0.52 **-1.604**
404.wcsp (100, 4, 19, 59)	0.4 -2.78	0.33 -2.878	16.39 **-2.78**	29.64 **-2.78**	66.3 **-2.78**	1.63 -2.78	0.0 -2.902	0.08 **-2.78**	3.94 **-2.78**	19.96 **-2.78**
bwt3ac (45, 11, 16, 27)	2.47 -0.561	2.44 **-0.561**	7.4 **-0.561**	11.05 **-0.561**	22.09 **-0.561**	4.46 -0.561	0.47 **-0.561**	1.16 **-0.561**	1.65 **-0.561**	3.14 **-0.561**
driverlog01ac.wcsp (71, 4, 9, 37)	0.06 -0.777	0.02 **-0.777**	0.11 **-0.777**	0.2 **-0.777**	0.5 **-0.777**	0.04 -0.777	0.0 **-0.777**	0.01 **-0.777**	0.01 **-0.777**	0.02 **-0.777**
satellite01ac.wcsp (79, 8, 19, 56)	39.24 -0.076	101.88 **-0.076**	201.28 **-0.076**	267.65 **-0.076**	464.84 **-0.076**	5645.74 -0.076	58.9 **-0.076**	115.24 **-0.076**	152.65 **-0.076**	265.7 **-0.076**
zenotravel02ac (116, 19, 18, 43)	7.79 -0.099	7.59 **-0.099**	26.59 **-0.099**	42.75 **-0.099**	92.63 **-0.099**	— —	— —	— —	— —	— —
Type4	I-bound=6					I-bound=16				
type4b_100_19 (3938, 5, 29, 354)	— —	2.03 -1309.91	— —	— —	— —	— —	1.37 -1171.002	5.34 -1124.091	— —	— —
type4b_120_17 (4072, 5, 24, 319)	— —	1.75 -1483.588	— —	— —	— —	— —	1.02 -1362.607	2.26 -1331.24	82.97 -1327.776	— —
type4b_140_20 (4848, 5, 30, 524)	— —	6.83 -1658.008	— —	— —	— —	— —	17.79 -1448.739	109.11 -1388.239	— —	— —
type4b_150_14 (5804, 5, 32, 522)	— —	6.36 -2007.388	— —	— —	— —	— —	4.59 -1727.035	25.06 -1622.851	— —	— —
type4b_170_23 (5590, 5, 21, 427)	— —	4.16 -2191.859	600.0 -1961.605	— —	— —	— —	2.36 -1978.588	5.08 -1934.153	27.23 -1925.883	— —

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_T - pseudo tree height. In **bold** - cost by AOBF equal to C^*. 4 Gb memory limit, 1 hour time limit.

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.

compute) and all are solving the maximization task (higher costs are better). The algorithms output solutions at different times until either the optimal solution is found or the time limit of 1 hour is reached or the scheme runs out of memory (4 Gb).

We experimented with benchmarks from UAI 2008 competition[2]. The benchmark parameters are presented in Ta-

[2]http://graphmod.ics.uci.edu/group/Repository

ble 1. Each problem was solved with 11 values of i-bound, ranging from 2 to 22. After considering 5 different weight policies, described in the full paper, we settled on the the following: $w_i = \sqrt{w_{i-1}}$, w_i is the weight at iteration i. The starting weight value $w_0 = 64$ was chosen: a) to explore the schemes behaviour on a large range of weights; b) to make the search greedy enough initially to solve harder instances, known to be infeasible for regular BF within the memory limit.

Effectiveness of BF Schemes as Approximations

In this section we report the bounds output by wAOBF and wR-AOBF and talk about the impact of the weight.

Fixing the weight for wAOBF and wR-AOBF provides a *suboptimality guarantee* for the cost (which is bounded by $w \cdot C^*$), while managing the runtime and required space.

In **Table 2** we report runtime (sec) and solution cost for AOBF that uses weighted heuristic with selected values of weight (w=2.828, 2.239, 1.033, 1.00), for 3 selected instances from each benchmark and for 2 values of i-bound for 4 Gb. We also report the run time and the optimal cost found by BRAOBB, where available. We defer discussing BRAOBB as an anytime scheme in the next section.

