Beyond Static Mini-Bucket: Towards Integrating with Iterative Cost-Shifting Based Dynamic Heuristics

William Lam, Kalev Kask, Rina Dechter, and Alexander Ihler
Donald Bren School of Information and Computer Sciences
University of California, Irvine
{willmlam,kkask,dechter,ihler}@ics.uci.edu

Abstract

We explore the use of iterative cost-shifting as a dynamic heuristic generator for solving MPE in graphical models via Branch and Bound. When mini-bucket elimination is limited by its memory budget, it may not provide good heuristics. This can happen often when the graphical model has a very high induced width with large variable domain sizes. In addition, we explore a hybrid setup where both MBE and the iterative cost-shifting bound are used in a combined heuristic. We compare these approaches with the most advanced statically generated heuristics.

Introduction

Combinatorial optimization tasks are prevalent in many applications and can be formulated as a *most probable explanation* (MPE) query in graphical models. Graphical models are a widely-used framework for tasks not limited to optimization and provide structure to a problem, which can be used to reason about it and perform efficient computation (Pearl 1988).

Branch-and-Bound (BnB) is one such algorithm for solving combinatorial optimization queries in graphical models. It is guided by a heuristic evaluation function that provides an optimistic bound on the *cost to go* for every node. One such bounding scheme used is *mini-bucket elimination* (MBE) (Dechter and Rish 2003), which generates bounds for all of the nodes in the search space at once, given a static variable ordering for the problem. It duplicate variables in order to bound the treewidth of the problem and applys the exact *bucket elimination* (BE) algorithm (Dechter 1999) on the resulting relaxed problem. This is typically referred to as the *static mini-bucket heuristic*. Still, the heuristics generated by MBE can be poor when the treewidth of the problem is very high since we would need to relax the problem by duplicating many variables.

An alternative is to compute heuristics *dynamically*. In this setup, we recompute the heuristic we use on the subproblem induced by the current partial conditioning during search. However, this means that we must keep the computational cost of generating heuristics at each node, low. One such approach is maintaining *soft arc consistency* (SAC) (Larrosa and Schiex 2004) during branch-and-bound. Maintaining SAC is a method of *re-parameterizing* the problem by shifting costs from higher arity functions toward lower arity functions, which bounds the problem with a single nullary function that has cost shifted into it. One of these algorithms is optimal soft arc consistency (OSAC), which formulates the process of finding the set of re-parameterizations that maximize the cost shifted into the nullary function as a linear program (Cooper, de Givry, and Schiex 2007). However, maintaining OSAC entails solving a large LP for each search node, which is cost-prohibitive in terms of time. Another algorithm is virtual arc-consistency, which finds a sequence of cost shifts to tighten the bound based on a connection with classical arc consistency, yielding an iterative cost-shifting method (Cooper et al. 2008).

In other literature, there are several iterative approximation techniques based on solving linear programming (LP) relaxation of a graphical model (Wainwright, Jaakkola, and Willsky 2005). This initial work established connections between these LP relaxations and message-passing, which led to coordinate-descent update algorithms such as max-product linear programming (MPLP) (Globerson and Jaakkola 2007).

The ideas from this literature were used recently to extend the approach of static MBE (Ihler et al. 2012). It employs a similar algorithm to MPLP, known as *factor graph linear programming/join graph linear programming* (FGLP/JGLP) as a preprocessing step on the original problem. Next, it uses *MBE with moment-matching* (MBE-MM), an enhanced version of MBE that includes cost-shifting to enforce consistency between the duplicated variables, as the heuristic. However, as a static heuristic this still have similar drawbacks of generating weak heuristics for problems having large indcued-width and large domains.

In this work, we aim to 1. explore the use of FGLP as a heuristic generator *dynamically* for every node during the search, and compare with the most advanced statically generated heuristics describe above as in (Ihler et al. 2012), and 2. to combine both static and dynamic schemes into a single, potentially more powerful heuristic for BnB search. While generating dynamic heuristics based on FGLP is closely related to the soft-arc consistency algorithms such as those in the *toulbar2* solver (http://mulcyber.toulouse.

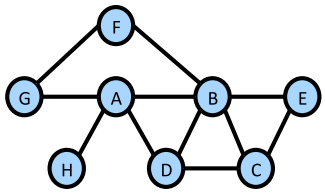

Figure 1: Primal graph of a graphical model over 8 variables.

inra.fr/projects/toulbar2/, our work provides an alternative based on techniques that come from LP literature. In particular, FGLP solves a problem that is identical to that of the LP for Optimal Soft Arc Consistency. Since FGLP is an LP coordinate-descent algorithm, it allows us to aim towards an optimal re-parameterization, yet terminating short of solving it to optimality, based on a given computation budget. The comparison of our dynamic re-parameterization schemes against recent static mini-bucket based schemes is performed here for the first time. Also, the combination of the two approaches (static, mini-bucket-based and dynamic, re-parameterization-based) is carried out here for the first time.

We present preliminary empirical results showing that on some problem instances for which static mini-bucket evaluation is quite weak when we have very limited memory, the dynamic FGLP scheme can prune the search space far more effectively, and in some cases this is carried out in a cost-effective manner despite the significant overhead inherent to the dynamically generated heuristics. We acknowledge however that the overhead of the dynamic re-parameterization is often quite prohibited limiting its effectiveness both when applied in a pure form and within a hybrid scheme.

The remainder of this paper is structured as follows: Section 2 introduces the underlying concepts. Section 3 presents a faster FGLP algorithm. Section 4 identifies the differences between the heuristics generated by the static MBE heuristic and dynamic cost-shifting heuristics and shows how to combine them into a hybrid scheme. Section 5 presents empirical results and analysis before Section 6 concludes.

