AND/OR Search for Marginal MAP

Radu Marinescu
IBM Research – Ireland
radu.marinescu@ie.ibm.com

Rina Dechter and **Alexander Ihler**
University of California, Irvine
{dechter,ihler}@ics.uci.edu

Abstract

Marginal MAP problems are known to be very difficult tasks for graphical models and are so far solved exactly by systematic search guided by a join-tree upper bound. In this paper, we develop new AND/OR branch and bound algorithms for marginal MAP that use heuristics extracted from weighted mini-buckets enhanced with message-passing updates. We demonstrate the effectiveness of the resulting search algorithms against previous join-tree based approaches, which we also extend to accommodate high induced width models, through extensive empirical evaluations. Our results show not only orders-of-magnitude improvements over the state-of-the-art, but also the ability to solve problem instances well beyond the reach of previous approaches.

1 INTRODUCTION

Graphical models provide a powerful framework for reasoning with probabilistic and deterministic information. These models use graphs to capture conditional independencies between variables, allowing a concise representation of knowledge as well as efficient graph-based query processing algorithms. Combinatorial maximization (max-inference) or maximum *a posteriori* (MAP) tasks arise in many applications and often can be efficiently solved by search schemes.

The marginal MAP problem distinguishes between maximization variables (called MAP variables) and summation variables (the others). Marginal MAP is $\mathrm{NP}^{\mathrm{PP}}$-complete [1]; it is difficult not only because the search space is exponential in the number of MAP variables, but also because evaluating the probability of any full instantiation of the MAP variables is PP-complete [2]. Algorithmically, this means that the variable elimination operations (max and sum) are applied in a constrained, often more costly order.

State-of-the-art exact algorithms for marginal MAP are typically based on depth-first branch and bound search. A key component of branch and bound search is the heuristic function; while partitioning based heuristics such as *mini-bucket elimination* (MBE) [3] or *mini-cluster-tree elimination* (MCTE) [4, 5] can be applied to the constrained elimination order, the current state-of-the-art is to use a heuristic based on an *exact* solution to an *unconstrained* ordering, introduced by Park and Darwiche [6] and then refined by Yuan and Hansen [7]. These techniques appear to work well when the unconstrained ordering results in a small *induced width*. However, in many situations this is a serious limitation. As one contribution of this work, we extend both algorithms to use mini-bucket partitioning schemes, enabling them to be applied to a wider variety of problem instances.

Importantly however, exact algorithms for pure max- or sum-inference problems have greatly improved in recent years. AND/OR branch and bound (AOBB) algorithms search a significantly smaller search space, exploiting problem structure far more effectively [8]. The partition-based heuristics used by AOBB have also seen significant improvements – for MAP, cost-shifting [9] can be used to tighten the heuristic, while for summation, an extension of MBE called *weighted mini-bucket* (WMB) [10] uses Hölder's inequality and cost-shifting to significantly enhance the likelihood bounds. WMB is closely related to variational bounds such as tree-reweighted belief propagation [11] and conditional entropy decompositions [12], and similar principles have also been used recently to develop message-passing approximations for marginal MAP [13].

Our contributions. In this paper, we develop AND/OR branch and bound search for marginal MAP, using a heuristic created by extending weighted mini-bucket to the constrained order elimination of marginal MAP. We evaluate both a single-pass heuristic, which uses cost-shifting by moment matching (WMB-MM) during construction, and an iterative version that passes messages on the corresponding join-graph (WMB-JG). We show empirically that the new heuristic functions almost always improve over standard mini-bucket, and in many cases give tighter bounds and faster searches than the unconstrained join-tree methods, yielding a far more empowered AND/OR search algorithms.

We demonstrate the effectiveness of the proposed search algorithms against the two previous methods at solving a variety of problem instances derived from the recent PASCAL2 Inference Challenge benchmarks. Our results show not only orders of magnitude improvements over the current state-of-the-art approaches but also the ability to solve many instances that could not be solved before.

Following background and brief overview of earlier work (Sections 2 and 3), Section 4 presents the AND/OR search approach for marginal MAP. Section 5 describes the weighted mini-bucket schemes, Section 6 is dedicated to our empirical evaluation and Section 7 concludes.

2 BACKGROUND

A *graphical model* is a tuple $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, where $\mathbf{X} = \{X_i : i \in V\}$ is a set of variables indexed by set V and $\mathbf{D} = \{D_i : i \in V\}$ is the set of their finite domains of values. $\mathbf{F} = \{\psi_\alpha : \alpha \in F\}$ is a set of discrete positive real-valued local functions defined on subsets of variables, where we use $\alpha \subseteq V$ and $\mathbf{X}_\alpha \subseteq \mathbf{X}$ to indicate the *scope* of function ψ_α, ie, $\mathbf{X}_\alpha = var(\psi_\alpha) = \{X_i : i \in \alpha\}$. The function scopes imply 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. The graphical model $\mathcal{M}$ defines a factorized probability distribution on $\mathbf{X}$, $P(\mathbf{X}) = \frac{1}{Z} \prod_{\alpha \in F} \psi_\alpha$. The *partition function*, Z, normalizes the probability to sum to one.

Let $\mathbf{X}_S$ be a subset of $\mathbf{X}$ and $\mathbf{X}_M = \mathbf{X} \backslash \mathbf{X}_S$ be the complement of $\mathbf{X}_S$. The *Marginal MAP* problem is to find the assignment x^*_M to variables $\mathbf{X}_M$ that maximizes the value of the marginal distribution after summing out variables $\mathbf{X}_S$:

$$x^*_M = \operatorname*{argmax}_{\mathbf{X}_M} \sum_{\mathbf{X}_S} \prod_{\alpha \in F} \psi_\alpha \quad (1)$$

We call $\mathbf{X}_M$ "MAP variables", and $\mathbf{X}_S$ "sum variables".

If $\mathbf{X}_S = \emptyset$ then the problem is also known as maximum *a posteriori* (MAP) inference. The marginal MAP problem is however significantly more difficult. The decision problem for marginal MAP was shown to be NP^{PP}-complete [1], while the decision problem for MAP is only NP-complete [14]. The main difficulty arises because the max and sum operators in Eq. (1) do not commute, which restricts efficient elimination orders to those in which all sum variables $\mathbf{X}_S$ are eliminated before any max variables $\mathbf{X}_M$.

Bucket Elimination (BE) [15] solves the marginal MAP problem exactly by eliminating the variables in sequence. Given a *constrained elimination order* ensuring the sum variables are processed before the max variables, BE partitions the functions into buckets, each associated with a single variable. A function is placed in the bucket of its argument that appears latest in the ordering. BE processes each bucket, from last to first, by multiplying all functions in the current bucket and eliminating the bucket's variable (by summation for sum variables and by maximization for MAP variables), resulting in a new function which is placed in an earlier bucket. The complexity of BE is time and space exponential in the *constrained induced width* w^*_c of the primal graph given a constrained elimination order [15]. BE can be viewed as message passing in a join-tree whose nodes correspond to buckets and which connects nodes a, b if the function generated by a's bucket is placed in b's [16].