Note, since calculation of mini-bucket heuristics is time and space $O(exp(i))$, for some instances the heuristics can't be obtained for large i-bounds (e.g. pedigree38, $i = 16$).

We can interpret the results in Table 2 by comparing across pairs of columns. We immediately see the expected behaviour across all instances: as the weight decreases towards 1, time and accuracy both increase, or at least do not decrease.

Quality of solutions. We observe that often the actual results are far better and closer to optimal than the bound suggests. In particular, in a few of the cases, the actual optimal solution is obtained for a $w > 1$. These cases are highlighted by boldface in Table 2.

Time saving for w-bounded suboptimality. It is most informative to compare the exact results by BRAOBB (column 2) and by AOBF (columns 6 and 11, when $w = 1$) with any one of the other columns. Each weighted column represents a particular level of guaranteed suboptimality.

Let's consider, for example, the columns associated with the weight $w = 2.828$, 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 (column 2) both when the i-bound is small (e.g., pedigree38, i=6, 0.79 vs 204.0 seconds) and when it is large (e.g., 0.99 vs 1082.02 sec for pedigree9, i=16), except for WCSPs, where the times are not so different. Notice however, that for WCSPs the actual costs found by wAOBF are often optimal (though the guarantee is still of a factor 2.828). Comparing columns 6 and 11, exhibiting full AOBF with $w = 1$ (when it did not run out of memory) against $w = 2.828$ we see similar behaviour in terms of time savings.

More significantly, consider the column of weight $w = 1.033$, especially for the higher i-bound (strong heuristics). These results are just a factor of 1.033 away from optimal, yet the time savings compared with BRAOBB are significant (e.g., 29.4 vs 1082.02 sec for pedigree9, i=16). We observe two Type4 instances that BRAOBB and pure AOBF were unable to solve (within 3600 sec), for which the time with $w = 1.033$ was just 82.97 and 27.33 sec, respectively.

i	Geometric mean time (sec)					
	BRAOBB	wAOBF weight / # inst				
		2.828	1.139	1.033	1.001	1.00
Grids						
6	121.75 / 12	0.99 / 25	20.81 / 13	30.38 / 8	121.98 / 8	116.04 / 8
16	7.11 / 23	0.19 / 26	0.62 / 22	2.4 / 20	9.98 / 19	9.34 / 29
20	18.4 / 25	0.19 / 26	0.45 / 22	1.48 / 21	4.67 / 20	4.17 / 20
Pedigrees						
6	18.8 / 2	1.45 / 9	3.64 / 4	15.11 / 2	121.79 / 2	145.98 / 2
14	12.76 / 2	0.33 / 9	2.27 / 6	1.19 / 2	11.95 / 2	14.68 / 2
16	122.18 / 3	0.6 / 9	0.69 / 5	1.27 / 3	1.57 / 2	1.84 / 2
WCSPs						
2	9.63 / 14	0.6 / 15	2.67 / 12	4.13 / 12	7.69 / 12	8.28 / 12
6	5.09 / 14	0.0 / 13	0.45 / 7	0.83 / 7	1.96 / 7	0.0 / 7
8	6.52 / 14	0.0 / 60	0.0 / 7	0.55 / 7	1.29 / 7	0.0 / 7
Type4						
6	—	5.51 / 10	58.81 / 1	—	—	—
16	—	3.49 / 10	13.53 / 9	47.53 / 2	—	—
20	44.67 / 1	1.53 / 4	3.4 / 4	4.45 / 2	11.68 / 1	12.66 / 1

Table 3: Geometric mean runtime (sec) and # instances, for which averaging is done, wAOBF for selected weights and BRAOBB. 4 Gb memory, 1 hour time limit.

Table 3 provides summaries over all instances from each benchmark. For each benchmark and for several i-bounds we provide a geometric mean of runtime (sec) for the same set of weights as before. Also shown are the number of instances averaged over (for some w's wAOBF sometimes runs out of memory) and the results for BRAOBB. We observe that the mean runtime of AOBF increases as w decreases towards 1. More accurate heuristics (higher i-bounds) typically allow solving more instances, e.g. for grids $w = 1$ AOBF solves 8 problems for $i = 6$ vs 20 for $i = 20$. However, there are exceptions, since heuristic calculation may become infeasible for larger i-bound, see, for example, WCSPs, $w = 1$.