Background

We describe combinatorial optimization problems as graphical models, which include Bayesian networks, Markov networks, and constraint networks. We provide some definitions below. (Pearl 1988; Dechter 2003; 2013).

Definition 1. ***(graphical model)*** *A* graphical model *is a 4-tuple* $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F}, \otimes)$*, where* $\mathbf{X} = \{X_i : i \in V\}$ *is a set of variables indexed by a set* V *and* $\mathbf{D} = \{D_i : i \in D\}$ *is the set of finite domains of values for each* X_i*.* $\mathbf{F} = \{f_\alpha : \alpha \in F\}$ *is a set of discrete functions, where* $\alpha \subseteq V$ *and* $X_\alpha \subseteq X$ *is the scope of* f_α*. We associate a* primal graph G *by representing each variable* X_i *as a node and include an edge* (X_i, X_j) *if* $i, j \in X_\alpha$ *for any* f_α*.* $\otimes = \{\prod, \sum, \bowtie\}$ *is a combination operator that defines the function represented by the graphical model* $\mathcal{M}$ *as* $\otimes_{\alpha \in F} f_\alpha(X_\alpha)$*.*

We focus on the *max-sum problem*, $C^* = \max_{\mathbf{X}} \sum_{\alpha \in F} f_\alpha(X_\alpha)$, in this work and assume upper bounds when discussing bounds on this quantity.

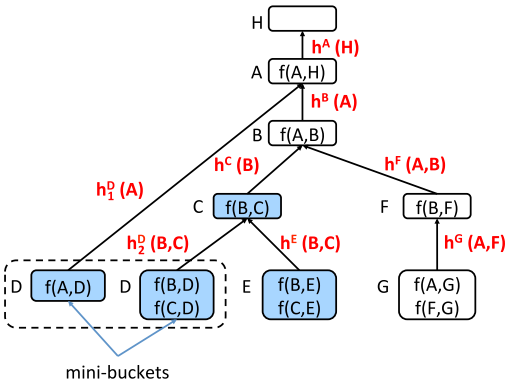

Figure 2: Mini-bucket tree for the model in Figure 1, with a z-bound of 3.

Search in Graphical Models

The max-sum problem and other inference queries can be solved via traversing the search space of a graphical model. Internal nodes represent partial assignments and leaf nodes represent complete assignments. For a node n, the cost of an assignment is defined by the combination of the costs of all functions that have been fully instantiated, which we denote as $g(n)$.

The size of the *OR search tree* is $O(k^n)$, representing all possible assignments to the graphical model. In this work, we search a smaller graph-based represenation of the search space, formed by capturing problem structure. Identical subproblems are identified by their *context*, which for a particular variable at depth d, is the partial instantiation that defines the subproblem, and can contain fewer variables than d. The size of the *context-minimal OR search graph*, formed by merging identical subproblems based on their context is $O(nk^{pw})$. n is the number of variables, k is maximum domain size, and pw is the pathwidth of the primal graph G along the search ordering (Dechter and Mateescu 2007).

Branch and Bound (BnB) is a search algorithm which performs a depth-first traversal of the search space and maintains a lower bound on the cost of the optimal solution, which corresponds to the best solution found so far. At each node, a heuristic $h(n)$ is computed to give an upper bound of the cost to go. $c(n) = g(n)+h(n)$ then gives an upper bound of the best solution extending the current partial assignment. If the upper bound is smaller than the lower bound, then we can prune the search space below that node. The effectiveness of BnB is highly dependent on the quality of heuristics. We discuss two possible approaches that are used in the following section. One is the mini-bucket heuristic and the other is based on re-parameterization.

Mini-Bucket Elimination

Bucket elimination (BE) (Dechter 1999) is a standard algorithm for solving reasoning tasks over graphical models. The *buckets* correspond to variables to be eliminated and are formed by organizing the functions of the graphical model

into these buckets according to an *elimination ordering* o. For each function f_α, it is placed in bucket i if $i \in \alpha$ and $i \prec j$ in o $\forall j \in \alpha, j \neq i$. Buckets are then processed by applying the $\otimes$ operator ($\sum$ for the max-sum task) to all the functions within a bucket, then eliminating the variable based on the reasoning task at hand (maximization for the max-sum task). For a processed bucket i, we denote the resulting function from the combination and elimination steps as a message h^i, which is passed to its parent bucket, defined by the same bucket placement procedure as the original functions. The result after all messages have been passed forms a tree structure, known as a *bucket tree*, with each h^i passed along an edge from a bucket to its parent. The largest scope size amongst all h^i is known as the *induced width* or *treewidth* w of the problem along ordering o. The algorithm's time and space complexity are exponential in w. (Dechter 1999)

Mini-bucket elimination (MBE) (Dechter and Rish 2003) approximates BE with a systematic way to avoid the computational complexity of BE. We consider an integer parameter z. For a bucket $\mathbf{B}_i$, we create a partition $Q_i = \{q_i^1, \ldots, q_i^p\}$, where q_i^j is referred to as a *mini-bucket* and corresponds to a subset of the functions of $\mathbf{B}_i$ such that the combined scope of these functions have no more than $z+1$ variables. We refer to this as satisfying the z-bound. Each mini-bucket is then processed separately as in BE and we denote the messages generated as h_j^i. As such, each mini-bucket within $\mathbf{B}_i$ can be viewed as duplicate copies of variable i and the bucket tree of MBE corresponds to the bucket tree of BE for a relaxed version of the problem with such duplicate variables. The result is a bound (upper-bound in the case of maximization) on the solution to the original problem. Using a single parameter z, we have a simple way to trade off computational complexity and the tightness of the bound. Increasing z generally tightens this bound, and MBE is identical to BE when $z \geq w$.