Mini-Bucket Elimination (MBE) [3] 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 [3]. MBE processes sum buckets and the max buckets differently. Max mini-buckets (in $\mathbf{X}_M$) are eliminated by maximization, while for variables in $\mathbf{X}_S$, one (arbitrarily selected) mini-bucket is eliminated by summation, while the rest of the mini-bucket are eliminated by maximization. MBE outputs an upper bound on the optimal marginal MAP value. The complexity of the algorithm, which is parametrized by the i-bound, is time and space exponential in i only. When i is large enough (i.e., $i > w^*_c$), MBE coincides with full BE. MBE is often used to generate heuristics for branch and bound search.

Another related approximation with bounded complexity, more similar in structure to join-tree inference, is *Mini-Cluster-Tree Elimination* (MCTE) [5]. In MCTE, we pass messages along the structure of the join-tree, except that when computing a message, rather than combining all the functions in the cluster, we first partition it into mini-clusters, such that each mini-cluster has a bounded number of variables (the i-bound). Each mini-cluster is then processed separately to compute a set of outgoing messages. Like mini-bucket elimination, this procedure produces an upper bound on the results of exact inference, and increasing i typically provides tighter bounds, but at higher computational cost. Thus, both MBE and MCTE allow the user to trade upper bound accuracy for time and space complexity.

3 CURRENT SEARCH METHODS

The current state-of-the-art methods for marginal MAP are based on branch and bound search using specialized heuristics. In particular, Park and Darwiche [6] construct an upper bound on each subproblem using a modified join-tree algorithm along an *unconstrained* elimination order that interleaves the MAP and sum variables. During search, the

Algorithm 1: BBBT for marginal MAP

Input: Graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, i-bound i, unassigned MAP variables $\mathbf{X}_M$, lower bound L, partial assignment to MAP variables $\bar{x}$
Output: Optimal marginal MAP value
if $\mathbf{X}_M = \emptyset$ **then**
  return Solve($\mathcal{M}|_{\bar{x}}$);
else
  $X_k \leftarrow SelectVar(\mathbf{X}_M)$;
  Update MCTE(i) ;
  foreach *value* $x_k \in D_k$ **do**
    Assign X_k to x_k: $\bar{x} \leftarrow \bar{x} \cup \{X_k = x_k\}$;
    $U(\bar{x}) \leftarrow extract(\text{MCTE}(i))$
    if $U(\bar{x}) > L$ **then**
      $L = max(L, \text{BBBT}(i, \mathbf{X}_M \setminus \{X_k\}, L, \bar{x})$;
    $\bar{x} \leftarrow \bar{x} \setminus \{X_k = x_k\}$;
  return L;

join-tree is fully re-evaluated at each node in order to compute upper bounds for all uninstantiated MAP variables simultaneously, which allows the use of dynamic variable ordering. Although this approach provides effective bounds, the computation can be quite expensive. More recently, Yuan and Hansen [7] proposed an incremental evaluation of the join-tree bounds which reduces significantly their computational overhead during search. However, this requires the search algorithm to follow a static variable ordering. In practice, Yuan and Hansen's method proved to be cost effective, considerably outperforming [6]. However, both methods require that the induced width of the unconstrained join tree is small enough to be feasible, which often may not be the case.

3.1 ALGORITHM BBBT

Our first two algorithms, then, can be viewed as generalizations of [6] and [7] schemes for models with high unconstrained induced width. In particular, we use MCTE(i) to approximate the exact, unconstrained join-tree inference to accommodate a maximum clique size defined by the i-bound i. The resulting branch and bound with MCTE(i) heuristics, abbreviated hereafter by BBBT[1], for marginal MAP is given in Algorithm 1.

The algorithm is called initially as BBBT(i, $\mathbf{X}_M$, 0, $\emptyset$), where $\mathbf{X}_M$ are the MAP variables of the input graphical model, and i is the i-bound. The algorithm maintains the best solution found so far, giving a lower bound L on the optimal marginal MAP value. The algorithm searches the simple tree of all partial variable assignments (also called the OR tree). At each step, BBBT uses MCTE(i) to compute an upper bound $U(\bar{x})$ on the optimal marginal MAP extension of the current partial MAP assignment $\bar{x}$ (lines 5-8). If $U(\bar{x}) \leq L$, then the current assignment $\bar{x}$ cannot lead to a better solution and the algorithm can backtrack (line 9). Otherwise, BBBT expands the current assignment by selecting the next MAP variable in a static or dynamic variable ordering (line 4) and recursively solves a set of subproblems, one for each un-pruned domain value. Notice that when $\bar{x}$ is a complete assignment, BBBT calculates its marginal MAP value by solving a summation task over $\mathcal{M}|_{\bar{x}}$, the subproblem defined by the sum variables conditioned on $\bar{x}$ (line 2). Given sufficient resources (high enough i-bound), this can be done by variable elimination, but for consistency with our other algorithms, our implementation uses AND/OR search with caching [18] (see also Section 4). If a better new assignment is found then the lower bound L is updated (line 10).

If MCTE(i) is fully re-evaluated at each iteration, it produces upper bounds for all uninstantiated MAP variables simultaneously. In this case, BBBT can accommodate dynamic variable orderings and can thus be viewed as a generalization of Park and Darwiche [6]. Alternatively, MCTE(i) can be done in an incremental manner as in [7]. In this case BBBT requires a static variable ordering and can be viewed as a generalization of Yuan and Hansen.

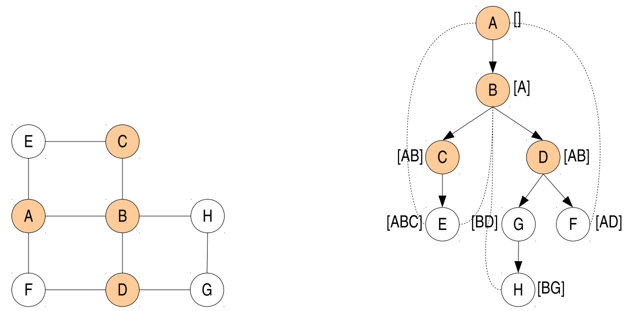
4 AND/OR SEARCH

Significant improvements in search for pure MAP inference have been achieved by using AND/OR search spaces, which often capture problem structure far better than standard OR search methods [18]. In this section, we give an AND/OR search algorithm for marginal MAP. First, we define the *pseudo tree* of the primal graph, which defines the search space and captures problem decomposition.