Effectiveness of Weighted BF as Anytime Schemes

This section focuses on the *anytime performance* of wAOBF and wR-AOBF and on their comparison with the BRAOBB.

Figure 2 displays the anytime behaviour of the three schemes for typical instances from each benchmark for 2 values of i-bound. For each instance we show the ratio between the cost available at a particular time point (at 5, 10, 60 and 600 seconds) and the optimal (if known) or overall maximal cost. The closer the ratio is to 1, the better. The four leftmost bars correspond to wAOBF, the central ones - to wR-AOBF and the four rightmost - to BRAOBB. For wAOBF and wR-AOBF we additionaly display the weight at the time bound above the respective bar, $w = 1$ shown in red.

WCSP instances are hard for BF schemes (e.g. capmo2), except for the easiest ones, solved equally fast by all algorithms (e.g. bwt3ac). Figure 2 demonstrates the superiority

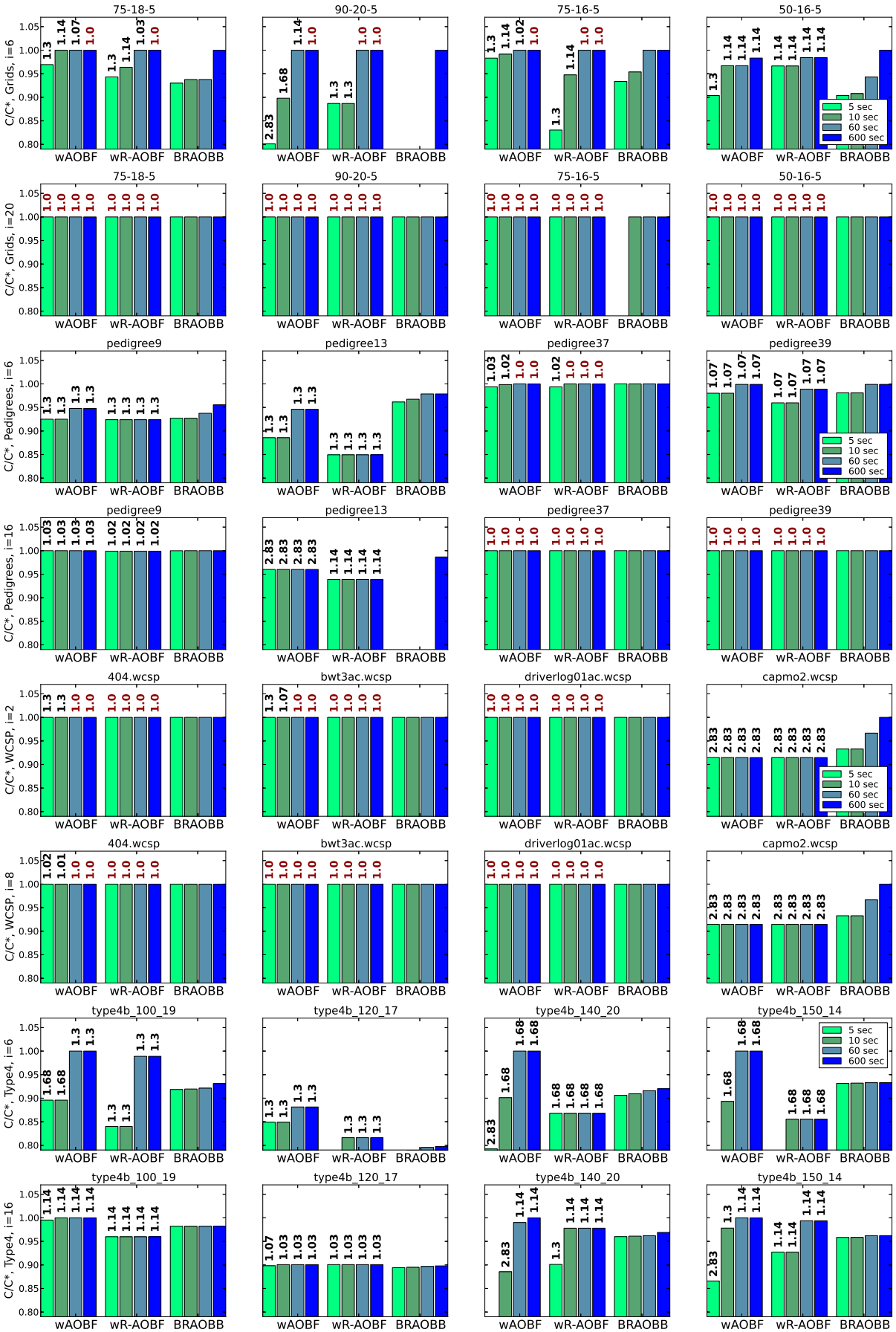


Figure 2: Ratio of the cost obtained by some time point (5, 10, 60, 600 sec) and max cost. Max. cost = optimal, if known, otherwise = best cost found for the problem. Corresponding weight - above the bars. Memory limit 4 Gb, time limit 1 hour.

