Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence

Pushing the Power of Stochastic Greedy Ordering Schemes for Inference in Graphical Models

Kalev Kask, Andrew Gelfand, Lars Otten, Rina Dechter
Donald Bren School of Information and Computer Sciences
University of California, Irvine
Irvine, CA 92697, USA

Abstract

We study iterative randomized greedy algorithms for generating (elimination) orderings with small induced width and state space size - two parameters known to bound the complexity of inference in graphical models. We propose and implement the Iterative Greedy Variable Ordering (IGVO) algorithm, a new variant within this algorithm class. An empirical evaluation using different ranking functions and conditions of randomness, demonstrates that IGVO finds significantly better orderings than standard greedy ordering implementations when evaluated within an anytime framework. Additional order of magnitude improvements are demonstrated on a multi-core system, thus further expanding the set of solvable graphical models. The experiments also confirm the superiority of the MinFill heuristic within the iterative scheme.

Introduction

Having good (i.e., small tree-width) variable orderings in graphical models is known to be of central significance. While it was thought initially that this will impact only exact algorithms, many approximation schemes such as generalized belief propagation, mini-bucket elimination and AND/OR sampling (Yedidia, Freeman, and Weiss 2005; Dechter and Rish 2002; Gogate and Dechter 2008) are highly dependent on the availability of good orderings.

Finding a minimal tree-width ordering is known to be NP-complete (Arnborg, Corneil, and Proskourowski 1987). Therefore, the past two decades have seen serious effort devoted to developing anytime complete algorithms (Gogate and Dechter 2004) and approximation schemes, e.g. (Robertson and Seymour 1983; Bodlaender 2009; 2007). However, the most popular and practical class of approximations are greedy variable-ordering schemes that utilize various variable-ordering heuristics (Dechter and Meiri 1989; Kjaerulff 1992). Greedy schemes are popular because they are relatively fast, leaving ample time for the exponential inference computation that follows. Indeed, most recent research in probabilistic inference and optimization in graphical models has focused on advancing exact and approximate inference methods. This work has resulted in significant computational improvements (Darwiche 2001; Bacchus, Dalmo, and Piassi 2003; Dechter and Mateescu 2007; Kask, Dechter, and Gelfand 2010). However, since inference is exponential in the tree-width, a small reduction in tree-width (say by even by 1 or 2) can amount to one or two orders of magnitude reduction in inference time. Thus, huge computational gains are also possible by simply finding improved variable orderings.

An approach that has proven successful in practice is iteratively executing a greedy ordering algorithm with random tie-breaking and keeping track of the best ordering generated (Fishelson and Geiger 2003). This scheme is appealing because it is anytime, easy to employ and easily parallelized. While empirical evaluations of greedy ordering algorithms have been conducted (Clautiaux et al. 2004; Kjaerulff 1992; Larranaga et al. 1997; Bodlaender 2009), iterative schemes have not been systematically studied.

In this paper we study iterative randomized greedy schemes for finding variable orderings that can minimize both tree-width and state-space measures. We focus on the following ideas:

1. Implementing greedy variable-ordering schemes in a highly efficient manner.
2. Adding stochasticity to the selection process to encourage discovery of a more diverse set of variable orderings (in the spirit of stochastic local search).
3. Aborting iterations of the greedy algorithm that yield unpromising orderings (in the spirit of branch and bound).
4. Exploiting parallel processing.

Specifically, the paper describes the development of a highly efficient, parallel variable ordering algorithm - the Iterative Greedy Variable Ordering (IGVO) algorithm - that can accommodate any greedy ranking function. The IGVO algorithm also employs randomized "pooling" during variable selection (an idea from (Fishelson and Geiger 2003)) and early termination rules to stop unpromising iterations.

In an extensive empirical evaluation, we demonstrate an impressive (5-10x) speedup over a standard greedy variable ordering implementation. We also show that IGVO finds far better orderings than a standard implementation. These gains are attributed to a more efficient implementation, the pooling scheme and, to a lesser extent, early termination. In addition, we show that parallelization of the iterative scheme yields linear speedup using multi-core CPUs.

Copyright © 2011, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Background

A **graphical model** $\mathcal{R}$ is a 4-tuple $\mathcal{R} = \langle X, D, F, \otimes \rangle$ that specifies variables, (finite) domains and functions. The arguments of a function is its *scope*. It represents the combination of all its functions: $\otimes_{i=1}^{r} f_i$. For example, for Bayesian or Markov networks, the functions are probabilities and the combination operator is multiplication.