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

The set of valid pseudo trees for marginal MAP is restricted to those for which the MAP variables form a *start pseudo tree*, a subgraph of pseudo tree $\mathcal{T}$ that has the same root as $\mathcal{T}$. Given a graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$ with primal graph G and pseudo tree $\mathcal{T}$ of G, the *AND/OR search tree* $S_{\mathcal{T}}$ based on $\mathcal{T}$ has alternating levels of OR nodes corresponding to the variables and AND nodes corresponding to the values of the OR parent's variable, with edges weighted according to $\mathbf{F}$. 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* [18]. Merging all context-mergeable nodes yields the *context minimal AND/OR search graph*, denoted $C_{\mathcal{T}}$. The size of $C_{\mathcal{T}}$ is exponential in the induced width of G along a depth-first

[1] For consistency with prior work, we use the name used in [17], (Branch and Bound with Bucket-Tree heuristic) to denote the same algorithm applied to pure MAP queries.



(a) Primal graph (b) Pseudo tree

Figure 1: A simple graphical model.

traversal of $\mathcal{T}$ (i.e., the constrained induced width) [18].

A *solution tree* $\hat{x}$ of $C_{\mathcal{T}}$ is a subtree that: (1) contains the root of $C_{\mathcal{T}}$; (2) if an internal OR node $n \in C_{\mathcal{T}}$ is in $\hat{x}$, then n is labeled by a MAP variable and exactly one of its children is in $\hat{x}$; (3) if an internal AND node $n \in C_{\mathcal{T}}$ is in $\hat{x}$ then all its OR children labeled by MAP variables are in $\hat{x}$.

Each node n in $C_{\mathcal{T}}$ can be associated with a *value* $v(n)$; for MAP variables, $v(n)$ captures the optimal marginal MAP value of the conditioned subproblem rooted at n, while for sum variables it is the conditional likelihood of the subproblem. Clearly, $v(n)$ can be computed recursively based on the values of n's successors: OR nodes by maximization or summation (for MAP or sum variables, respectively), and AND nodes by multiplication.

Example 1 *Figure 1(a) shows a simple graphical model with $\mathbf{X}_M = \{A, B, C, D\}$ and $\mathbf{X}_S = \{E, F, G, H\}$. Figure 2 displays the context minimal AND/OR search graph based on the constrained pseudo tree from Figure 1(b) (the contexts are shown next to the pseudo tree nodes). It is easy to see that the MAP variables form a start pseudo tree. A solution tree corresponding to the MAP assignment $(A = 0, B = 1, C = 1, D = 0)$ is indicated in red.*

Algorithm 2 describes the AND/OR Branch and Bound (AOBB) for marginal MAP. We use the notation that $\bar{x}$ is the current partial solution and the table *Cache*, indexed by node contexts, holds the partial search results. The algorithm assumes that variables are selected statically according to a valid pseudo tree $\mathcal{T}$. A heuristic $f(\bar{x})$ calculates an upper bound on the optimal marginal MAP extension of $\bar{x}$.

If the set $\mathbf{X}$ is empty, the result is trivially computed (line 1). Else, AOBB selects the next variable X_k in $\mathcal{T}$ and if the corresponding OR node is not found in cache, it expands it and iterates over its domain values to compute the OR value $v(X_k)$ (lines 7-22). Notice that if X_k is a MAP variable, then AOBB attempts to prune unpromising domain values by comparing the upper bound $f(\bar{x})$ of the current partial solution tree $\bar{x}$ to the current best lower bound L which is maintained by the root node of the search space (line 10). For each domain value x_k, the problem rooted at AND node $\langle X_k, x_k \rangle$ is decomposed into q independent subproblems $\mathcal{M}_l = \langle \mathbf{X}_l, \mathbf{D}_l, \mathbf{F}_l \rangle$, one for each child X_l of X_k in $\mathcal{T}$. These problems are then solved independently and their results accumulated by the AND node value $v(X_k, x_k)$ (lines 12-13 and 18-19). After trying all possible values of variable X_k, the marginal MAP value of the subproblem rooted by X_k is $v(X_k)$ if X_k is a MAP variable, and is returned (line 21). If X_k is a sum variable, then $v(X_k)$ holds the likelihood value of that conditioned subproblem (line 22). The optimal marginal MAP value to the original problem is returned by the root node.

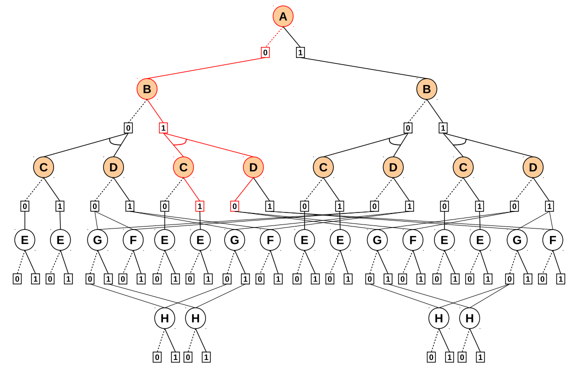


Figure 2: AND/OR search spaces for marginal MAP.

AOBB typically computes its heuristic $f(\cdot)$ using a mini-bucket bounding scheme (see Section 2), which can be precompiled along the reverse order of a depth-first traversal of the pseudo tree (which is a valid constrained elimination order). Unfortunately, our AOBB cannot use the join-tree/MCTE(i) based heuristics of Section 3, since these are compiled along an unconstrained variable ordering which is not compatible, in general, with the constrained pseudo tree that drives the AOBB search order. For this reason, we next turn to improving our mini-bucket bounds.

5 MINI-BUCKET FOR MARGINAL MAP

In this section, we develop improved, constrained order mini-bucket bounds compatible with AOBB search. MBE has been effective for pure MAP, but less so for marginal MAP; previously, its bounds appeared to be far less accurate than the unconstrained join-tree bounds [6, 7]. Therefore, we revisit the mini-bucket approach and enhance it with recent iterative cost-shifting schemes [10, 9, 13].

5.1 WEIGHTED MINI-BUCKETS

Weighted mini-bucket elimination (WMB) [10] is a recent algorithm developed for likelihood (summation) tasks that replaces the naïve mini-bucket bound with Hölder's inequality. For a given variable X_k, the mini-buckets Q_{kr} associated with X_k are assigned a non-negative *weight* $w_{kr} \geq 0$, such that $\sum_r w_{kr} = 1$. Then, each mini-

