ECAI 2012
Luc De Raedt et al. (Eds.)

doi:10.3233/978-1-61499-098-7-917

Advances in Distributed Branch and Bound

Lars Otten and **Rina Dechter**[1]

Abstract. We describe a distributed version of an advanced branch and bound algorithm over graphical models. The crucial issue of load balancing is addressed by estimating subproblem complexity through learning, yielding impressive speedups on various hard problems using hundreds of parallel CPUs.

1 Introduction

Parallelizing search algorithms such as branch and bound using *parallel tree search* [4], distributing conditioned subproblems to different CPUs, is highly challenging. Large variance and therefore unpredictability of subproblem complexities makes load balancing extremely elusive, which can be devastating to parallel performance. As documented in earlier work [citation withheld for anonymity], this is particularly difficult for advanced algorithms geared towards sequential execution that are typically far from embarrassingly parallel. One such state-of-the-art algorithm is AND/OR Branch and Bound [9], which has been very competitive in recent inference competitions.[2]

This paper reports on a new distributed version of AOBB running on a computational grid (a set of autonomous, loosely connected systems) using hundreds of CPUs – the only parallel search scheme in a general graphical model framework to date that we are aware of ([1] is related, but parallelizes variable elimination and only provides simulated results). Our load balancing is based on a complexity estimator learned offline from previously solved subproblems of the same problem class. Preliminary results on a set of instances from the domain of genetics are encouraging, in some cases reducing computation time from many days to less than an hour.

2 Background

AND/OR Branch and Bound (AOBB) is an adaptation of branch and bound search to the framework of AND/OR search spaces over graphical models such as Bayesian networks or weighted constraint satisfaction problems. It exploits conditional independencies through *problem decomposition* and avoids redundant computations via *caching* of context-identical subproblems; worst-case time and space complexity is exponential in the problem's induced width [3], which can imply exponential savings over traditional search spaces. A mini bucket heuristic [5] is used to provide upper bounds (assuming a maximization setting) on subproblem solutions which, together with a lower bound from the current best solution maintained by the algorithm, allows pruning of unpromising parts of the search space.

Our distributed implementation of AND/OR Branch and Bound is based on the notion of *parallel tree search* [4], where a search tree is explored centrally up to a certain depth and the remaining subtrees are processed in parallel. In our context we solve the resulting conditioned subproblems using a grid of computers. Figure 1 demonstrates this concept on an example problem with six variables: conditioning on A and B (in gray) yields eight independent subproblems.

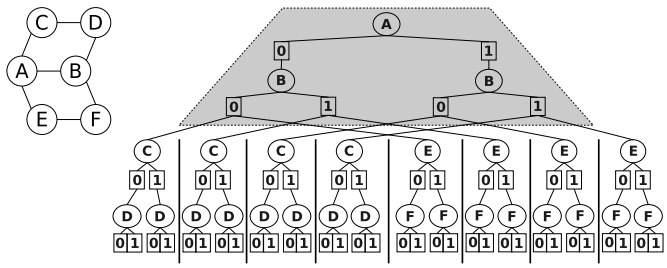

Figure 1: Example problem graph (left) and possible parallel search space with eight parallel subproblems (right).

The central decision in this parallelization scheme is clearly where to place the *parallelization frontier*, which will determine the number and shape of the parallel jobs. To establish the best possible overall performance this choice needs to ensure effective load balancing, i.e., spreading the parallel workload evenly across all available CPUs. The following section discusses our approach in more detail.

3 Load Balancing for Distributed AOBB

A first, natural choice for the parallelization frontier is a fixed depth d in the conditioning search space that ensures a sufficient number of subproblems to occupy all available CPUs. In practice, however, this is often detrimental: even when the underlying subgraph structure is identical across subproblems, the size of the explored subproblem search spaces rooted at the same depth d is far from uniform due, in large part, to the pruning power of AOBB. Thus often very few subproblems dominate the overall runtime (cf. results in Section 4).