The **primal graph** of a graphical model associates a node with each variable and connects any two nodes appearing in the same scope. The set of neighbors of v in a graph $G = (V, E)$ is denoted $N_G(v) = \{w \in V | \{v, w\} \in E\}$ and the neighbors including v itself is denoted $N_G[v] = N_G(v) \cup \{v\}$. We assume that the reader is familiar with graph concepts, such as cycles, chords and cliques.

DEFINITION **1.** ***Variable ordering, Simplicial variable*** *- An ordering of a graph $G = (V, E)$ is a bijection $d : V \rightarrow \{1, 2, ..., n\}$, where $|V| = n$. An ordered graph is a pair (G, d), where $d = V_{(1)}, ..., V_{(n)}$ is an ordering. A vertex $v \in V$ is called* **simplicial***, if the set of its higher ordered neighbors $\{w | \{v, w\} \in E \wedge d(w) > d(v)\}$ forms a clique. An elimination ordering d if perfect, if all $v \in V$ are simplicial.*

Note that we often denote the nodes V by the variables of the graphical model X. A variable ordering d induces a sequence of supergraphs of G, defined as follows. $G_0 = G$, and for all $1 \leq i \leq n$, G_i is the graph obtained from G_{i-1} by adding edges so that all vertices in $C_i = N_{G_{i-1}}(X_i) \cap \{X_{i+1}, ..., X_n\}$ are pairwise connected. New edges added during the elimination process are called fill edges. $F_i = E(G_i) \setminus E(G_{i-1})$ is the set of fill edges added during step i. Note that graph G_n is chordal and the set C_i is a clique. The process of obtaining graph G_i from G_{i-1} is referred to as *eliminating* vertex $X_{(i)}$.

DEFINITION **2.** ***Induced width, Treewidth*** *- Given an ordered graph (G, d), the induced width of $X_{(i)} = |C_i|$. The width of an ordering is $w(d, \mathcal{R}) = \max_{i=1}^{n} |C_i| - 1$. The induced width of a graph, $w^\star$, is the minimal induced width over all possible orderings. Treewidth is identical to induced-width and we will use those terms interchangeably.*

DEFINITION **3.** ***Total State Space*** *- The* total state space *of an elimination order d with respect to a graphical model $\mathcal{R}$ is: $s(d, \mathcal{R}) = \sum_{i=1}^{n} s(X_{(i)}, G_i)$ where $s(X_{(i)}, G_i) = \prod_{u \in N_{G_i}[X_{(i)}]} |D(u)|$ is the space needed to eliminate node $X_(i)$ from G_i and $D(u)$ is the domain of node u (Kjaerulff 1992).*

Greedy Variable Ordering (GVO) Algorithms

The general greedy variable ordering (GVO) scheme constructs an ordering as follows. A vertex v is greedily selected according to a heuristic ranking function. This vertex is placed into the first position of the ordering, its neighbors are connected and it is then eliminated. Then a second vertex is selected, placed into the next position of the ordering and eliminated. This process is repeated until all vertices have been eliminated. At each step, more than one vertex may be of minimum cost under the heuristic ranking function. In such cases, the vertex to be eliminated is selected randomly from the set of minimum cost vertices - i.e. random tie-breaking is used.

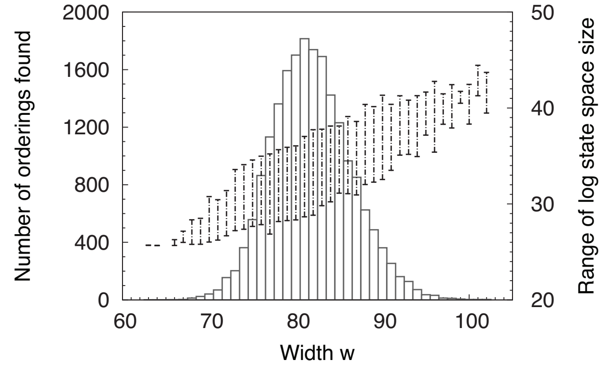

Figure 1: Histogram of induced width and state space size observed over 20,000 iterations of MinFill with random tie-breaking on problem *largeFam3-15-54* (3511 variables).

Vertex selection is guided by a heuristic ranking function. Using different ranking functions amounts to building elimination orders via different greedy algorithms. The following are three common ranking functions.

1. **MinFill** Cost - The number of fill edges added. Specifically, $MF(v) = |F_v|$, where F_v is the set of fill edges that would be added if variable v were to be eliminated.
2. **MinDegree** (also known as MinInduced-Width) Cost - The degree of variable v in the current fill graph. Specifically, $MD(v) = \deg(v)$.
3. **MinComplexity** Cost - The complexity of variable elimination. Specifically, $MC(v) = \prod_{u \in N_G[v]} d(u)$, where $d(u)$ is the domain size of u. This ranking function is designed to minimize the total state space size.