Algorithm 2: AOBB for marginal MAP

Input: Graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, pseudo tree $\mathcal{T}$, partial solution tree $\bar{x}$, heuristic evaluation function $f(\bar{x})$
Output: Optimal marginal MAP value

```
if X = ∅ then return 1;
;
else
    X_k ← SelectVar(X) according to T;
    if v(X_k) ∈ Cache then return v(X_k);
    ;
    if X_k ∈ X_M then v(X_k) ← −∞;
    ;
    else v(X_k) ← 0;
    ;
    foreach value x_k ∈ D_k do
        if X_k ∈ X_M then
            x̄ ← x̄ ∪ {X_k = x_k};
            if f(x̄) > L then
                v(X_k, x_k) ← 1;
                foreach child X_l of X_k in T do
                    v(X_k, x_k) ← v(X_k, x_k) × AOBB(M_l);
            else v(X_k, v_j) ← −∞;
            ;
            x̄ ← x̄ \ {X_k = x_k};
        else
            v(X_k, x_k) ← 1;
            foreach child X_l of X_k in T do
                v(X_k, x_k) ← v(X_k, x_k) × AOBB(M_l);
        val ← w(X_k, x_k) × v(X_k, x_k);
        if X_k ∈ X_M then v(X_k) ← max(v(X_k), val);
        ;
        else v(X_k) ← v(X_k) + val;
        ;
    Cache ← Cache ∪ v(X_k);
    return v(X_k)
```

bucket r is eliminated using a weighted or power sum, $(\sum_{X_k} f(X)^{1/w_{kr}})^{w_{kr}}$. It is useful to note that w_{kr} can be interpreted as a "temperature"; if $w_{kr} = 1$, it corresponds to a standard summation, while if $w_{kr} \to 0$, it instead corresponds to a maximization over X_k. Thus, standard mini-bucket corresponds to choosing one mini-bucket r with $w_{kr} = 1$, and the rest with weight zero.

Weighted mini-bucket is closely related to variational bounds on the likelihood, such as conditional entropy decompositions [12] and tree-reweighted belief propagation (TRBP) [11]. The single-pass algorithm of Liu and Ihler [10] mirrors standard mini-bucket, except that within each bucket a cost-shifting (or reparameterization) operator is performed, which matches the marginal beliefs (or "moments") across mini-buckets to improve the bound.

The temperature viewpoint of the weights enables us to apply a similar procedure for marginal MAP. In particular, for $X_k \in \mathbf{X}_S$, we enforce $\sum_r w_{kr} = 1$, while for $X_k \in \mathbf{X}_M$, we take $\sum_r w_{kr} = 0$ (so that $w_{kr} = 0$ for all r). The resulting algorithm, listed in Algorithm 3, treats MAP and sum variables differently: for sum variables it mirrors [10],

Algorithm 3: WMB-MM(i)

Input: Graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, MAP variables $\mathbf{X}_M$, constrained ordering $o = X_1, \ldots, X_n$, i-bound i
Output: Upper bound on optimal marginal MAP value

```
foreach k ← n downto 1 do
    // Create bucket B_k and mini-buckets Q_kr
    B_k ← {ψ_α | ψ_α ∈ F, X_k ∈ var(ψ_α)}; F ← F \ B_k;
    Let Q = {Q_k1, ..., Q_kR} be an i-partition of B_k;
    foreach r = 1 to R do
        ψ_kr = ∏_{ψ∈Q_kr} ψ; Y_r = vars(Q_kr) \ X_k;
    // Moment Matching
    if X_k ∈ X_S then
        Assign mini-bucket r weight w_kr > 0, st Σ_r w_kr = 1;
        μ_r = Σ_{Y_r} (ψ_kr)^{1/w_kr}; μ = ∏_r (μ_r)^{w_kr};
        Update ψ_kr = ψ_kr · (μ/μ_r)^{w_kr};
    else
        μ_r = max_{Y_r} ψ_kr; μ = (∏_r μ_r)^{1/R};
        Update ψ_kr = ψ_kr · (μ/μ_r);
    // Downward Messages (eliminate X_k)
    foreach r = 1 to R do
        if X_k ∈ X_S then λ_kr ← (Σ_{X_k} (ψ_kr)^{1/w_kr})^{w_kr};
        ;
        else λ_kr ← max_{X_k} ψ_kr;
        ;
        F ← F ∪ {λ_kr};
return ∏_{ψ∈F} ψ
```

while taking the zero-temperature limit for MAP variables we obtain the max-marginal matching operations described for pure MAP problems in [9]. This mirrors the result of Weiss et al. [19], that the linear programming relaxation for MAP corresponds to a zero-temperature limit of TRBP.

5.2 ITERATIVE UPDATES

While the single-pass algorithm is often very effective, we can further improve it using iterative updates. The iterative weighted mini-bucket algorithm [10], alternates between downward passes, which look like standard mini-bucket with cost-shifting, and upward passes, which compute messages used to "focus" the cost shifting in the next downward pass. The algorithm can be viewed as message passing on a join graph defined by the mini-bucket cliques, and is listed in Algorithm 4.

Standard MBE computes "downward" messages $\lambda_{kr} = m_{a \to c}$ from each clique $a = (kr)$ (the rth mini-bucket for variable X_k) to a single child clique $c = \text{ch(a)}$. For the iterative version, we also compute "upward" messages $m_{c \to a}$ from clique c to its parent cliques $a \in \text{pa}(c)$. For $w_a > 0, w_c > 0$, these upward messages are given by [10]:

$$m_{c \to a} \propto \Big[\sum_{Y_c \setminus Y_a} (\psi_c \, m_{\sim c})^{1/w_c} \, m_{a \to c}^{-1/w_a} \Big]^{w_a}$$

where $\psi_c = \prod_{\psi \in Q_c} \psi$ are the model factors assigned to clique c, and $m_{\sim c}$ is the product of all messages into c.

These upward messages are used during the cost-shifting updates of X_k in later downward passes:

$$\forall r,\ \mu_{kr} \propto \sum_{Y_{kr}} (\psi_{kr}\, m_{\sim kr})^{1/w_{kr}}; \quad \mu = \Big(\prod_r (\mu_{kr})^{w_{kr}}\Big)^{1/w_k}$$

$$\forall r,\ \psi_{kr} \leftarrow \psi_{kr}\, \Big(\frac{\mu}{\mu_{kr}}\Big)^{\gamma\, w_{kr}}$$

in which we include the upward message $m_{\mathrm{ch}(kr)\to kr}$ in the marginals μ_{kr} being matched, and define $w_k = \sum_r w_{kr}$. These fixed-point updates are not guaranteed to be monotonic; to assist convergence, we also include a "step size" $\gamma \leq 1$. By initializing the upward messages $m_{\mathrm{ch}(c)\to c} = 1$ and taking $\gamma = 1/t$, the first iteration of Alg. 4 corresponds exactly to WMB-MM (Alg. 3).