i	wAOBF wR-AOBF BRAOBB	wAOBF wR-AOBF BRAOBB	wAOBF wR-AOBF BRAOBB	wAOBF wR-AOBF BRAOBB
	mean ac / w / #	*mean ac / w / #*	*mean ac / w / #*	*mean ac / w / #*
	Time bound (sec)			
	10	60	600	3600
Grids				
6	0.88 / 2.43 / 23 0.85 / 1.79 / 23 0.93 / 15	0.92 / 4.19 / 24 0.88 / 1.81 / 24 0.93 / 18	0.95 / 1.48 / 24 0.92 / 1.43 / 24 0.95 / 21	0.95 / 1.48 / 24 0.92 / 1.43 / 24 0.97 / 23
10	0.93 / 1.78 / 22 0.94 / 1.29 / 22 0.94 / 17	0.95 / 3.88 / 24 0.95 / 1.23 / 24 0.97 / 21	0.98 / 1.23 / 24 0.96 / 1.2 / 24 0.99 / 22	0.98 / 1.23 / 24 0.96 / 1.2 / 24 0.99 / 24
16	0.99 / 1.19 / 24 0.99 / 1.08 / 24 0.99 / 21	1.0 / 1.05 / 24 1.0 / 1.03 / 24 0.99 / 22	1.0 / 1.03 / 24 1.0 / 1.03 / 24 1.0 / 23	1.0 / 1.03 / 24 1.0 / 1.03 / 24 1.0 / 24
20	1.0 / 1.07 / 22 1.0 / 1.03 / 22 0.99 / 20	1.0 / 1.01 / 22 1.0 / 1.01 / 22 1.0 / 22	1.0 / 1.01 / 22 1.0 / 1.01 / 22 1.0 / 22	1.0 / 1.01 / 22 1.0 / 1.01 / 22 1.0 / 22
Pedigrees				
6	0.90 / 9.19 / 9 0.90 / 1.31 / 9 0.94 / 9	0.95 / 1.28 / 9 0.93 / 1.23 / 9 0.95 / 9	0.95 / 1.27 / 9 0.93 / 1.23 / 9 0.96 / 9	0.95 / 1.27 / 9 0.93 / 1.23 / 9 0.96 / 9
10	0.94 / 8.39 / 10 0.95 / 1.23 / 10 0.95 / 8	0.97 / 1.42 / 11 0.96 / 1.16 / 11 0.96 / 10	1.02 / 1.28 / 11 0.96 / 1.16 / 11 0.97 / 11	1.02 / 1.28 / 11 0.96 / 1.16 / 11 1.02 / 11
14	0.98 / 1.41 / 9 0.97 / 1.14 / 9 0.98 / 10	0.99 / 1.25 / 9 0.97 / 1.13 / 9 0.98 / 10	0.99 / 1.24 / 9 0.98 / 1.11 / 9 0.99 / 10	0.99 / 1.24 / 9 0.98 / 1.11 / 9 0.99 / 10
16	0.98 / 1.46 / 8 0.98 / 1.09 / 8 0.98 / 8	0.97 / 2.12 / 9 0.97 / 1.1 / 9 0.99 / 8	0.98 / 1.36 / 9 0.97 / 1.1 / 9 0.99 / 9	0.98 / 1.36 / 9 0.97 / 1.1 / 9 0.99 / 9
WCSPs				
2	0.0004 / 7.44 / 11 0.0004 / 7.36 / 11 0.0006 / 18	0.0004 / 1.8 / 11 0.0004 / 1.64 / 11 0.0006 / 18	0.0004 / 1.64 / 11 0.0004 / 1.64 / 11 0.0006 / 18	0.0004 / 1.64 / 11 0.0004 / 1.64 / 11 0.0006 / 18
4	0.50 / 1.73 / 11 0.50 / 1.7 / 11 0.0006 / 18	0.50 / 1.65 / 11 0.50 / 1.64 / 11 0.0006 / 18	0.50 / 1.64 / 11 0.50 / 1.64 / 11 0.0006 / 18	0.50 / 1.64 / 11 0.50 / 1.64 / 11 0.0006 / 18
6	0.51 / 15.21 / 9 0.51 / 15.2 / 9 0.0004 / 13	0.52 / 2.63 / 9 0.52 / 2.63 / 9 0.0004 / 14	0.52 / 2.63 / 9 0.52 / 2.63 / 9 0.0004 / 14	0.52 / 2.63 / 9 0.52 / 2.63 / 9 0.0004 / 14