The Iterative GVO (IGVO) Algorithm

In this section, we describe our iterative GVO (IGVO) algorithm. We begin by motivating the need for iterative randomized algorithms. We then give an algorithmic description of the basic randomized GVO algorithm, which is a subroutine of IGVO. Last, we describe the primary differences between IGVO and existing GVO schemes. This includes a description and complexity analysis of two implementation details that yield significant speedup.

Demonstrating Variability

Consider the histogram shown in Figure 1, which contains the empirical distribution of widths over 200000 runs of GVO using the MinFill ranking function on a particular Markov network. It is typical of the variability in both width and state-space size observed while running our greedy scheme on instances from linkage analysis, and shows the induced widths of the orderings found (against the left vertical axis) and the (log of) the minimum and maximum state space size for each induced width found (right vertical axis). For most of the width values, there is a wide range of state

space sizes; moreover, there is a significant overlap between state space sizes of neighboring widths. As expected we also see strong correlation between width and state space size - as width gets smaller, the state space size declines. This variability encourages the use of randomness in an iterative search for low width and low state space variable orderings and also demonstrates the need for early termination. The IGVO algorithm is described next.

Algorithmic Description

The basic randomized GVO algorithm is presented in Figure 2. The algorithm utilizes any ranking function $VC(X)$, such as the min-fill, min-degree or min-complexity heuristics described above. The algorithm has a total of n iterations, where n is the number of variables in the graphical model. At each iteration a variable is selected for elimination. Any simplicial variables in the graph are always eliminated (item 2a) because their elimination does not introduce any new fill edges and increase the width of an ordering (Robertson and Seymour 1983). If there are no simplicial variables (item 2b), a pool of the p variables having the smallest variable cost (wrt. the given ranking function) are identified. A variable X is randomly selected from this pool with probability proportional to its cost. The variable selected is added to the ordering (step 4) and the graph is updated by removing the variable and adding the necessary fill edges (step 3).

The GVO algorithm is a subroutine in the main iterative algorithm presented in Figure 3. The iterative algorithm uses an objective function $C(d, R)$ to compare different variable orderings. The objective function can be either the induced width of the ordering (i.e. $C(d, R) \equiv w(d, R)$) or the total state space of the ordering (i.e. $C(d, R) \equiv s(d, R)$). After every iteration, the variable ordering found is compared to the best ordering found so far and the best ordering is retained. The cost of a partial ordering is also evaluated in every iteration of the GVO. If this cost exceeds the current upper bound, UB, GVO is terminated as completing the variable ordering will lead to an inferior one. We refer to discontinuing the GVO in this manner as *Early Termination*. Since an execution of GVO is independent of other executions (except for the early termination condition), the main iterative algorithm also executes multiple runs of GVO in parallel.

Key Enhancements

The iterative GVO algorithm differs from standard GVO algorithms in the following ways:

1. **Randomization by pooling** - Randomization is introduced into the basic variable selection step. This allows our algorithm to select variables that are non-optimal according to the heuristic ranking function. This is similar to the idea used in (Fishelson and Geiger 2003), but their work is based on elimination cost only.
2. **Early termination** - Our algorithm improves upon naive iterative greedy implementations by terminating inferior variable orderings.

Algorithm **Greedy Variable Order (GVO)**
Input: Graphical model R, its primal graph G=(V,E), a ranking function $VC(X)$, pool size p, exponent e, Objective Function $C(d, R)$ and Upper Bound UB
Output: A variable ordering $d = (X_{(1)}, ..., X_{(n)})$.
• **Initialize:** Set $d = \emptyset$, $W = V$.

• **For** k = 1, ..., n, do, using filled graph G_{k-1} :

1. **If:** $C(d, R) > UB$, *Terminate Early*
2. Select a variable X to eliminate:
   (a) **If:** any simplicial variables in W, pick one as X,
   (b) **Else:** Order the variables from W according to cost VC. Let P be a pool of the p lowest cost variables. Select X from P with probability: $VC(X)^e / \sum_{Y \in P} VC(Y)^e$.
3. Eliminate X from G_k: connect neighbors of X, remove X.
4. Set $W = W \setminus \{X\}$ and $d(k) = X$.

• **Return:** d

Figure 2: The Greedy Variable Ordering (GVO) Subroutine

Algorithm **Iterative Greedy Variable Ordering (IGVO)**
Input: Graphical model R, a cost function $VC(X)$, pool size p, exponent e, number of threads m, Objective Function $C(d, R)$ and Upper Bound UB, timeout/# iterations.
Output: A partial variable order $d_O = (X_{(1)}, ..., X_{(m)})$.

• **Initialize:** Let $d^\star$ be the best ordering known at any point, and $UB = C(d^\star, r)$ be its cost. Let G be the primal graph of R.