For marginal MAP, we can take the limit as some weights $w_a = \epsilon \to 0$; then, when both a and $c = \mathrm{ch}(a)$ correspond to MAP variables we have

$$m_{c\to a} \propto \big[\max_{Y_c\setminus Y_a} (\psi_c m_{\sim c})\, m_{a\to c}^{-1}\big]$$

$$\mu_a = \max_{Y_a}(\psi_a\, m_{\sim a}); \quad \mu = \Big(\prod_a \mu_a\Big)^{\frac{1}{|\mathcal{Q}|}}; \quad \psi_a \leftarrow \psi_a\, \Big(\frac{\mu}{\mu_a}\Big)^{\gamma}$$

When clique a corresponds to a sum variable and clique $c = \mathrm{ch}(a)$ to a MAP variable, we take $w_c = \epsilon$ to give:

$$m_{c\to a} \propto \Big[\sum_{Y_c\setminus Y_a} \sigma_\epsilon(\psi_c m_{\sim c})\, m_{a\to c}^{-1/w_a}\Big]^{w_a}$$

$$\sigma_\epsilon(f(X)) = \big(f(X)/\max_x f(x)\big)^{1/\epsilon}$$

When $\epsilon \to 0$, σ_ϵ becomes an indicator function of the maximizing arguments of f, "focusing" the matching step at parent a on configurations relevant to the max values of child c. The resulting algorithm is also closely related to a (tree-reweighted) mixed-product belief propagation algorithm for marginal MAP [13]. Unfortunately, directly taking $\epsilon = 0$ can cause the objective function to be highly non-smooth, and lead to undesirable, non-monotonic fixed-point updates. To alleviate this, in practice we use a schedule $\epsilon = 1/t$ to decrease the temperature over iterations.

Algorithm 4: WMB-JG(i)

Input: Graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}\rangle$, constrained ordering $o = X_1, \ldots, X_n$, i-bound i, number of iterations T
Output: Upper bound on optimal marginal MAP value

```
for t = 1 to T do
    // Downward pass with moment matching
    foreach k ← n downto 1 do
        Let Q = {Q_a | a = kr} be the mini-buckets of B_k;
        foreach Q_a ∈ Q do
            ψ_a = ∏_{ψ∈Q_a} ψ;
            Y_a = vars(Q_a) \ X_k;
            m_{~a} = m_{ch(a)→a} · ∏_{p∈pa(a)} m_{p→a};
        if X_k ∈ X_S then
            foreach Q_a ∈ Q do μ_a = Σ_{Y_a} (ψ_a m_{~a})^{1/w_a};
            ;
            μ = ∏_{Q_a∈Q} (μ_a)^{w_a};
            foreach Q_a ∈ Q do Update ψ_a = ψ_a · (μ/μ_a)^{w_a};
            ;
        else
            foreach Q_a ∈ Q do μ_a = max_{Y_a} (ψ_a m_{~a});
            ;
            μ = ∏_{Q_a∈Q} (μ_a)^{1/|Q|};
            foreach Q_a ∈ Q do Update ψ_a = ψ_a · (μ/μ_a);
            ;
        foreach Q_a ∈ Q, c = ch(a), do
            if X_k ∈ X_S then
                m_{a→c} = (Σ_{X_k} (ψ_a m_a)^{1/w_a} )^{w_a};
            else
                m_{a→c} = max_{X_k} (ψ_a m_a);
    // Backward pass
    foreach k ← 1 to n do
        Let Q = {Q_c | c = kr} be the mini-buckets of B_k;
        foreach Q_c ∈ Q and a ∈ pa(c), with c=kr, a=js do
            Y = vars(Q_c) \ vars(Q_a);
            if X_k ∈ X_S and X_j ∈ X_S then
                m_{c→a} =
                (Σ_Y (ψ_c m_{~c})^{1/w_c} · (m_{a→c})^{-1/w_a})^{w_a};
            if X_k ∈ X_M and X_j ∈ X_M then
                m_{c→a} = (max_Y (ψ_c m_{~c}) · (m_{a→c})^{-1});
            if X_k ∈ X_S and X_j ∈ X_M then
                m_{c→a} =
                (Σ_Y σ_ε(ψ_c m_{~c}) · (m_{a→c})^{-1/w_a})^{w_a};
return upper bound from B_1;
```

6 EXPERIMENTS

We empirically evaluate the proposed branch and bound algorithms on problem instances derived from benchmarks used in the PASCAL2 Inference Challenge [20] as well as the original instances from [7].

Algorithms. We consider three AND/OR branch and bound search algorithms (Section 4): AOBB guided by basic MBE(i) heuristics (denoted AOBB), AOBB guided by heuristics from WMB-MM(i) (denoted AOBB-MM), and AOBB guided by heuristics from WMB-JG(i) (denoted by AOBB-JG), respectively. All of the mini-bucket heuristics were generated in a pre-processing phase, prior to search. The weighted schemes used uniform weights. In addition, we also tested two OR branch and bound schemes guided by MCTE(i) heuristics, denoted BBBTi and BBBTd, respectively. BBBTi performs MCTE(i) incrementally, while BBBTd fully re-evaluates MCTE(i) at each iteration.

We compare all five algorithms against each other and against the current state-of-the-art branch and bound with incremental join-tree upper bounds [7], denoted by YUAN, along with the original approach by Park and Darwiche [6], denoted by PARK. Algorithms BBBTd and PARK

Table 1: Upper bounds (log scale) and CPU time (sec) for a typical set of instances. $i = 10$ and $i = 20$.

instance	i	MBE	WMB-MM	MCTE	WMB-JG			JT
					5 iterations	10 iterations	100 iterations	
(n, m, k, w_c^*, w_u^*)		UB/time	UB/time	UB/time	UB/time	UB/time	UB/time	UB/time
cpcs360	10	5.9607/0.03	-0.0228/0.04	4.3008/0.16	-0.0353/0.34	-0.0360/0.75	-0.0363/6.71	**-0.0468**/4.73
(360,25,2,24,20)	20	2.4871/11.4	-0.0402/1.36	-0.0468/3.45	-0.0465/47.2	-0.0467/145	**-0.0468**/1339	
2-17-s-s	10	-44.2658/0.02	-49.5830/0.01	-40.3520/0.06	-55.4555/0.12	-55.5996/0.24	-55.6633/2.44	-55.5170/0.31
(228,69,2,20,15)	20	-55.5083/3.54	-55.5082/0.30	-55.5170/0.36	-55.7433/5.38	-55.7436/12.7	**-55.7437**/197	
or-chain-10.fg-s	10	-10.5621/0.01	-13.2118/0.01	-6.4940/0.1	-17.2899/0.10	-18.7859/0.18	-21.3428/1.71	-21.0314/4.29
(453,135,2,22,18)	20	-18.2977/3.56	-19.3815/0.33	-9.8054/0.5	**-21.3600**/5.46	**-21.3600**/11.8	**-21.3600**/137	
cpcs422	10	10.026/0.61	-1.3206/0.88	7.0553/1.62	-1.3764/4.06	-1.3878/8.67	-1.4275/75.1	**-1.4982**/41.6
(422,74,2,74,23)	20	7.9245/18.5	-1.4427/9.29	-0.1331/9.78	-1.4545/191	-1.4554/371	-1.4718/2353	
2-2-s-l	10	-57.0433/0.04	-75.1643/0.03	-39.1568/0.07	-80.9224/0.17	-81.5811/0.33	-81.9044/3.56	-81.5883/0.14
(227,68,2,73,14)	20	-67.2268/5.52	-80.4492/1.68	-81.5883/0.17	-82.0108/25.0	-82.1039/58.6	**-82.1960**/400	
or-chain-18.fg-l	10	-2.7317/0.01	-2.5168/0.02	-2.3325/0.42	-6.4279/0.36	-7.1655/0.51	-11.1244/3.78	-11.4487/0.43
(890,267,2,25,8)	20	-10.2463/0.88	**-11.4534**/0.07	-11.4487/0.46	**-11.4534**/1.48	**-11.4534**/2.97	**-11.4534**/31.7	