Bounds by Re-parameterization

The second orthogonal approach is based on re-parameterization coming from the literature on solving Linear Programming (LP) relaxations of the graphical model to generate bounds. The literature on enforcing soft arc consistency also offers various methods, instead based on varying strengths of soft arc consistency.

In the LP relaxation view, the idea is to consider bounding the max-sum objective by the maxima of each individual function:

$$C^* = \max_X \sum_{\alpha \in F} f_\alpha(X_\alpha) \leq \sum_{\alpha \in F} \max_X f_\alpha(X_\alpha), \qquad (1)$$

which exchanges the sum and max operators. One view of this is that each function has its own copy of each variable, and each are optimized separately. If each variable copy does happen to agree on its maximizing assignment, then this relaxation is tight and the bound produced is equal to the objective. However, this is generally not the case, so we need to compensate for the errors that accumulate.

The idea is to introduce a collection of functions over individual variables $\Lambda = \{\lambda_\alpha(X_i)\}$ and require

$$\forall i, \quad \sum_{\alpha \in F_i} \lambda_\alpha(X_i) = 0$$

This yields

$$\begin{aligned} C^* &= \max_X \sum_{\alpha \in F} f_\alpha(X_\alpha) \\ &= \max_X \sum_{\alpha \in F} f_\alpha(X_\alpha) + \sum_i \sum_{\alpha \in F_i} \lambda_\alpha(X_i) \\ &= \max_X \sum_{\alpha \in F} f_\alpha(X_\alpha) + \sum_{\alpha \in F} \sum_{i \in \alpha} \lambda_\alpha(X_i) \\ &= \max_X \sum_{\alpha \in F} (f_\alpha(X_\alpha) + \sum_{i \in \alpha} \lambda_\alpha(X_i)) \\ &\leq \sum_{\alpha \in F} \max_X (f_\alpha(X_\alpha) + \sum_{i \in \alpha} \lambda_\alpha(X_i)) \qquad (2) \end{aligned}$$

Our objective is to find λ such that we minimize (2), which serve to produce an optimal *re-parameterization* of the original model.

Depending on the literature, these λ functions are viewed as equivalence-preserving transformations (EPTs), in the soft arc consistency literature (Cooper and Schiex 2004), or as Lagrange multipliers enforcing consistency between the variable copies in the LP relaxation (Yedidia, Freeman, and Weiss 2005; Wainwright, Jaakkola, and Willsky 2005). In the latter view, these functions are computed by coordinate descent updates that can be viewed as message passing (Globerson and Jaakkola 2007).

Improving Factor Graph Linear Programming

Factor graph LP (FGLP) algorithm (Ihler et al. 2012) is a coordinate-descent algorithm for solving the LP in the previous section. We show the pseudocode in Algorithm 1.

Algorithm 1: FGLP($\mathbf{X}, \mathbf{D}, \mathbf{F}$)

Input: Graphical Model $\langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$, where f_α is a function defined on variables X_α

Output: Re-parameterized factors $\mathbf{F}'$, bound on optimum value

Iterate until convergence:
for *each variable* X_i **do**
  Let $F_i = \{\alpha : i \in \alpha\}$ with X_i in their scope
  $\forall \alpha \in F_i$, compute max-marginals: $\lambda_\alpha(X_i) = \max_{X_\alpha \setminus X_i} f_\alpha(X_\alpha)$
  $\forall \alpha \in F_i$, update parameterization: $f_\alpha(X_\alpha) \leftarrow f_\alpha(X_\alpha) - \lambda_\alpha(X_i) + \frac{1}{|F_i|} \sum_{\beta \in F_i} \lambda_\beta(X_i)$
return Re-parameterized factors $\mathbf{F}'$ and bound $\sum_{\alpha \in F} \max_X f_\alpha(X_\alpha)$

The main loop of the algorithm updates functions along a particular variable such that their max-marginals become

identical, which is known to achieve the minimum given that we fix the updates along all of the other variables. (Sontag et al. 2008). A typical fixpoint from iteratively applying the updates on each variable is that all of the function maximums become identical.

In our version, we make improvements to the algorithm to make efficient execution during search. This includes a change that normalizes the functions and a improved schedule for updates.

Normalization

In the setup for dynamic heuristics, we apply conditioning on a single variable and compute our heuristic based on the conditioned functions. For FGLP, we want to leverage the state of the problem achieved by running FGLP before conditioning. After conditioning, the max-marginals of the conditioned functions may not be the same and updates can be performed to tighten the bound once again. A typical scenario is that the conditioning removes the maximum value of one or more functions. Since their max-marginals now have lower values, running FGLP will shift costs from the rest of the problem into the conditioned functions to reach the fixpoint, so all functions need to be updated.

However, it is not necessary to update all of the functions if we normalize the re-parameterized functions such that their maximum is zero. By doing this, we only need to apply updates to variables in functions with non-zero maximums to achieve the same bound. We perform an additional cost shift given by the maximum of the function to a nullary function $f_\emptyset$, which is added to the set of functions in the graphical model. This is the same as the *unary project* operator in the soft arc consistency literature. We get the same bound as before, but can avoid many updates, as the maximum value for most functions after conditioning are already zero. By collecting costs into a single nullary function, subsequent cost-shifts during search do not move these costs around other functions.

We present this change as a subroutine that performs an update on a single variable in Algorithm 2. Figure 3 presents the update as message passing on a part of a factor graph. Line 4 describes the cost shift for normalization and line 5 is changed from the original FGLP algorithm to take into account the cost shifted into $f_\emptyset$.