• **On** m **threads** execute :

1. Compute d=GVO(R, G,VC,p,e,UB).
2. If $C(d, R) < UB$, set $d^\star = d$ and set $UB = C(d, R)$.

• **Return:** $d^\star$

Figure 3: The Iterative GVO (IGVO) Algorithm

3. **Optimizing the efficiency of GVO** - A significant reduction in run-time is obtained due to the following algorithmic improvements.
   - Adding fill edges to the graph has complexity of $O(deg^3)$ because the adjacency of all pairs of neighbors of X must be checked. By keeping all adjacency lists sorted, we can reduce this to $O(2 \cdot deg^2)$, where deg is the degree of the variable being eliminated.
   - When using the MinFill ranking function, the number of fill in edges (MinFill removal cost) must be updated every time a variable is eliminated. Rather than recomputing the MinFill cost of all vertices, this is typically done by recomputing the MinFill cost of only the neighbors of X and the neighbors of $N_G(X)$. Updating costs in this fashion is wasteful, since there are only 3 cases where the MinFill cost actually changes. Assuming that X is being eliminated, the 3 updates are:
   (a) For every variable w and u, such that $(w, u) \in E$,

$(u, X) \in E$, $(w, X) \notin E$, subtract 1 from u.

(b) For every fill edge (u, v) added, for every w such that $(w, u) \in E$, $(w, v) \notin E$, add 1 to u.

(c) For every fill edge (u, v) added, for every w such that $(w, u) \in E$ and $(w, v) \in E$ were not added, subtract 1 from w.

The first case can be handled as fill edges are added at no overhead. The second and third add complexity $O(nf \cdot 2 \cdot deg)$ where nf is the number of fill edges added.

4. **Combined objectives** - The algorithm can consider both tree-width and state space size as objectives in searching for variable orderings that minimize both the time and space of inference computations.

5. **Parallelism** - The algorithm is clearly massively parallel and can exploit the multi-core architecture of modern CPUs.

The above enhancements were incorporated into our implementation of the randomized GVO algorithm with MinFill ranking function. The complexity gains resulting from point 4 is summarized in the following theorem:

Theorem 1. *The complexity of GVO is* $O(n(w^2 + n \cdot log(n)) + NF \cdot w)$*, where* n *is the number of variables,* w *is the width,* NF *is the number of fill edges. The complexity of standard MinFill is* $O(n(n + w^5))$.

Proof. The complexity of a single iteration of GVO is $O(deg^2 + nf \cdot deg + n \cdot log(n))$, where $n \cdot log(n)$ is the cost of constructing the pool. Over n iterations this is bounded by $O(n(w^2 + n \cdot log(n)) + NF \cdot w)$. Without pooling, the complexity would be $O(n(w^2 + n) + NF \cdot w)$. In a standard MinFill algorithm, eliminating a variable X is an $O(deg^3)$ operation, requiring enumeration of all pairs of neighbors adding edges where necessary. It requires updating MinFill ranking values for all neighbors and neighbors of neighbors of X, at a cost of $O(deg^5)$. Selecting a variable is $O(n)$, for a total single iteration cost of $O(n + deg^3 + deg^5)$. Over n iterations this is bounded by $O(n(n + w^5))$. □

Experiments

We conducted an extensive empirical evaluation of the suggested scheme and its parameters. The bulk of our experiments were performed on 242 *largeFam* problems modeling haplotype and linkage queries on biological pedigree data from the domain of computational genetics. The problems have between 2000 and 6000 variables with domain sizes from 2-6 and induced width ranging from the teens to over 100.

Comparing Greedy Ranking Functions

We first evaluated the impact of different ranking functions on the induced width and state space objectives.

Minimizing Induced Width. IGVO was run on each instance with each ranking function for 1 hour and the best width recorded. Figure 4 shows the results in a cumulative manner: For a given width (x-axis) the curves depict the number of instances (out of 242) having width lower or equal to the one specified in the x-axis, for each ranking function. Clearly MinFill outperforms the other ranking functions and returns lower-width orderings across the problem set.

For lack of space we don't include the full set of results, but we can state that on 186 problems MinFill is strictly better than both other schemes. On average, the best orderings found by MinFill have width 3 lower than MinDegree and 7 lower than MinComplexity (corresponding to up to 3 and 7 orders of magnitude better algorithmic performance). Furthermore, on average MinFill finds the best ordering after 836 seconds, 88 seconds before MinDegree and 717 seconds before MinComplexity.

