Refined Bounds for Instance-Based Search Complexity of Counting and Other #P Problems ⋆

Lars Otten and Rina Dechter

Bren School of Information and Computer Sciences
University of California, Irvine, CA 92697-3425, U.S.A.
{lotten,dechter}@ics.uci.edu

Abstract. We present measures for bounding the instance-based complexity of AND/OR search algorithms for solution counting and related $\#P$ problems. To this end we estimate the size of the search space, with special consideration given to the impact of determinism in a problem. The resulting schemes are evaluated empirically on a variety of problem instances and shown to be quite powerful.

1 Introduction

Inference algorithms like variable elimination have been known to be exponentially bounded by the tree width of a problem's underlying graph structure. More accurate bounds were derived by looking at the respective domain sizes of the variables in each cluster of a tree decomposition of the underlying graph [5], which was later also applied to search algorithms that explore the context-minimal AND/OR search graph [1].

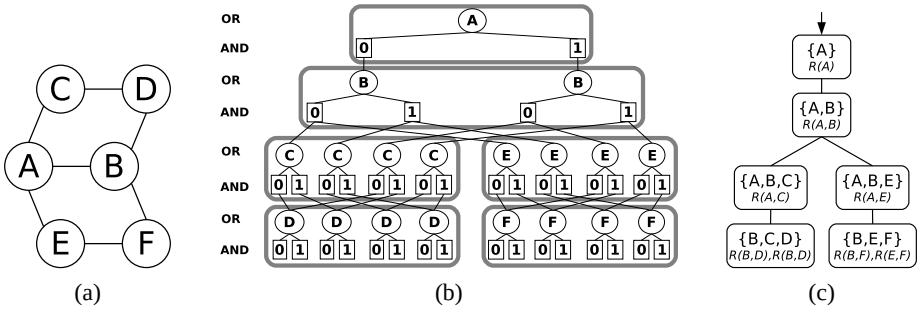
We recently introduced a more informed upper-bounding scheme, that selectively takes determinism into account [6]. We demonstrated its effectiveness empirically over a set of Bayesian networks and showed that the bounds it provides can in some cases be better by orders of magnitude. These tighter bounds allow us, for instance, to better predict parameters of algorithms (like variable orderings) ahead of time.

In this paper we extend our earlier work in four ways: First, we refine the bounding scheme by "reusing" relations during the estimation process, projecting their scope down to the currently relevant variables. Secondly, we introduce a simple scheme for lower bounding, that uses a sampling-based SAT solution counting algorithm. Thirdly, we show that these schemes are applicable to constraint networks by presenting experiments on various sets of constraint problem instances. Finally, we investigate our bounds' ability to discriminate between different variable orderings and demonstrate that they are indeed informative in this respect.

2 Bounding Search Space Size

We will assume a *graphical model*, given as a set of variables $X = \{x_1, \ldots, x_n\}$, their finite domains $D = \{D_1, \ldots, D_n\}$, a set of functions or relations $R = \{r_1, \ldots, r_m\}$, each of which is defined over a subset of X, and a combination operator (join, sum or

⋆ This work is supported in part by NSF grant IIS-0713118 and NIH grant R01-HG004175-02.



(a) (b) (c)

Fig. 1: (a) An example problem graph, (b) its AND/OR search graph along ordering $d = A, B, C, D, E, F$, and (c) the corresponding bucket tree decomposition.

product) over all functions. The scope of a relation r_j, denoted $scope(r_j)$, is the subset of X on which r_j is defined, its tightness t_j is the number of valid tuples in the relation. A *constraint satisfaction problem* (CSP) is then a special kind of graphical model.

Given a variable ordering d, we furthermore assume the usual definition of a *tree decomposition*, decomposing the variables and relations into clusters of a certain tree width w (the max. number of variables in a cluster). If the variables in each cluster are covered by the scopes of the cluster's relations, we have a *hypertree decomposition*, with associated hypertree width hw (the max. number of functions in a cluster). It is known that, if $k = \max_i |D_i|$ and $t = \max_j t_j$, the time and space complexity of processing a tree decomposition is dominated by k^w, while a hypertree decomposition can yield a solution with time and space complexity dominated by t^{hw} [3].