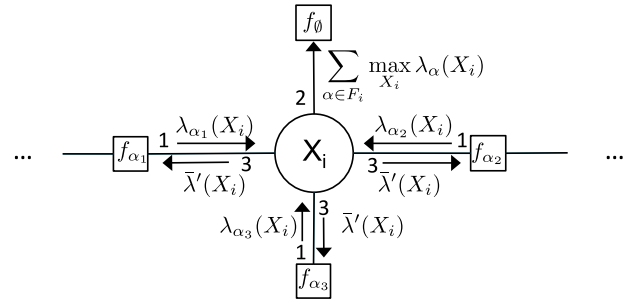

Figure 3: View of FGLP update as message passing. The numbers 1, 2, 3 indicate the order that the messages are computed.

Algorithm 2: FGLP-Variable-Update(X_i)

Input: Graphical Model $\langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$, where f_α is a function defined on variables X_α, variable X_i to update, current upper bound $f_\emptyset$

Output: Re-parameterized factors $f_\alpha(X_\alpha) \quad \forall \alpha \in F_i$

1 Let $F_i = \{\alpha : i \in \alpha\}$ with X_i in their scope

2 $\forall \alpha \in F_i$, compute max-marginals:

3 $\lambda_\alpha(X_i) = \max_{X_\alpha \setminus X_i} f_\alpha(X_\alpha)$

4 Shift costs into $f_\emptyset$:
$f_\emptyset \leftarrow f_\emptyset + \sum_{\alpha \in F_i} \max_{X_i} \lambda_\alpha(X_i)$

5 $\forall \alpha \in F_i$, update parameterization:
$\lambda'_\alpha(X_i) = \lambda_\alpha(X_i) - \max_{X_i} \lambda_\alpha(X_i)$
$\bar{\lambda}'(X_i) = \frac{1}{|F_i|} \sum_{\alpha \in F_i} \lambda'_\alpha(X_i)$
$f_\alpha(X_\alpha) \leftarrow f_\alpha(X_\alpha) - \lambda_\alpha(X_i) + \bar{\lambda}'(X_i)$

6 **return** Re-parameterized $\mathbf{F}'$ containing updates to f_α

Theorem 1. (complexity of FGLP-Variable-Update) *Computing each max-marginal takes $O(k^a)$ time, where k is the maximum domain size and a is the maximum arity size over the functions in the problem. Since we need to do this $|F_i|$ times, the total time complexity is $O(|F_i|k^a)$.*

Scheduling Updates

Unlike Algorithm 1 where the scheduling is left undefined, we address here scheduling to achieve faster performance. In particular, we seek to identify the variables that are the most important to update next.

Residual BP (Elidan, McGraw, and Koller 2006) maintains a priority queue of messages by computing them as if it were the next message to update and comparing them to the previously applied update via a distance measure known as a *message norm*. Message norms are computed by treating the messages as vectors and taking vector norms (L_1, L_2, L_∞, etc.) between them. This yields a weight indicating the importance of an update.

However, since this requires the calculation of messages we may possibly not use, we instead calculate priorities based on the changes made to functions that would affect a particular variable. Large changes to a function suggest that variables in that function should be updated. We maintain a priority p_i for each variable X_i. For a particular function and variable, the magnitude of an update is quantified by taking a message norm between the two normalized update messages $\|\lambda'_\alpha(X_i) - \bar{\lambda}'(X_i)\|$. Each variable in f_α except X_i then gets a priority value based on this quantity, or stays with its current priority value, whichever is greater. To stop the computation early we have two parameters: a tolerance ϵ that stops updates with the highest priority p_i is less than that value, and a maximum number of iterations m. Our full version of the FGLP algorithm with normalization and scheduling (FGLP-S) is given in Algorithm 3. The algorithm runs in a main loop that runs while the maximum priority is greater than the ϵ value or if the number of iterations m has not been exceeded (line 1). In each iteration, the variable with the maximum priority is extracted and its priority value is set to 0 (lines 2-3). The following lines are those of FGLP-

Variable-Update (Algorithm 2) (lines 4-8). Finally, we update all of the neighboring variable priorities with a priority based on the magnitude of the update just performed on variable i. The initial bound and priorities parameters are primarily used when we are computing dynamic heuristics, which we describe in a later section. Otherwise, the initial bound is 0, as no cost has been shifted from the problem at the start. The initial priorities are all ∞, to ensure that all variable updates are performed at least once.

Algorithm 3: FGLP-S($\mathbf{X}, \mathbf{D}, \mathbf{F}, f_\emptyset, p_1, ..., p_n, \epsilon, m$)

Input: Graphical Model $\langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$, where f_α is a function defined on variables X_α, initial bound $f_\emptyset$, initial priorities $p_1, ..., p_n$ for each X_i, tolerance value ϵ, maximum number of iterations m

Output: Re-parameterized factors $\mathbf{F}'$, bound on optimal value $f'_\emptyset$, updated priorities p'_i