Detecting and mitigating these extreme cases requires more detailed knowledge about a subproblem beforehand, namely we aim to estimate its complexity. Prior work in this area goes back to [7] and more recently [6], which predict the size of general backtrack trees through random probing. Similar schemes were devised for Branch and Bound algorithms [2], where search is run for a limited time and the partially explored tree is extrapolated. All of these, however, depend to a large extent on a substantial sample of the (sub)problem in question, which quickly becomes prohibitive in our setup with hundreds, if not thousands of subproblems to consider.

Our key progress in load balancing is due to an offline learning step similar in spirit to [8]: we collect a set of several thousand sample subproblems from past experiments, extract a number of features for each of them, and record their complexities using our AOBB algorithm. The features are structural (e.g., subproblem variable count and induced width) as well as cost function-related (e.g., subproblem upper/lower bound). We apply statistical feature selection and learn a linear regression model with subproblem log complexity as

[1] University of California, Irvine. {lotten,dechter}@ics.uci.edu
[2] cf. UAI' Inference Evaluation '10 and Pascal Inference Challenge '11

						Number of CPUs						
inst	n	k	w	h	seq	10	20	50	100	200	300	400
ped7	1068	4	32	90	26:11	02:49	01:29	00:39	00:21	00:12	00:09	00:09
ped9	1118	7	27	100	16:26	01:57	00:59	00:24	00:13	00:07	00:06	00:05
ped13	1077	3	32	102	28:42	02:51	01:28	00:42	00:24	00:16	00:13	00:13
ped19	793	5	25	98	105:11	13:48	07:38	03:17	01:56	01:14	00:50	00:42
ped31	1183	5	30	85	121:25	12:43	06:38	02:43	01:23	00:43	00:31	00:24
ped34	1160	5	31	102	12:34	02:05	00:54	00:24	00:13	00:08	00:06	00:05
ped41	1062	5	33	100	13:07	01:34	00:48	00:23	00:16	00:10	00:11	00:11
ped44	811	4	25	65	26:52	03:28	01:58	00:54	00:32	00:18	00:13	00:11
ped51	1152	5	39	98	46:13	04:54	02:31	01:06	00:36	00:22	00:21	00:19

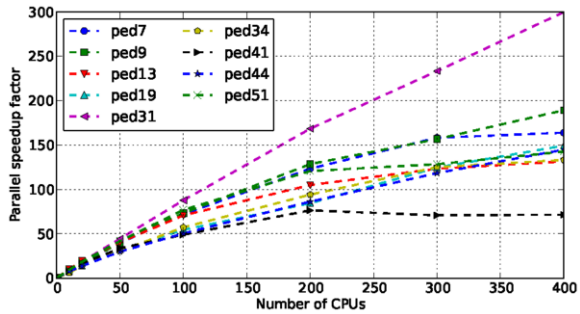

Figure 2: Parallel performance (left, times in *hh:mm*) and corresponding parallel speedup (right) on nine pedigree instances for varying number of CPUs. seq is time of sequential AOBB, n no. of problem variables, k max. domain size, w induced width, h guiding pseudo tree height.

the target, to account for the exponential nature of the search. The resulting regression model can then be used by the parallel scheme to very quickly compute complexity estimates; our policy is to iteratively grow the frontier by splitting the (estimated) most complex subproblem, until the desired number of subproblems is obtained.

4 Experimental Results

We note that "perfect" load balancing is not attainable in practice even if we had full prior knowledge of subproblem complexities (a hard problem we aim to solve as well), since splitting a given subproblem into its children often yields large jumps in complexity. This also makes perfect, linear speedup elusive, which is further impeded by grid-induced overhead and delays.

Overall parallel performance. Figure 2 shows parallel performance and speedup on nine very hard pedigree instances (encoding genetic haplotyping problems) for different number of CPUs. The hardest problems ped19 and ped31 in particular show impressive improvements from 4.5 and 5 days, respectively, to under one hour each. For easier problems the impact of the parallelization overhead is more pronounced and speedups level off somewhat, as expected.