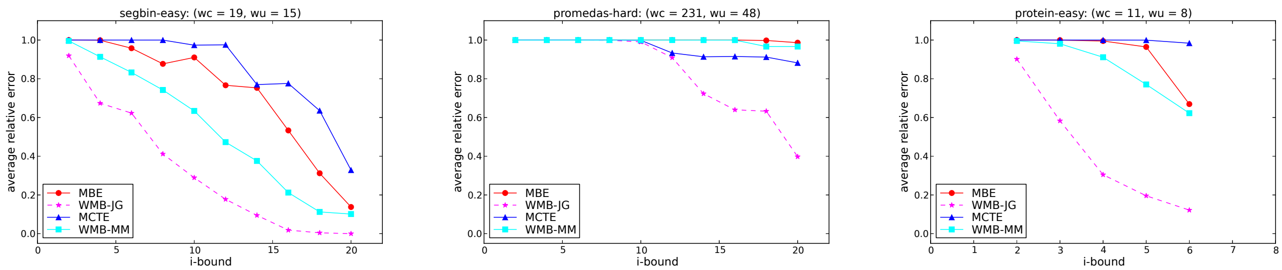


Figure 3: Average relative error (w.r.t. tightest upper bound) as a function of i-bound. WMB-JG(i) ran for 10 iterations.

use a dynamic variable ordering and select the next MAP variable whose domain values have the most asymmetric bounds. Algorithms BBBTi and YUAN are restricted to a static variable ordering that corresponds to a post-order traversal of the underlying join-tree. The pseudo trees guiding the AND/OR algorithms were obtained by a modified min-fill heuristic [18] that constrained the MAP variables to form a start pseudo tree. All algorithms were implemented in C++ (64-bit) and the experiments were run on a 2.6GHz 8-core processor with 80 GB of RAM.

Benchmarks. Our problem instances were derived from three PASCAL2 benchmarks: `segbin` (image segmentation), `protein` (protein side-chain interaction) and `promedas` (medical diagnosis expert system). For each network, we generated two marginal MAP problem instances with m MAP variables, as follows: an `easy` instance such that the MAP variables were selected as the first m variables from a breadth-first traversal of a pseudo tree obtained from a hypergraph decomposition of the primal graph (ties were broken randomly) [8], and a `hard` instance where the MAP variables were selected uniformly at random. The `easy` instances were designed such that problem decomposition is maximized and the constrained and unconstrained elimination orders are relatively close to each other, thus having comparable induced widths. In contrast, the `hard` instances tend to have very large constrained induced widths. We selected 30% of the variables as MAP variables. In total we evaluated 120 problem instances (20 `easy` and 20 `hard` instances per benchmark).

In all experiments we report total CPU time in seconds and number of nodes visited during search. We also record the problem parameters: number of variables (n), max domain size (k), number of MAP variables (m), and the constrained (w_c^*) and unconstrained (w_u^*) induced widths. The best performance points are highlighted. In each table, 'oom' stands for out-of-memory, while '-' denotes out-of-time.

Results: quality of the upper bounds. We compare the accuracy of the upper bounds obtained by the mini-bucket schemes MBE(i), WMB-MM(i) and WMB-JG(i) against those produced by the unconstrained join-tree scheme, denoted JT, and its generalization MCTE(i).

Table 1 shows results on a typical set of problem instances from both `easy` (top 3) and `hard` (bottom 3) categories, for two values of i-bound: $i = 10, 20$. For every problem instance, for each algorithm we report the upper bound obtained (lower values are better) and CPU time in seconds. The iterative scheme WMB-JG(i) ran for 5, 10 or 100 iterations, respectively. We see clearly that for all instances WMB-MM(i) provides significantly tighter upper bounds than the corresponding pure MBE(i) in a comparable CPU time (see also Figure 3). On the other hand, WMB-JG(i) is able to converge to the most accurate bounds in 4 out of 6 cases, but at a much higher computational cost. JT bounds are typically tighter than those produced by MCTE(i) and MBE(i) which is consistent with previous studies [6, 7].

In Figure 3 we plot the average relative error with respect to the tightest upper bound obtained, as a function of the i-bound. Since, the JT bounds were available only on a relatively small fraction of the instances tested, they are omitted for clarity. We observe that if given enough time WMB-JG(i) is superior to all its competitors, especially

Table 2: CPU time (sec) and nodes for the protein instances. Time limit 1 hour. WMB-JG(i) ran for 10 iterations.

instance (n, m, kw_c^*, w_u^*)	algorithm	$i=2$ time	$i=2$ nodes	$i=3$ time	$i=3$ nodes	$i=4$ time	$i=4$ nodes	$i=5$ time	$i=5$ nodes	$i=6$ time	$i=6$ nodes	YUAN PARK time	YUAN PARK nodes
protein easy instances													
	AOBB		-		-		-		-		-		
pdb1a1x	AOBB-JG	539	1746192	**85**	314801	164	3415	3067	746		-		oom
(95,28,81,14,14)	AOBB-MM		-		-	601	7625110	709	9004715	2087	316563		oom
	BBBTd		-		-		-		-		-		
	BBBTi		-		-		-		-		-		
	AOBB		-	1533	12650401	379	1505951	228	169618	753	274565		
pdb1a62	AOBB-JG		-	**13**	35	62	35	523	35	2228	35		oom
(105,31,81,13,10)	AOBB-MM	697	2437932	169	359560	114	135525	138	181286	112	1107		oom
	BBBTd		-		-		-		-		-		
	BBBTi		-		-		-		-		-		
	AOBB		-		-		-		-		-		
pdb1ad2	AOBB-JG		-	**76**	1355	227	431	3368	424				oom
(177,53,81,12,9)	AOBB-MM		-		-	983	838218	211	13902		-		oom
	BBBTd		-		-		-		-		-		
	BBBTi		-		-		-		-		-		
	AOBB	61	119726	9	5483	**4**	735	21	283	154	48		
pdb1aho	AOBB-JG	6	6581	**4**	365	19	271	65	17	1251	17	299	55
(54,16,81,7,6)	AOBB-MM	49	19890	10	3274	8	2057	7	593	44	17	963	16
	BBBTd	7	1224	6	128	28	26	165	29	426	17		
	BBBTi	77	291321	949	1151691	345	35506		-	356	4679		

for larger i-bounds. However, if time is bounded, then WMB-MM(i) provides a cost-effective alternative. Notice also that when the gap between the constrained and unconstrained induced width is very large, then MCTE(i) provides more accurate bounds than MBE(i) and WMB-MM(i) (eg, promedas `hard`), because MCTE(i) does less partitioning in this case. When the gap is relatively small, then the mini-bucket based bounds are often superior to the MCTE(i) ones for the same i-bound (eg, segbin `easy`).