1 **while** $\max_i p_i > \epsilon$ *and # iterations* $<= m$ **do**
2 $\quad i \leftarrow \arg\max_i p_i$
3 $\quad p_i \leftarrow 0$
4 $\quad$ Let $F_i = \{\alpha : i \in \alpha\}$ with X_i in their scope
5 $\quad \forall \alpha \in F_i$, compute max-marginals:
6 $\quad \lambda_\alpha(X_i) = \max_{X_\alpha \setminus X_i} f_\alpha(X_\alpha)$
7 $\quad$ Shift costs into $f_\emptyset$:
$\quad f_\emptyset \leftarrow f_\emptyset + \sum_{\alpha \in F_i} \max_{X_i} \lambda_\alpha(X_i)$
8 $\quad \forall \alpha \in F_i$, update parameterization:
$\quad \lambda'_\alpha(X_i) = \lambda_\alpha(X_i) - \max_{X_i} \lambda_\alpha(X_i)$
$\quad \bar{\lambda}'(X_i) = \frac{1}{|F_i|} \sum_{\alpha \in F_i} \lambda'_\alpha(X_i)$
$\quad f_\alpha(X_\alpha) \leftarrow f_\alpha(X_\alpha) - \lambda_\alpha(X_i) + \bar{\lambda}'(X_i)$
9 $\quad \forall \alpha, \forall j \in \alpha | j \neq i$, update priorities:
$\quad p_j \leftarrow \max(p_j, \|\lambda'_\alpha(X_i) - \bar{\lambda}'(X_i)\|)$
10 Let $\mathbf{F}'$ and $p'_1, ..., p'_n$ be the set of updated functions and priorities, respectively
11 **return** Re-parameterized factors $\mathbf{F}'$, updated bound $f'_\emptyset$, and updated priorities $p'_1, ..., p'_n$

Heuristics for Branch-and-Bound

We now turn to heuristic generation for Branch-and-Bound. We present static and dynamic methods of evaluating heuristics. In addition, we propose a hybrid that combines both methods.

Static Heuristics

The existing static mini-bucket schemes include MBE and MBEMM, with augmentations based on re-parameterizations via FGLP and JGLP (Ihler et al. 2012). The augmentations are based on executing FGLP/JGLP only once on the original problem to re-parameterize it, followed by an MBE variant (typically MBEMM) is run on the re-parameterized set of functions.

In any of these schemes, MBE is computed once and the h^i_j messages are then used to compute the heuristics $h_{MBE}(n)$, corresponding to a bound on the cost to go from any node in the search space. The value $c_{MBE}(n) = g(n) + h_{MBE}(n)$ is then compared against the best solution so far to decide whether to prune.

Dynamic Re-parameterization Heuristics

In a cost shifting approach, we instead focus re-parameterizing the problem during search to generate bounds dynamically. As is common in any dynamic heuristic scheme, the state of the functions are maintained at each node along a search path, allowing for backtracking and leveraging the re-parameterization performed at ancestor nodes. We also maintain the state of the priorities at each node. To quickly incorporate information from conditioning a variable with a value, we additionally set the priorities of all variables in functions that contain the conditioning variable to ∞.

We present the pure dynamic FGLP algorithm (dFGLP) in Algorithm 4. We start by selecting the functions that form the subproblem at the parent node of the search (line 1). Next, we condition on the parent variable X_{i-1} to form the subproblem at the current node (line 2). Since the functions may have been changed (especially if conditioning removes the maximum of the function), we set the priority of all of the remaining variables of these functions to ∞. The following line then calls the FGLP-S algorithm (Algorithm 3) (line 4). In order to generate a bound for each assignment to X_i, we select the remaining functions in the problem that contain variable X_i and maximize over the other variables. We then sum up these functions to generate a single unary function. This function represents a bound on the summation of the functions that contain X_i, for each assignment x_i. Each of the values of this function are then combined with the bound generated by FGLP-S to create a bound for each assignment (line 5).

Functions that are fully conditioned are implicitly collected into the $f_\emptyset$ function, so $f_\emptyset$ (and the $c(X_i)$ outputted by dFGLP) correspond to a bound on the best solution we can get from extending the current assignment $c_{FGLP}(n)$, rather than only bounding a best cost to go.

Hybrid Heuristic. We can also combine the power of the static heuristic directly with the cost-shifting methods by taking the minimum of the two at each node. The hybrid heuristic is then given by $c_{hybrid}(n) = \min(c_{MBE}(n), c_{FGLP}(n))$, taking the minimum of the two for the tighter upper bound.

Experiments

We experimented with running branch-and-bound with several heuristics.

For static heuristic schemes, we have:

- MBEMM with only FGLP preprocessing (FGLP+MBEMM)
- MBEMM with both FGLP and JGLP preprocessing (FGLP+JGLP+MBEMM)

For dynamic/hybrid heuristic schemes, we have:

- Dynamic FGLP alone with FGLP preprocessing (FGLP+dFGLP(m))

Algorithm 4: dFGLP($\mathbf{X}, \mathbf{D}, \mathbf{F}, f_\emptyset, p_1, ..., p_n, \epsilon, m, X_i$)

Input: Graphical Model $\langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$ formed after conditioning on previous search variables $X_1, ...X_{i-2}$, where f_α is a function defined on variables X_α, current bound $f_\emptyset$, current priorities $p_1, ..., p_n$, tolerance value ϵ, maximum number of iterations m, parent variable assignment $X_{i-1} = x_{i-1}$, current search variable X_i

Output: Re-parameterized factors $\mathbf{F}'$, best solution cost bounds given the assignment to ancestors of X_i for each assignment x_i: $c(X_i)$, updated bound $f'_\emptyset$, updated priorities $p'_1, ..., p'_n$

Let $F_{i-1} = \{\alpha : X_{i-1} \in \alpha\}$, set of scopes with X_{i-1} in their scope

$\forall \alpha \in F_{i-1}$, condition: $f_\alpha \leftarrow f_\alpha|_{X_{i-1}=x_{i-1}}$

Set maximum priority for variables of affected functions, $\forall \alpha \in F_{i-1}, \forall i \in \alpha$: $p_i \leftarrow \infty$

Re-parameterize factors, get bound and updated priorities: $\mathbf{F}', f'_\emptyset, p'_1, ..., p'_n \leftarrow$ FGLP-S($\mathbf{X}, \mathbf{D}, \mathbf{F}, \epsilon, m, f_\emptyset, p_i, ..., p_n, \epsilon, m$)