8	0.47 / 2.76 / 9 0.47 / 2.76 / 9 0.0004 / 13	0.52 / 1.93 / 9 0.52 / 1.93 / 9 0.0004 / 14	0.52 / 1.93 / 9 0.52 / 1.93 / 9 0.0004 / 14	0.52 / 1.93 / 9 0.52 / 1.93 / 9 0.0004 / 14
Type4				
6	0.89 / 1.68 / 2 0.80 / 1.49 / 2 0.81 / 2	0.93 / 1.3 / 2 0.80 / 1.49 / 2 0.82 / 2	0.93 / 1.3 / 2 0.88 / 1.3 / 2 0.83 / 2	0.93 / 1.22 / 2 0.88 / 1.3 / 2 0.83 / 2
10	0.93 / 1.3 / 2 0.90 / 1.22 / 2 0.90 / 2	0.94 / 1.14 / 2 0.93 / 1.14 / 2 0.91 / 2	0.94 / 1.14 / 2 0.93 / 1.14 / 2 0.91 / 2	0.94 / 1.1 / 2 0.93 / 1.14 / 2 0.91 / 2
16	0.94 / 1.07 / 2 0.94 / 1.03 / 2 0.94 / 2	0.94 / 1.05 / 2 0.94 / 1.03 / 2 0.94 / 2	0.94 / 1.03 / 2 0.94 / 1.03 / 2 1.06 / 2	0.94 / 1.02 / 2 0.94 / 1.02 / 2 0.94 / 2
20	0.95 / 1.01 / 2 0.94 / 1.0 / 2 0.94 / 2	0.95 / 1.0 / 2 0.95 / 1.0 / 2 0.94 / 2	0.95 / 1.0 / 2 0.95 / 1.0 / 2 0.94 / 2	0.95 / 1.0 / 2 0.95 / 1.0 / 2 0.94 / 2

Table 4: The average relative accuracy (*mean ac*) C/C^* and average weight (*mean w*) for fixed time bounds. # - number of instances, over which averaging is done. i - i-bound. C - current solution, C^* - optimal solution. 4 Gb, 1 hour limits.

of the BF schemes, especially wAOBF, on grids, pedigrees and type4 instances for weaker heuristics. For example, for grid 75-16-5, $i = 6$ wAOBF finds a considerably more accurate solution within 5 seconds, than the other two schemes. When the heuristics are stronger, there is little difference between the algorithms' performance on grids and pedigrees, while wAOBF is superior on type4 instances.

The weights corresponding to particular time points show several trends. For many instances wAOBF and wR-AOBF reach weight 1.0, i.e. find provably optimal solutions, very fast (e.g. 90-20-5, i=20, within 5 seconds). Often wR-AOBF reaches $w = 1$ faster than wAOBF (e.g. 75-16-5, i=6; bwt3ac, i=2), though when $w > 1$, wAOBF usually finds a better solution for the same time bound and same w (e.g. compare for 75-18-5, i=6, w=1.14, time=10 sec). Moreover, we again see that in many cases BF schemes find solutions of optimal costs for $w > 1$ (e.g. pedigree9, i=16).