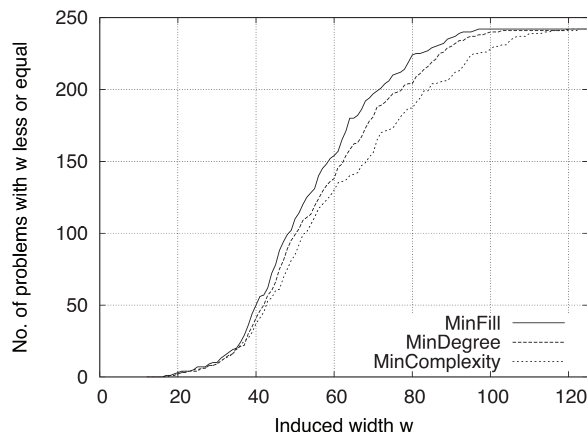

Figure 4: Cumulative plot of the best widths found by the different ranking functions for 242 *largeFam* instances (higher is better). Timeout 1 hour, $p = 8, e = -1$.

	MinFill	MinDeg	MinCompl
	domain sizes 2-6		
# wins	185	3	54
log of min SS	23.6	26.4	25.4
w of min SS	59	62	66
	domain sizes 2-30		
# wins	108	7	127
log of min SS	37.6	40.5	39.1
w of min SS	61	64	60

Table 1: Results when using state space size (SS) as objective. All values except number of wins are averaged across 242 *largeFam* problems (100,000 iterations per run).

Minimizing State Space Size. Results of using the three ranking functions with the state space objective are provided in Table 1. In the top half, problems have their original domain sizes (2-6), while in the bottom half we adjusted the domains to be either 2 (2/3 of the time) or 30 (1/3 of the time). We thus see that MinFill performs best by far on the original configuration, finding the ordering with the smallest state space size in the vast majority of cases (185 out of 242). However, once the domains are very uneven (bottom

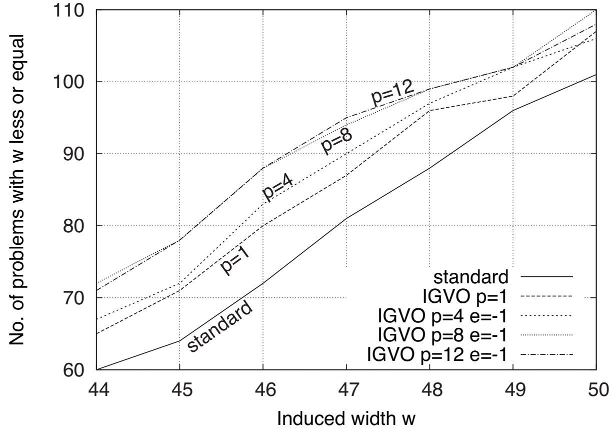

Figure 5: Magnified cumulative plot comparing various pool sizes ($e = -1$, timeout 30 minutes, *largeFam* problems).

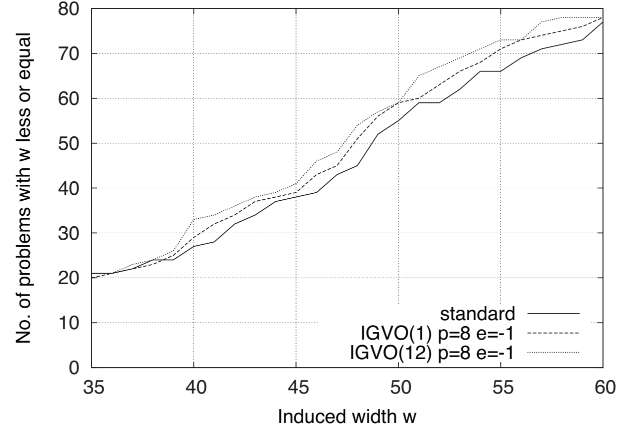

Figure 6: Magnified cumulative plot comparing standard MinFill, single-threaded IGVO, and 12-threaded IGVO on *type4* problems. Timeout 1 hour.

half of the table), MinComplexity ranking function outperforms MinFill in finding smallest state space size in more cases (127 vs 108), while MinFill has smaller average state space size.

The Effect of Randomization

Strictly greedy algorithms can get stuck in local minima, even with random tie-breaking. This is particularly true when few ties occur while running GVO on a problem instance. To escape such minima, IGVO is augmented with the option of making a greedily suboptimal choice, where we don't pick a variable with lowest cost, but instead choose among a pool (size p) of lowest-cost variables. In particular, the probability of picking variable X is $VC(X)^e / \sum_{Y \in P} VC(Y)^e$, where e is a weighing constant and P is the pool. In this form, $e = 0$ yields a uniform distribution, $e < 0$ gives preference to variables with lower cost, and $e > 0$ leans towards variables with higher cost.