Compute the bounds for each assignment, $c(X_i)$: $c(X_i) \leftarrow f'_\emptyset + \sum_{\alpha \in F: i \in \alpha} \max_{X_\alpha \setminus X_i} f_\alpha(X_\alpha)$

return $c(X_i)$, re-parameterized factors $\mathbf{F}'$, updated bound $f'_\emptyset$, updated priorities $p'_1, ..., p'_n$

- Hybrid Dynamic FGLP with static FGLP+MBEMM (F+MBEMM+dFGLP(m))
- Hybrid Dynamic FGLP with static FGLP+JGLP+MBEMM (F+J+MBEMM+dFGLP(m)).

where m indicates the maximum number of variable updates performed per search node. We varied this parameter with 50, 100, and 150 updates. We set the schedule tolerance value $\epsilon = 10^{-7}$ and used the L_1-norm as the message norm.

In all of the setups, the same variable ordering was used for search and compiling the static heuristics. Each algorithm also searches the context-minimal OR graph of the problem. We present the list of benchmarks we experimented with along with summary statistics on their properties in Table 1.

The block world benchmark is taken from block world conformant planning instances expressed as unrolled dynamic Bayesian networks. The CPD benchmark comes from work on modeling computational protein design as a cost function network problem (Allouche et al. 2014). The rest of the benchmarks are taken from the PASCAL2 Probabilistic Inference Challenge problems.

We ran the experiments on an Intel Xeon X5650 2.66GHz processor and limited the memory usage to 1GB [1], which limits the z-bound that we can use. The time limit for each problem was 1 hour. For each instance, we also ran stochastic local search for 60 seconds to generate an initial solution. For other preprocessing schemes, we ran FGLP for 30 seconds and JGLP for 30 seconds (for the applicable setups). From this table, we exclude (but not limited to) instances where $z = w$, which makes the static MBE heuristic compute the exact solution, and instances where none of the algorithms managed to solve the instance within the time limit.

Results

We report the preprocessing time, total execution time, and the number of nodes processed for each algorithm setting in Table 2 on a selected set of instances from each benchmark. A '-' indicates that the algorithm was unable to solve the problem within the time limit. On each benchmark, we focus on pointing out which method performs better.

We can interpret the results by looking across the columns for each problem instance. In terms of heuristic strength, we see that the number of nodes processed decreases on many instances when using any of the dynamic schemes.

Block World Planning These instances, based on unrolled dynamic Bayesian networks, have relatively large induced widths ranging up to 60. Due to their moderate domain size of 3, we are still able to use relatively larger z-bounds for MBEMM. However, there are also high amounts of determinism in these problems that FGLP is able to propagate throughout the network. As such, although the static heuristic alone does not solve the harder instances in this benchmark, augmenting it with dynamic FGLP helps greatly.

Graph Coloring For the instances we managed to solve with our schemes, the z-bound used was relatively close to the induced width, allowing MBEMM to produce good bounds. Therefore, although there were savings in terms of the number of nodes when applying our dynamic schemes, the savings were often not substantial enough to trade well with the increased node processing time.

Computation Protein Design Notably, the primal graphs for these instances are complete graphs. As such their induced width is always $n - 1$. In addition, the domain sizes are extremely large, ranging up to 198. As such, this places a great restriction on the z-bound that MBEMM can use, which heavily hinders its performance. On the other hand, the number of variables is relatively low (mostly < 100) compared to the other benchmarks and all of the functions are binary, which makes FGLP updates fast. In the hybrid setups, the dynamic FGLP component was often the heuristic that was actually used.

PDB (Protein DB) We managed to solve only 2 of the instances in this benchmark. Similar to the previous benchmark, the domain sizes are large compared to the other benchmarks we experimented on, which has the same expected effect of limiting the z-bound for MBEMM. How-

[1] We chose a 1GB limit, based on our experimentation with the algorithms and a range of test problems, in order to focus on problems that were neither trivial nor impossible, to get a meaningful comparison of the relative performance of the algorithms

Benchmark	# inst	# solved	n	w	pw	k	$\|F\|$	max arity	z
Block world	15	13	192-2695	17-60	101-1699	3	196-2703	5	12-17
Coloring	12	8	25-450	6-287	15-387	2-5	82-5715	2	6-15
CPD	47	33	11-120	10-119	10-119	48-198	67-7261	2	2-5
PDB	10	2	337-1972	22-51	151-988	81	1360-8817	2	3
Pedigree	22	19	298-1015	15-39	86-357	3-7	335-1290	4-5	10-21
Planning	13	7	45-431	9-90	18-226	4-27	272-9689	2	5-12
Radio frequency	11	3	14-458	7-119	10-295	44	58-1878	2	3-4

Table 1: Benchmark statistics. # inst - number of instances, # solved - number of instances solved by methods in this work, n - number of variables, w - induced width, pw - pathwidth, k - maximum domain size, $|F|$ - number of functions, a - maximum arity, z - z-bound used for MBE

ever, unlike the CPD instances, there is also a high number of variables and functions, which makes them difficult to solve even with dynamic FGLP, each update step can take a long time. The two solved instances are the two with the lowest number of variables and functions. For one of them, the preprocessing from FGLP/JGLP is sufficient to make the problem easy. For the other, we see that the dynamic FGLP scheme produces a much better heuristic.

Pedigrees The relationship between the z-bound used and the induced width is similar to those of the coloring instances, where z is relatively close to the induced width. We can note that for many of the instances, using the dynamic FGLP scheme alone does not return a solution. In the few cases that it does, considering the number of nodes processed, it is also producing weaker bounds.