AND/OR search, on the other hand, is a novel method to exploit problem decomposition during search. It introduces *AND* nodes into the search space that allow capturing the independence of subproblems. If we also apply context-based caching of identical subproblems, it is easy to see that the resulting AND/OR search space has a one-to-one correspondence to a (bucket) tree decomposition along the same ordering (cf. Fig. 1). Accordingly, similar asymptotic bounds can be proven [1].

For a more refined analysis of problem complexity, we can determine the size of the search space as follows: For each cluster of the bucket tree C_k, containing variables $X_k \subseteq X$ and relations $R_k \subseteq R$, we multiply the domain sizes of the variables in X_k – this represents all possible value combinations of the variables in X_k. Summing over all clusters we obtain an upper bound on the number of nodes in the AND/OR search space, denoted $twb := \sum_{k=1}^{n} \prod_{x_i \in X_k} |D_i|$ (this is essentially a fine-grained version of the asymptotic, tree width-exponential bound). The bound quality will depend on the degree of determinism present in the problem, which is not reflected in twb, but which can cause significant pruning of the search space in practice.

If we are working on a hypertree decomposition, we can take the product over the tightness of each relation in a cluster as an upper bound on the number of search nodes in that cluster and sum over clusters – a fine-grained version of the asymptotic hypertree width bound, thus accounting for determinism. However, since relation scopes typically overlap, this will be far from optimal. We therefore start from the twb bound,

Algorithm *GreedyCovering*
Input: Set of variables $X = \{x_1, \ldots, x_r\}$ and set of relations $R = \{r_1, \ldots, r_s\}$, with x_i having domain size $|D_i|$ and r_j having tightness t_j
Output: A subset of R (a partial covering of X)
Init: *Uncov* $:= X$, *Covering* $:= \emptyset$
(1) Find j^* that minimizes $q_j = t_j / \prod_{x_k \in I_j} |D_k|$, where $I_j = \textit{Uncov} \cap scope(r_j)$.
(2) If $q_{j^*} \geq 1$, terminate and return *Covering* .
(3) Add r_{j^*} to *Covering* and set *Uncov* $:=$ *Uncov* $\setminus scope(r_{j^*})$.
(4) If *Uncov* $= \emptyset$, terminate and return *Covering* .
(5) Goto (1) .

Algorithm *Compute-hwb*
Input: A bucket tree decomposition with clusters $C_1, \ldots, C_n$, where cluster C_k contains variables $X_k \subseteq X$ and relations $R_k \subseteq R$
Output: The bound hwb on the size of the search space
Init: $hwb := 0$
(1) **for** $i = 1$ to n:
(2) $R := R_k$.
(3) For every relevant relation r from the ancestral buckets, project it onto $scope(r) \cap X_k$ and add it to R with updated tightness t'_r .
(4) $G :=$ *GreedyCovering*(X_k, R) .
(5) $hwb \mathrel{+}= \prod_{r_j \in G} t_j \cdot \prod_{x_i \in X_k \setminus G} |D_i|$.
(6) end **for**.
(7) Return hwb .

Fig. 2: Greedy covering algorithm and procedure to compute the overall bound hwb.

the product of variable domains, and iteratively pick relations whose tightness we can use to improve the bound, greedily covering variables with relations, similar to a SET COVER problem [4]. This results in the algorithm *GreedyCovering* given in Fig. 2.

Propagating Cluster Size Downwards. When considering relations for the covering of variables X_k in cluster C_k, we can refine the above scheme even further: We collect all relations from the ancestral clusters in the rooted tree decomposition and project them down to X_k. This can be seen as propagation of information down the search tree. In practice, we found that for some problem instances it will decrease the bound by up to 30%.

Overall Bound and Complexity. The resulting overall algorithm *Compute-hwb* is given in Fig. 2. It computes the upper bound hwb on the number of nodes in the AND/OR search space. Its complexity can be shown to be time $\mathcal{O}(n \cdot m \cdot (t + w))$ and space $\mathcal{O}(m + t)$, where n and m are the number of variables and relations, respectively, w is the tree width of the problem along the given ordering, and t is the maximal tightness as before (proof in [7]).