Figure 5 contrasts a standard MinFill implementation against IGVO with exponent $e = -1$ and varying pool sizes, focusing on a small part of the cumulative plot for readability. We observe that performance improves as the pool size increases from 1 to 4 to 8, but not much after that (if at all).

Varying the value of e does not have a major impact on the overall solution, but we found that $e = -1$ discovered the best width sooner than $e = 0$ or $e = 1$. For instance, with $p = 8$, the minimum width ordering was found after an average of 478 seconds with $e = -1$, compared to 888 seconds and 946 seconds for $e = 0$ and $e = -1$, respectively.

Comparison with Standard MinFill

We furthermore conducted a comparison of IGVO against a standard MinFill implementation[1]. In addition to the 242

		standard		IGVO(1)		IGVO(12)		
instance	n	iter	w	iter	w	iter	w	spd
100-18	7,435	6,430	51	26,689	48	324,664	48	12.2
110-19	7,303	3,852	54	13,005	52	158,806	51	12.2
120-18	8,656	6,594	47	17,604	45	211,830	44	12.0
120-25	9,171	3,789	57	14,576	56	176,156	54	12.1
130-20	9,328	3,167	60	12,541	58	154,647	57	12.3
130-22	10,271	3,747	56	13,107	52	168,635	52	12.9
140-23	10,998	2,318	61	7,654	60	91,576	57	12.0
150-22	11,799	2,636	57	8,423	54	99,949	53	11.9
170-18	12,186	2,202	59	6,913	55	82,756	55	12.0
170-22	14,641	2,795	58	8,147	56	97,423	54	12.0
190-19	15,433	3,044	56	6,473	54	77,287	52	11.9
190-21	15,125	5,284	43	9,545	42	115,048	40	12.1

Table 2: Exemplary results comparing standard MinFill with single- and 12-threaded IGVO ($p = 8, e = -1$) on *type4* problems. n denotes the number of problem variables, iter is the number of ordering iterations performed within 1 hour, w the best width found, spd the parallel speedup of IGVO(12).

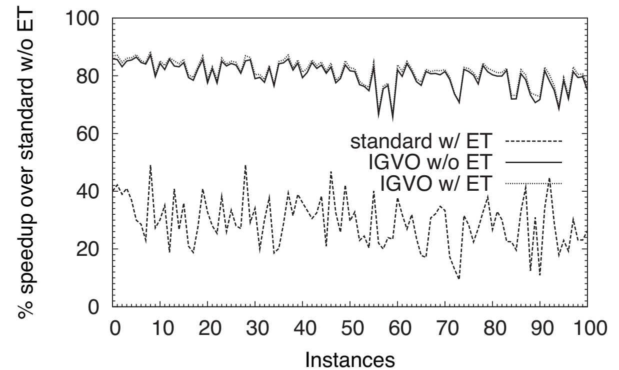

Figure 7: Time savings relative to standard MinFill across a subset of 100 *largeFam* instances (20,000 iterations each).

[1] An earlier version of our standard MinFill was used in the UAI-2010 competition with solver(s) that won first place in 4 categories, and was subsequently improved/enhanced, yielding the standard MinFill used in this paper. It eliminates all simplical vertices as described in Figure 2 and uses random tie-breaking without

			previous		new	
instance	n	k	w	space	w	space
110-21	7,675	5	*37*	16 TB	*33*	215 GB
140-20	9,355	5	*35*	10 TB	*28*	4 GB
180-21	14,157	5	*38*	9 TB	*31*	67 GB
200-18	15,319	5	*36*	19 TB	*30*	41 GB

Table 3: Four *type4* problems that were previously infeasible because of their space requirements, but are now solvable (n is number of problem variables, k max. domain size).

problem instances used before, we also ran experiments on 82 *type4* problems from the domain of genetic linkage analysis as used in (Kask, Dechter, and Gelfand 2010) and on which we will focus in the following. (Size ranges from several thousand to more than fifteen thousand variables; timeout was set to 1 hour.)

Figure 6 summarizes the results in the same cumulative manner as before; it includes standard MinFill as well as single- and 12-threaded IGVO ($p=8, e=-1$). Table 2 also lists detailed results for a number of instances. Focusing first on single-threaded execution, we observe the following:

- IGVO benefits from the strategy of pooling non-optimal choices and consistently finds orderings with lower width than standard MinFill, evidenced by the difference between the two respective curves in Figure 6 and the examples in Table 2.

- Efficient data structures and implementation allow IGVO to perform many more iterations – often more than 3 times as many – in the same time as the standard implementation (cf. Table 2), in spite of the additional overhead from pooling. Figure 7 provides further evidence of this speedup, with more than 80% time savings across a subset of 100 problem instances.