Planning For many of these instances, the z-bound used is once again limited relative to the induced width, due to the larger domain sizes. As expected, the dynamic FGLP scheme is producing much of the heuristics that are actually being used at each node.

Radio Frequency Assignment This benchmark has similar properties to the protein problems (high domain sizes and induced width). For the two instances shown, though the dynamic FGLP scheme alone has the best runtime, it is not significantly better than those of the static heuristic.

Summary

We omit from Table 2 many results from each benchmark due to space constraints. To provide a general summary, most problem classes either perform well with the static heuristic alone or the dynamic heuristic alone. In particular, we noticed that 5 of 8 of the solved coloring instances and 11 of 19 of the solved pedigrees had the best time performance using the static heuristic only. The rest of the solved instances for these benchmarks worked better in the hybrid setup, but the improvement was marginal. On the other hand, 28 of the 33 solved CPD instances and 5 of the 7 solved planning instances had the best time performance using the dynamic heuristic only. Only the block world instances performed better with the hybrid setup, with 8 of the 13 instances having the best time performance using the hybrid heuristic.

We generally see that we are able to solve some problems more quickly compared with the static heuristics for two reasons. First, there are cases where the cost of compiling the static heuristic is large enough that using the dynamic heuristic alone is better. Second, the heuristics produced by the dynamic scheme are often stronger, enough to trade off the additional cost incurred by the computation required on each node. We can also observe that increasing the number of variable updates generally improves the heuristic, though there is also a time trade-off here in many cases.

In a number of cases, we can see some unintuitive results such as getting decreased performance in terms of the number of nodes when increasing the number of variable updates. This is due to variances in the bound quality as a result of the tolerance parameter, which may stop the update iterations early.

Conclusion

We explored FGLP, a method that keeps the objective of finding the set of optimal cost shifts to tighten in mind, placing our focus on improving the coordinate descent procedure used. Results show that it generates a powerful heuristic compared to static mini-bucket heuristics in many cases where our memory budget is limited. In addition, the hybrid heuristic yields the smallest number of nodes expanded in the search space for some instances.

Existing scheduling work for message passing algorithms lead us in the correct direction, but they are not designed for our scenario of search. The current schedule which is parameterized by the maximum number of variable updates and a tolerance has its shortcomings in that it may not balance the computation load correctly. For instance, one might consider performing more updates at a higher level of the search space to allow descendant nodes to start at a better parameterization. This suggests future work into more sophisticated scheduling schemes tuned for use at every node within search. Additionally, expanding the heuristic to work in AND/OR search spaces could also help by decomposing the subproblems into independent parts, making both the search space smaller and possibly allowing FGLP to operate on much smaller problems. Lastly, we did not investigate the scheme's strength when searching along dynamic variable orderings. Though this would result in not being able to use the hybrid scheme, it would be interesting to observe how our FGLP heuristic compares to the various soft-arc consistency methods in the literature.

problem $(n, w, pw, k, \|F\|, a, z)$	FGLP+MBEMM FGLP+JGLP+MBEMM			F+dFGLP(50) F+MBEMM+dFGLP(50) F+J+MBEMM+dFGLP(50)			F+dFGLP(100) F+MBEMM+dFGLP(100) F+J+MBEMM+dFGLP(100)			F+dFGLP(150) F+MBEMM+dFGLP(150) F+J+MBEMM+dFGLP(150)		
	ptime	time	nodes	ptime	time	nodes	ptime	time	nodes	ptime	time	nodes
Block World Planning												
bw6_3_4_6	95	168	11205738	66	447	229640	66	113	30499	66	**108**	23339
(892,34,527,3,899,5,14)	131	326	25416493	89	165	51040	89	120	17999	89	121	**16546**
				126	-	-	131	1047	25132	131	1222	22662
bw6_3_4_8	100	-	-	67	2034	901590	66	395	274699	67	3204	2102317
(1184,34,715,3,1191,5,14)	131	-	-	94	174	47250	94	**142**	27029	94	**142**	**24991**
				133	3301	398789	133	2893	2317756	134	1374	488446
bw7_4_4_7	85	-	-	68	-	-	67	655	228642	68	690	264774
(1894,58,1151,3,1902,5,12)	117	-	-	78	1363	486831	78	615	190273	79	**542**	**155002**
				118	-	-	115	-	-	118	-	-
Graph Coloring												
myciel5g_3.wcsp	103	**104**	405719	67	165	198249	67	158	106599	67	155	80655
(47,19,25,3,237,2,14)	136	137	232895	96	127	62737	96	132	42587	96	138	34771
				139	226	43575	139	289	35875	139	354	**32797**
queen5_5_3.wcsp	130	**131**	138452	68	510	1087310	68	793	911666	69	1087	859040
(25,18,21,3,161,2,15)	143	144	274648	121	153	72494	122	173	65414	121	193	**62960**
				143	288	126575		-	-	144	535	111323
Computational Protein Design												
1DKT.mat...	130	305	5121273	91	404	4625	92	285	1756	92	**235**	**896**
(46,45,45,190,1082,2,3)	152	284	3605639	123	559	6701	121	379	2246	122	304	1142
				161	392	3745	161	453	2530	161	383	1324
1PIN.mat...	707	2221	59060806	92	1767	14094	92	**1196**	3390	92	1465	2569
(28,27,27,194,407,2,3)	1002	3296	94827316	700	2571	17986	705	1590	3394	701	1709	**2161**
				728	2603	18395	726	1594	3054	727	1787	2485
2TRX.mat...	242	-	-	91	**359**	6406	91	590	4587	91	872	**4330**
(61,60,60,186,1892,2,5)	235	-	-	220	1098	22526	221	1750	14638	225	2221	12610
				803	-	-	783	-	-	771	-	-
Protein DB												
pdb1i24	121	121	**338**	90	**91**	395	90	92	396	90	92	395
(337,29,151,81,1360,2,3)	139	139	395	93	104	1696	90	98	929	90	100	984
				148	149	**338**	148	149	395	148	150	**338**
pdb1kwh	125	126	10652	91	96	664	90	**95**	643	90	96	558
(424,26,216,81,1881,2,3)	146	638	7743310	97	-	-	97	-	-	97	1957	61774
				152	156	554	152	156	514	152	157	**471**
Pedigree												
pedigree37	103	**104**	6047	74	-	-	65	-	-	65	-	-
(726,20,217,5,1033,4,14)	113	113	10375	99	106	5931	99	107	5931	99	107	5931
				112	754	1155216	112	620	889454	113	120	**871**
pedigree38	107	107	4011	68	-	-	68	-	-	68	-	-
(581,16,221,5,725,4,11)	113	153	1959805	99	**102**	2824	99	103	2718	99	103	**2655**
				114	-	-	114	-	-	114	-	-
pedigree41	83	-	-	64	-	-	64	-	-	64	-	-
(885,33,323,5,1063,5,17)	126	**699**	**123869387**	79	-	-	79	-	-	79	-	-
				126	-	-	126	-	-	126	-	-
Planning												
bwt4ac.wcsp	99	-	-	85	86	323	86	86	238	86	86	229
(179,40,73,18,2563,2,7)	126	-	-	75	76	414	74	**75**	275	75	76	369
				151	153	448	151	153	256	144	146	**200**
bwt5ac.wcsp	116	-	-	91	154	8402	91	129	4056	91	**123**	**3084**
(431,61,165,27,9246,2,5)	132	-	-	88	303	27659	86	139	5653	86	132	4474
				137	245	7978	137	247	4970	138	242	3504
zenotravel04ac.wcsp	154	-	-	91	95	900	91	94	474	90	**93**	392
(239,31,75,24,3147,2,5)	151	-	-	125	131	1680	125	129	697	126	130	539
				154	167	1125	154	164	**332**	153	167	474
Radio Frequency Assignment												
graph05.wcsp	114	114	7816	91	**105**	**4739**	90	112	4843	90	117	4843
(100,24,60,44,417,2,3)	145	146	55830	87	185	195304	86	199	195314	86	205	195314
				145	178	17584	145	203	14463	145	187	29387
graph06.wcsp	147	148	2521	91	129	2951	91	**102**	1333	92	104	885
(200,47,130,44,844,2,3)	178	178	**236**	120	-	-	120	-	-	120	-	-
				179	181	242	179	182	242	179	183	242