Results: comparison with state-of-the-art search. Tables 2 and 3 report CPU time in seconds and number of nodes expanded by each search algorithm on a subset of instances from the protein and promedas benchmarks. The columns are indexed by the i-bound and the time limit was set to 1 hour. WMB-JG(i) ran for 10 iterations. We can see clearly that AOBB-JG(i) is the overall best performing algorithm, especially for relatively small i-bounds. For example, on the `pdb1a62`, AOBB-JG(3) proves optimality in 13 seconds while AOBB(3) and AOBB-MM(3) finish in 1533 and 169 seconds, respectively. The search space explored by AOBB-JG(3) is also dramatically smaller than those explored by AOBB(3) or AOBB-MM(3). Therefore, the much stronger heuristics generated by WMB-JG(i) translate into impressive time savings. When the i-bound increases, the accuracy ceases to offset the computational overhead and the running time of AOBB-JG(i) increases (e.g., `pdb1aho`). In this case, AOBB-MM(i) is a cost-effective alternative, with reduced overhead for precompiling the heuristic (see also Figure 4 for a profile of the CPU time of the algorithms across all benchmarks). The performance of algorithms YUAN and PARK is quite poor in this domain due to the relatively large unconstrained induced widths, which prevent computation of their heuristic. In contrast, BBBTi/BBBTd with relatively higher i-bounds are sometimes competitive and are able to solve more problem instances than YUAN/PARK.

For completeness, we also tested on the Bayesian net-

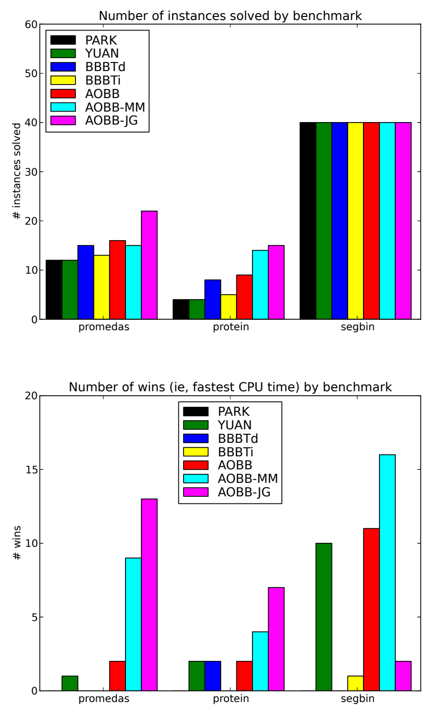


Figure 5: Number of instances solved (top) and number of wins (bottom) by benchmark.

works from [7] (results omitted for space). We observed that all of our proposed algorithms were competitive with YUAN/PARK, but due to the relatively small unconstrained induced widths on these problems, very accurate join-tree heuristics could be computed. Thus, there was very little room for improvement by the new methods.

Summary of the experiments. Figure 4 plots the number of problem instances solved from each benchmark (top) and the median CPU time (bottom) as a function of the i-bound. Clearly, AOBB-JG solves the largest number of in-

Table 3: CPU time (sec) and nodes for the promedas instances. Time limit 1 hour. WMB-JG(i) ran for 10 iterations.

instance (n, m, k, w_c^*, w_u^*)	algorithm	$i=4$ time	$i=4$ nodes	$i=6$ time	$i=6$ nodes	$i=10$ time	$i=10$ nodes	$i=14$ time	$i=14$ nodes	$i=18$ time	$i=18$ nodes	$i=20$ time	$i=20$ nodes	PARK YUAN time	PARK YUAN nodes
promedas easy instances															
	AOBB		-		-	65	6242529	14	1871710	4	471708	7	235860		
or-chain-4.fg-e	AOBB-JG		-	1046	75598793	9	1045873	55	5457626	6	208	19	1144		oom
(691,207,2,33,26)	AOBB-MM		-		-	116	7354956	8	991915	**1**	156030	**1**	73526		oom
	BBBTd		-		-	579	39989	132	4624	233	1900	425	1285		
	BBBTi		-		-		-	394	2001912		-		-		
	AOBB	447	67968093	64	12082065	3	518292	1	162224	1	1920	2	0		
or-chain-17.fg-e	AOBB-JG	38	3943341	57	8830508	**0**	72575	**0**	6940	3	160	6	160	87	159
(531,159,2,20,18)	AOBB-MM	238	26609470	65	9743803	2	306313	**0**	45462	**0**	757	**0**	521	3	162
	BBBTd		-		-	103	5520	85	2921	125	1363	148	633		
	BBBTi		-		-		-	5	61467	10	29232	12	25588		
	AOBB		-		-		-		-		-		-		
or-chain-22.fg-e	AOBB-JG		-		-		-	**2118**	183274481		-		-		oom
(1044,313,2,72,59)	AOBB-MM		-		-		-		-		-		-		oom
	BBBTd		-		-		-		-		-		-		
	BBBTi		-		-		-		-		-		-		
promedas hard instances															
	AOBB		-		-		-		-		-	2254	124886725		
or-chain-4.fg-h	AOBB-JG		-		-	192	5529085	**11**	555059	21	377992	66	215655		oom
(691,207,2,140,28)	AOBB-MM		-		-	752	17706171	304	13152476	188	5662611	78	2134464		oom
	BBBTd		-		-		-		-	1810	12397		-		
	BBBTi		-		-		-		-		-		-		
	AOBB		-		-		-		-		-		-		
or-chain-8.fg-h	AOBB-JG		-		-		-		-		-	**1786**	31316917		oom
(1195,358,2,255,39)	AOBB-MM		-		-		-		-		-		-		oom
	BBBTd		-		-		-		-		-		-		
	BBBTi		-		-		-		-		-		-		
	AOBB		-		-	67	7544343	12	1282228	13	1556793	11	606211		
or-chain-17.fg-h	AOBB-JG		-		-	42	3992210	**3**	212839	8	230955	29	169192	259	159
(531,159,2,72,18)	AOBB-MM		-		-		-		-	7	793696	287	9274776	4	439
	BBBTd		-		-		-	412	12954	861	6003	1931	4649		
	BBBTi		-		-	477	5618175	61	494659	54	136679	106	126093		