Parallelization. Table 2 and Figure 6 also include the results of running IGVO with 12 parallel threads (on dual 6-core CPUs, i.e. 12 cores). Apart from the fact that it returns further improved orderings, we see that the parallel algorithm completes roughly 12 times as many iterations as the single-threaded one, confirming fairly linear scaling as expected. (Speedups greater than 12 can be explained by the increasing impact of early termination as the minimum width improves with the number of iterations.)

Early Termination. Figure 7 also shows the relative impact on time of early termination. We see that it gives a significant speedup (between 20 and 40%) when applied to standard MinFill. In case of IGVO, however, which already incorporates a host of other optimizations and extensions, it only provides a small additional benefit – about 5% on average.

pooling. However, updating each vertex takes $O(deg^3)$ time as opposed to the $O(nf \cdot deg)$ of the optimized variant. In addition, the standard implementation is run iteratively and employs the early termination criteria.

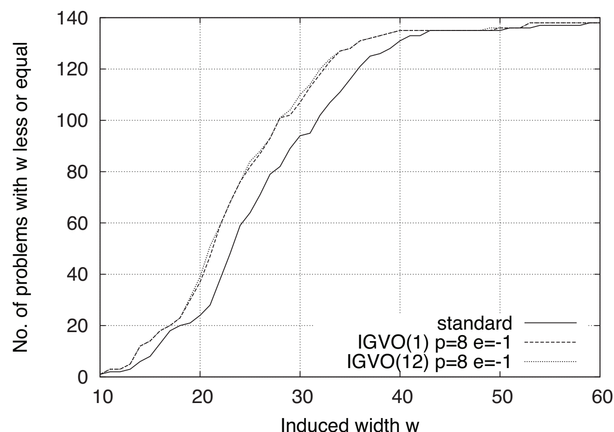

Figure 8: Cumulative plot comparing standard MinFill, single-threaded IGVO, and 12-threaded IGVO on protein folding problems. Timeout 1 hour.

		standard		IGVO(1)		IGVO(12)		
instance	n	iter	w	iter	w	iter	w	spd
pdb1e18	618	1.72E+5	37	8.61E+5	32	1.02E+7	31	11.8
pdb1gnt	443	3.28E+5	33	1.53E+6	26	1.82E+7	25	11.9
pdb1i24	337	4.46E+5	33	2.20E+6	27	2.62E+7	27	11.9
pdb1m6i	375	4.88E+5	32	2.06E+6	25	2.45E+7	25	11.9
pdb1nqe	457	3.82E+5	34	1.56E+6	31	1.81E+7	30	11.5
pdb1qpk	332	4.13E+5	32	2.38E+6	26	2.87E+7	26	12.1
pdb1c3r	636	1.48E+5	40	7.01E+5	32	8.33E+6	32	11.9
pdb1e3d	1298	4.09E+4	59	1.83E+5	53	2.30E+6	53	12.5
pdb1eg5	609	1.63E+5	37	7.46E+5	32	8.91E+6	32	12.0
pdb1fnn	658	1.82E+5	27	7.57E+5	24	8.99E+6	23	11.9
pdb1gnl	866	1.38E+5	34	4.94E+5	30	5.95E+6	29	12.0
pdb1h80	744	1.92E+5	33	6.69E+5	30	7.92E+6	29	11.9
pdb1i2m	919	1.07E+5	39	4.35E+5	34	5.09E+6	34	11.7
pdb1i7n	530	2.79E+5	29	1.17E+6	26	1.41E+7	25	12.0
pdb1jet	457	2.40E+5	36	1.25E+6	31	1.46E+7	30	11.7

Table 4: Exemplary results comparing standard MinFill with single- and 12-threaded IGVO ($p = 8, e = -1$) on protein folding problems (max. domain size $k = 81$). n denotes the number of problem variables, iter is the number of ordering iterations performed within 1 hour, w the best width found, spd the parallel speedup of IGVO(12).

Pushing the Boundaries of Feasibility

Table 3 lists four problem instances whose massive memory requirements of many terabytes made solving them previously impossible, even for powerful schemes utilizing external memory (Kask, Dechter, and Gelfand 2010). Running IGVO(12) for one hour, however, yielded good orderings that allow for solving these instances with mere gigabytes of hard disk space, a significant improvement.

Protein Folding Problems

We conducted another set of experiments on 138 protein folding / side-chain prediction problems modeled as Markov networks (Yanover and Weiss 2002) – here the max. do-

main size is $k = 81$, which makes finding a low-width ordering particularly important. Figure 8 shows a cumulative summary plot while Table 4 presents select results in detail. Again we see that IGVO is able to perform many more iterations than the standard implementation and consistently finds orderings with lower width, often by a considerable margin. However, the problems are small enough (mostly less than 1,000 variables) that the move to parallel IGVO does not significantly improve the width of the returned orderings.