Lower Bounds on Cluster Size. To obtain a lower bound on the size of the search space, we employ a different scheme: In each cluster we generate a SAT formula from all relevant relations (i.e., from within the current cluster and ancestral ones). We encode the invalid tuples of each relation as the nogoods of the of the SAT formula, thus the number of SAT solutions will correspond to the number of valid nodes in the cluster. We feed each cluster's formula to the sampling-based SAT solution counting algorithm SampleSearch-LB [2], which gives a (probabilistic) lower bound. To get a lower bound on the overall number of search nodes, we again sum the cluster bounds. We call this bound $satb$.

3 Experimental results

We ran a variety of empirical tests on a large set of different problem instances from various domains. Here we present selected results, for the full set we refer to [7]. For every problem instance, we report the number of variables n, the number of relations

Table 1: twb, hwb, and $satb$ bounds compared to true search space size $\#cm$.

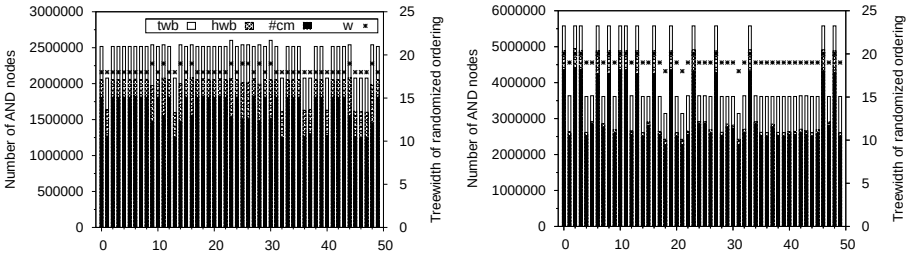
instance	n	m	k	r	w	tr	twb	hwb	$satb$	$\#cm$	Q_t	Q_h	Q_s
pret-60	60	40	2	3	4	0.50	1,534	1,102	839	998	1.51	1.10	0.89
pret-150	150	100	2	3	4	0.50	3,934	2,862	2,303	2,598	1.54	1.10	0.84
ssa-0432-003	435	738	2	5	31	0.75	4,244,330	2,059,616	1,116,669	1,868,283	2.27	1.10	0.60
ssa-2670-130	1359	2366	2	5	31	0.75	160,631,566	123,388,312	104,689,598	106,638,207	1.51	1.16	0.98
ssa-7552-038	1501	2444	2	6	63	0.75	308,861,278	115,499,146	6,815,140	36,718,327	8.41	3.15	0.19
ssa-7552-158	1363	1985	2	5	31	0.50	90,702	74,406	56,863	69,365	1.31	1.07	0.82
ssa-7552-159	1363	1983	2	5	31	0.50	92,238	73,586	48,929	68,694	1.34	1.07	0.71
BN_105	40	44	2	21	18	0.62	2,477,054	363	69	131	18909	2.77	0.53
BN_107	40	46	2	21	21	0.62	29,983,742	1,643	191	272	110234	6.04	0.70
BN_109	40	46	2	20	20	0.62	13,054,974	4,052	1,309	2,531	5158	1.60	0.52
BN_111	40	45	2	20	19	0.62	8,406,270	2,299	465	979	8587	2.35	0.47
BN_113	40	47	2	21	21	0.62	18,916,350	2,752	336	630	30026	4.37	0.53
aim-50-1-6-sat-1	50	77	2	3	18	0.88	2,517,118	2,053,046	931,492	1,813,906	1.39	1.13	0.51
aim-50-1-6-sat-2	50	76	2	3	16	0.88	767,678	626,955	73,180	551,659	1.39	1.14	0.13
aim-50-1-6-sat-3	50	78	2	3	20	0.88	4,742,590	3,859,278	2,023,465	3,848,835	1.23	1.00	0.53
aim-50-1-6-sat-4	50	77	2	3	19	0.88	3,615,166	2,616,824	2,079,752	2,532,968	1.43	1.03	0.82
aim-50-1-6-unsat-1	50	69	2	3	15	0.88	377,502	256,482	26,806	211,168	1.79	1.21	0.13
aim-50-1-6-unsat-2	50	77	2	3	19	0.88	3,484,734	2,551,090	16,995	1,908,441	1.83	1.34	0.01
aim-50-1-6-unsat-3	50	70	2	3	17	0.88	1,190,910	971,254	43,382	685,060	1.74	1.42	0.06
aim-50-1-6-unsat-4	50	76	2	3	20	0.88	7,236,702	5,195,870	2,386,893	3,873,236	1.87	1.34	0.62

m, the maximum variable domain size k, the maximum relation arity r, and the median tightness ratio tr over all its relations, defined as the ratio of valid tuples in a (full) relation table.