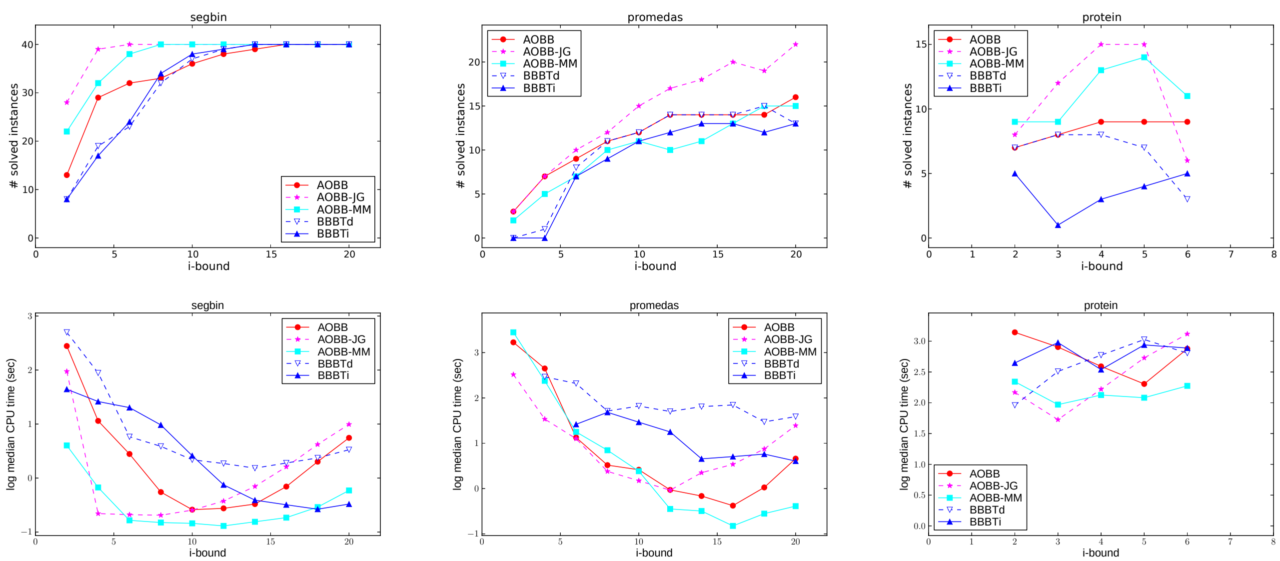


Figure 4: Number of instances solved (top) and median CPU time (bottom) as a function of i-bound for the segbin, promedas and protein instances. Time limit 1 hour. WMB-JG(i) ran for 10 iterations.

stances across i-bounds. Moreover, the running time profile shows that AOBB-JG is faster at lower i-bounds due to more accurate heuristics, while AOBB-MM is faster at higher i-bounds due to reduced overhead. Figure 5 summarizes the total number of instances solved as well as the total number of wins (defining a 'win' as the fastest time) across the benchmarks, for all competing algorithms. Overall, we see that the proposed search algorithms consistently solve more problems and in many cases are significantly faster than the current approaches.

In summary, based on our empirical evaluation, we can conclude that:

- Cost-shifting (especially the iterative version) tightened significantly the MBE bounds for marginal MAP. This yielded considerably faster AOBB search.

- The AOBB algorithms with improved mini-bucket heuristics outperformed in many cases the previous search methods guided by join-tree based heuristics.

7 CONCLUSION

In this paper, we develop AND/OR branch and bound search algorithms for marginal MAP that use heuristics extracted from weighted mini-buckets with cost-shifting. We evaluate both a single-pass version of the heuristic with cost-shifting by moment matching as well as an iterative version that passes messages on the corresponding join-graph. We demonstrate the effectiveness of our proposed search algorithms against previous unconstrained join-tree based methods, which we also extend to apply to high induced-width models, through extensive empirical evaluations on a variety of benchmarks. Our results show not only orders of magnitude improvements over the current state-of-the-art, but also the ability to solve many instances that could not be solved before.

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

References

[1] J. Park. MAP complexity results and approximation methods. In *Uncertainty in Artificial Intelligence (UAI)*, pages 388–396, 2002.

[2] D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1-2):273–302, 1996.

[3] R. Dechter and I. Rish. Mini-buckets: A general scheme of approximating inference. *Journal of ACM*, 50(2):107–153, 2003.

[4] R. Mateescu, K. Kask, V. Gogate, and R. Dechter. Join-graph propagation algorithms. *Journal of Artificial Intelligence Research*, 37:279–328, 2010.

[5] R. Dechter, K. Kask, and J. Larrosa. A general scheme for multiple lower bound computation in constraint optimization. In *Principles and Practice of Constraint Programming*, pages 346–360, 2001.

[6] J. Park and A. Darwiche. Solving MAP exactly using systematic search. In *Uncertainty in Artificial Intelligence (UAI)*, pages 459–468, 2003.

[7] C. Yuan and E. Hansen. Efficient computation of jointree bounds for systematic MAP search. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1982–1989, 2009.

[8] R. Marinescu and R. Dechter. AND/OR branch-and-bound search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173(16-17):1457–1491, 2009.

[9] A. Ihler, N. Flerova, R. Dechter, and L. Otten. Join-graph based cost-shifting schemes. In *Uncertainty in Artificial Intelligence (UAI)*, pages 397–406, 2012.

[10] Q. Liu and A. Ihler. Bounding the partition function using hölder's inequality. In *International Conference on Machine Learning (ICML)*, pages 849–856, 2011.

[11] M. Wainwright, T. Jaakkola, and A. Willsky. A new class of upper bounds on the log partition function. *IEEE Trans. Info. Theory*, 51(7):2313–2335, July 2005.

[12] A. Globerson and T. Jaakkola. Approximate inference using conditional entropy decompositions. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 130–138, 2007.

[13] Q. Liu and A. Ihler. Variational algorithms for marginal MAP. *Journal of Machine Learning Research*, 14:3165–3200, 2013.

[14] S. E. Shimony. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, 2(68):399–410, 1994.

[15] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

[16] K. Kask, R. Dechter, J. Larrosa, and A. Dechter. Unifying cluster-tree decompositions for reasoning in graphical models. *Artificial Intelligence*, 166 (1-2):165–193, 2005.

[17] R. Marinescu, K. Kask, and R. Dechter. Systematic vs non-systematic algorithms for solving the MPE task. In *Uncertainty in Artificial Intelligence (UAI)*, pages 394–402, 2003.

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

[19] Y. Weiss, C. Yanover, and T. Meltzer. MAP estimation, linear programming and belief propagation with convex free energies. In *Uncertainty in Artificial Intelligence (UAI)*, pages 416–425, 2007.

[20] G. Elidan, A. Globerson, and U. Heinemann. PASCAL 2011 probabilistic inference challenge. http://www.cs.huji.ac.il/project/PASCAL/, 2012.