Table 2: Results on selected instances of each benchmark. n - number of variables, w - induced width, pw - pathwidth, k - maximum domain size, $|F|$ - number of functions, a - maximum arity, z - z-bound used for MBE. In **bold** are the minimum running time/nodes achieved for a problem over all of the heuristic setups. Time limit: 1 hour, memory limit 1GB.

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.

References

Allouche, D.; André, I.; Barbe, S.; Davies, J.; de Givry, S.; Katsirelos, G.; O'Sullivan, B.; Prestwich, S.; Schiex, T.; and Traoré, S. 2014. Computational protein design as an optimization problem. *Artificial Intelligence* 212:59–79.

Cooper, M., and Schiex, T. 2004. Arc consistency for soft constraints. *Artificial Intelligence* 154(1):199–227.

Cooper, M. C.; de Givry, S.; Sánchez, M.; Schiex, T.; and Zytnicki, M. 2008. Virtual arc consistency for weighted csp. In *AAAI*, volume 8, 253–258.

Cooper, M. C.; de Givry, S.; and Schiex, T. 2007. Optimal soft arc consistency. In *IJCAI*, volume 7, 68–73.

Dechter, R., and Mateescu, R. 2007. And/or search spaces for graphical models. *Artificial intelligence* 171(2):73–106.

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

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

Dechter, R. 2003. *Constraint processing*. Morgan Kaufmann.

Dechter, R. 2013. *Reasoning with probabilistic and deterministic graphical models: exact algorithms*, volume 23 of Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Elidan, G.; McGraw, I.; and Koller, D. 2006. Residual belief propagation. In *Uncertainty in Artificial Intelligence (UAI)*.

Globerson, A., and Jaakkola, T. S. 2007. Fixing max-product: Convergent message passing algorithms for map lp-relaxations. In *Advances in neural information processing systems (NIPS 2007)*, 553–560.

Ihler, A. T.; Flerova, N.; Dechter, R.; and Otten, L. 2012. Join-graph based cost-shifting schemes. In *Proceedings of the 28th Conference on Uncertainty of Artificial Intelligence (UAI 2012)*.

Larrosa, J., and Schiex, T. 2004. Solving weighted csp by maintaining arc consistency. *Artificial Intelligence* 159(1):1–26.

Pearl, J. 1988. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann.

Sontag, D.; Meltzer, T.; Globerson, A.; Jaakkola, T.; and Weiss, Y. 2008. Tightening lp relaxations for map using message passing. In *Proceedings of the 24th Conference on Uncertainty of Artificial Intelligence (UAI 2008)*.

Wainwright, M. J.; Jaakkola, T. S.; and Willsky, A. S. 2005. Map estimation via agreement on trees: message-passing and linear programming. *Information Theory, IEEE Transactions on* 51(11):3697–3717.

Yedidia, J. S.; Freeman, W. T.; and Weiss, Y. 2005. Constructing free-energy approximations and generalized belief propagation algorithms. *Information Theory, IEEE Transactions on* 51(7):2282–2312.