We build a bucket tree decomposition of the problem along a minfill ordering and report the tree width w. We then compute and report our bounds twb, hwb, and $satb$. Lastly we record the exact size of the AND/OR search graph, denoted $\#cm$. To make comparing values easier, we also report the ratios $Q_t = \frac{twb}{\#cm}$, $Q_h = \frac{hwb}{\#cm}$, and $Q_s = \frac{satb}{\#cm}$. The results are shown in Table 1. We note that computation of twb and hwb is performed within milliseconds, while $satb$ can take up to three seconds for ssa-7552-038 on our 2.66 GHz system (with 1000 samples generated in each cluster).

Bound Tightness. Going from twb to hwb, i.e., exploiting determinism, yields significantly tighter bounds across all instances, from, for instance, a 28% decrease on the *Pret* instances or 18% on aim-50-1-6-sat-2 to several orders of magnitude on the *BN* instances (which were generated with forced determinism and are thus amenable to our method). In many cases the hwb bound is indeed quite tight when compared to $\#cm$, getting to within 10% on the *Pret* and some *SSA* instances. Overall, the quality of the hwb bound seems to decrease with growing problem complexity. The current results for $satb$ are somewhat less impressive at this point, often being more than 50% smaller than $\#cm$ – yet they can sometimes give a rough idea and, on top of that, set the stage for future improvements.

Impact of Orderings. To investigate the power of our bounds in predicting good orderings, we processed the same problem instance 50 times along a randomized minfill ordering, each time computing twb and hwb as well as recording $\#cm$. The results for two instances are presented in Fig. 3. We find that hwb can provide valuable information: On aim-50-1-6-sat-4 the twb bound can not distinguish between orderings with tree width 19, yet hwb captures the different $\#cm$ values rather accurately. On aim-50-1-6-sat-1, some orderings yield a higher tree width of 19, yet have a smaller search space size $\#cm$, which is correctly indicated only by hwb.



(a) aim-50-1-6-sat-1 (b) aim-50-1-6-sat-4

Fig. 3: Plots of the twb and hwb bounds versus the true search space size $\#cm$ on two problem instances, each over 50 randomized minfill variable orderings. Also shown is the tree width w for each ordering, which is plotted against a separate scale on the right.

4 Summary & Future Work

We have previously introduced a scheme that extends known methods for bounding the size of the search space by taking determinism in the relation specification into account [7]. In this work we expand upon this in four ways: We account for propagation of determinism down the search tree by reconsidering and projecting relations, we develop an approach for lower bounding search space size, we show empirically the applicability to constraint networks, and we demonstrate the bounds' ability to discriminate between variable orderings. Our experimental results show that the upper bounds we obtain can be quite tight and provide valuable information; the lower bounds, however, still leave room for improvement at this point.

We believe our bounding scheme can be extended to optimization tasks by using the cost function itself. By dynamically adapting the bounds throughout the search process, we plan to allow for run time parameter updates. Finally, recent advances in sampling-based counting should also allow us to improve the quality of the lower bounds.

References

1. R. Dechter and R. Mateescu: AND/OR Search Spaces for Graphical Models. In *Artificial Intelligence* **171** (2007): 73–106.
2. V. Gogate and R. Dechter: Approximate Counting by Sampling the Backtrack-free Search Space. In *Proceedings of AAAI'07*.
3. G. Gottlob, N. Leone, and F. Scarcello: A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence* **124** (2000): 243–282.
4. D. S. Johnson: Approximation algorithms for combinatorial problems. In *Proceedings of STOC'73*: 38–49.
5. U. Kjærulff: Triangulation of Graphs – Algorithms Giving Small Total State Space. *Research Report R-90-09, Dept. of Mathematics and Computer Science*, Aalborg University (1990).
6. L. Otten and R. Dechter: Bounding Search Space Size via (Hyper)tree Decompositions. In *Proceedings of UAI'08*.
7. L. Otten and R. Dechter: Refined Bounds for Instance-Based Search Complexity of Counting and Other #P Problems. *Technical Report, University of California, Irvine* (2008).