Conclusions

The paper studies the iterative application of stochastic greedy ordering schemes showing that it may currently be the primary (and perhaps the only) practical scheme for finding low-treewidth decompositions. Such schemes are appealing because of their simplicity, flexibility, anytime nature and trivial parallelization. Within this class of schemes, we presented the *IGVO* algorithm and demonstrated its superiority to current variants in terms of both run-time and finding improved orderings. We showed empirically, on hard instances from computational biology, that IGVO's superiority can be attributed to: 1) Implementation efficiency - by utilizing new data-structures and good software engineering, IGVO can execute many more iterations in a given amount of time. This is particularly impressive since the baseline (standard) implementation is by itself highly efficient. 2) Randomization via pooling - the parametrized scheme for introducing stochasticity to the selection process allows IGVO to explore more diverse orderings than standard randomized GVO. For certain parameter settings this yields better results, despite the pooling overhead. 3) Early termination - this was shown to have a marginal effect on IGVO but much greater impact when the basic greedy scheme is less efficient.

We demonstrated additional speedup and found improved orderings when running IGVO on a multi-core machine. This additional gain allows us to perform exact computations on quite a few problems that could not be solved even when using external memory (i.e., solved by BEEM (Kask, Dechter, and Gelfand 2010)).

The IGVO algorithm can accommodate any greedy ranking function. Using IGVO we verified that MinFill is indeed a superior greedy heuristic when seeking low-treewidth decompositions. For problems with non-uniform domains, we observed that the MinComplexity ranking function is often superior to the MinFill heuristic. We also compared (but do not report here) the performance of IGVO to exact anytime algorithms (e.g. (Gogate and Dechter 2004) and stochastic local search schemes and found IGVO to superior to these schemes as well.

The contribution of this paper is really two-fold. First, it brings many known ideas into a single, anytime variable ordering framework. Second, it demonstrates benefit of iterative randomized greedy schemes and underscores the importance of certain design choices when using such schemes.

Acknowledgments

This work was partially supported by NSF grants IIS-0713118, IIS-1065618 and NIH grant 5R01HG004175-03.

References

Arnborg, S.; Corneil, D.; and Proskourowski, A. 1987. Complexity of finding embeddings in a k-tree. *SIAM Journal of Discrete Mathematics.* 8:277–284.

Bacchus, F.; Dalmo, S.; and Piassi, T. 2003. Value elimination: Bayesian inference via backtracking search. In *Uncertainty in AI (UAI03)*.

Bodlaender, H. L. 2007. Treewidth: structure and algorithms. In *SIROCCO'07*, 11–25.

Bodlaender, H. L. 2009. Kernelization: New upper and lower bound techniques. In *IWPEC*, 17–37.

Clautiaux, F.; Moukrim, A.; Négre, S.; and Carlier, J. 2004. Heuristic and meta-heuristic methods for computing graph treewidth. *RAIRO Operations Research* 38:13–26.

Darwiche, A. 2001. Recursive conditioning. *Artificial Intelligence* 125(1-2):5–41.

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

Dechter, R., and Meiri, I. 1989. Experimental evaluation of preprocessing techniques in constraint satisfaction problems. In *IJCAI'89*, 271–277.

Dechter, R., and Rish, I. 2002. Mini-buckets: A general scheme for approximating inference. *Journal of the ACM* 107–153.

Fishelson, M., and Geiger, D. 2003. Optimizing exact genetic linkage computations. *RECOMB* 114–121.

Gogate, V., and Dechter, R. 2004. A complete anytime algorithm for tree width. In *UAI04*.

Gogate, V., and Dechter, R. 2008. AND/OR importance sampling. In *UAI*, 212–219.

Kask, K.; Dechter, R.; and Gelfand, A. E. 2010. BEEM : Bucket elimination with external memory. In *UAI'2010*.

Kjaerulff, U. 1992. Optimal decomposition of probabilistic networks by simulated annealing. *Statistics and Computing* 2:2–17.

Larranaga, P.; Kuijpers, C.; Poza, M.; and Murga, R. 1997. Decomposing bayesian networks: triangulation of the moral graph with genetic algorithms. *Statistics and Computing* 7:19–34.

Robertson, N., and Seymour, P. 1983. Graph minors i. excluding a forest. *J. Combin. Theory, Ser. B* 35:39–61.

Yanover, C., and Weiss, Y. 2002. Approximate inference and protein-folding. In *NIPS*, 1457–1464.

Yedidia, J. S.; Freeman, W.; and Weiss, Y. 2005. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transaction on Information Theory* 2282–2312.