Table 4 provides summaries of the anytime performance, presenting for each algorithm the mean time and mean relative accuracy (C/C^*) achieved at a fixed time point for selected values of i-bounds. The closer mean accuracy is to 1, the better. We also show the number of instances, over which we average. Note that for type4 relative accuracy is only computed for 2 instances, since for the rest the optimal solutions are unavailable.

wAOBF vs wR-AOBF. Both schemes solve the same number of instances for all time bounds, but wAOBF often has better accuracy, especially for low i-bound (e.g. type4, $i = 6$, all times). wAOBF mostly has larger mean weight than wR-AOBF for the same time bound (e.g. pedigrees, $i = 6$, time=10 sec: 9.19 vs 1.31), implying that wAOBF spends more time at each iteration, but achieves better accuracy, similar to what we saw in Figure 2.

BF vs BRAOBB. On WCSPs BF schemes found solutions for considerably less instances than BRAOBB (e.g. $i = 6$ and $i = 8$, time=60 to 3600 sec, 9 instances vs 14 by BRAOBB). At a first glance wAOBF and wR-AOBF seem to have considerably better mean accuracy, but that is due to only solving the easiest WCSP problems. On grids and pedigrees wAOBF and wR-AOBF tend to find solutions for more instances than BRAOBB for most time bounds (e.g. grids, time=10 sec, $i = 6$, 23 problems vs 15), while providing comparable mean accuracy. On type4 instances BF schemes mostly achieve slightly better accuracy than BRAOBB.

Impact of heuristic strength. Unsurprisingly, all three schemes find more accurate solutions faster as heuristic strength increases, e.g. on grids for time=10 sec wAOBF has mean accuracy of 0.93 for $i = 10$ and 1.0 for $i = 20$, while solving the same number of instances. However, as i-bound increases, so does time and memory required for computing heuristics, thus sometimes larger i yields fewer solved instances for the same time bounds, e.g. WCSPs, for time=60 sec BRAOBB finds solutions for 18 instances when $i = 2$, but only for 14 when $i = 8$.

Conclusion

In this paper we extended advanced best-first scheme for graphical models into a weighted scheme and evaluated

its performance in comparison with a highly competitive Branch and Bound scheme. Our empirical results show that weighted best-first is valuable in providing relatively fast solutions together with suboptimality bounds. We demonstrated that weighted best-first search schemes should definitely be included in the set of good optimization schemes for solving MPE/MAP tasks. The weight mechanism can mitigate the memory/time trade off in a useful way that can be harnessed into an anytime scheme that not only improves with time, but can also guarantee its level of suboptimality.

References

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

de Givry, S.; Schiex, T.; and Verfaillie, G. 2006. Exploiting tree decomposition and soft local consistency in weighted csp. In *AAAI*, 22–27.

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

Dechter, R., and Pearl, J. 1985. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM* 32:506–536.

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

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

Hart, P.; Nilsson, N.; and 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.

Kask, K., and Dechter, R. 2001. A general scheme for automatic search heuristics from specification dependencies. *Artificial Intelligence* 91–131.

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

Likhachev, M.; Gordon, G.; and Thrun, S. 2003. ARA*: Anytime A* with provable bounds on sub-optimality. *NIPS* 16.

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

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

Nillson, N. J. 1980. *Principles of Artificial Intelligence*. Tioga, Palo Alto, CA.

Nilsson, N. 1982. *Principles of artificial intelligence*. Springer Verlag.

Otten, L., and Dechter, R. 2011. Anytime AND/OR depth first search for combinatorial optimization. In *SOCS*.

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

Richter, S.; Thayer, J.; and Ruml, W. 2010. The joy of forgetting: Faster anytime search via restarting. In *ICAPS*, 137–144.

van den Berg, J.; Shah, R.; Huang, A.; and Goldberg, K. 2011. Anytime nonparametric A*. In *AAAI*.