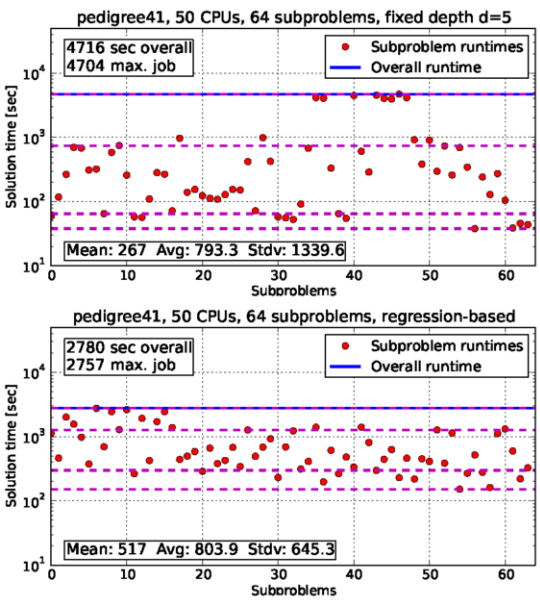

Figure 3: Subproblem statistics for fixed-depth (top) and regression-based frontier (bottom). Dashed lines: 0, 20, 80 and 100 percentile.

Load balancing. Figure 3 compares the two alternative policies for subproblem selection, fixed-depth (top) and using the complexity estimates to build the frontier (bottom). We notice a handful of subproblems that clearly dominate overall performance for the fixed-depth frontier (note the log scale); using the regression estimates avoids these extreme outliers, thereby reducing overall runtime by over 40%. A number of other test cases exhibited similar behavior.

5 Summary & Future Work

We have presented a new distributed branch and bound scheme over graphical models that works on hundreds of computers, to our knowledge the first of its kind. The crucial issue of load balancing is addressed through offline learning of a complexity model, which has yielded impressive speedups on several hard problem instances.

Ongoing and future research directions include extending and analyzing the quality of the complexity prediction as well as studying its applicability across problem domains. More generally we plan to evaluate the distributed scheme on a wider range of problems and investigate how varying levels of parallelism impact performance. For instance, given p CPUs we can generate $k \cdot p$ subproblems and assign k subproblems to each processor, exploiting stochasticity in subproblem runtime for better load balancing.

REFERENCES

[1] David Allouche, Simon de Givry, and Thomas Schiex, 'Towards parallel non serial dynamic programming for solving hard weighted csp', in *CP*, pp. 53–60, (2010).

[2] Gérard Cornuéjols, Miroslav Karamanov, and Yanjun Li, 'Early estimates of the size of branch-and-bound trees', *INFORMS Journal on Computing*, **18**(1), 86–96, (2006).

[3] Rina Dechter and Robert Mateescu, 'AND/OR search spaces for graphical models', *Artif. Intell.*, **171**(2-3), 73–106, (2007).

[4] Ananth Grama and Vipin Kumar, 'State of the art in parallel search techniques for discrete optimization problems', *IEEE Trans. Knowl. Data Eng.*, **11**(1), 28–35, (1999).

[5] Kalev Kask and Rina Dechter, 'A general scheme for automatic generation of search heuristics from specification dependencies', *Artif. Intell.*, **129**(1-2), 91–131, (2001).

[6] Philip Kilby, John Slaney, Sylvie Thiébaux, and Toby Walsh, 'Estimating search tree size', in *AAAI*, pp. 1014–1019. AAAI Press, (2006).

[7] Donald E. Knuth, 'Estimating the efficiency of backtrack programs', *Mathematics of Computation*, **29**(129), 121–136, (1975).

[8] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham, 'Empirical hardness models: Methodology and a case study on combinatorial auctions', *Journal of the ACM*, **56**(4), 1–52, (2009).

[9] Radu Marinescu and Rina Dechter, 'AND/OR Branch-and-Bound search for combinatorial optimization in graphical models', *Artif. Intell.*, **173**(16-17), 1457–1491, (2009).