Load Balancing for Parallel Branch and Bound*

Lars Otten and Rina Dechter

Bren School of Information and Computer Sciences
University of California, Irvine
{lotten,dechter}@ics.uci.edu

Abstract. A strategy for parallelization of a state-of-the-art Branch and Bound algorithm for weighted CSPs and other graphical model optimization tasks is introduced: independent worker nodes concurrently solve subproblems, managed by a Branch and Bound master node; the problem cost functions are used to predict subproblem complexity, enabling efficient load balancing, which is crucial for the performance of the parallelization process. Experimental evaluation on up to 20 nodes yields very promising results and suggests the effectiveness of the scheme. The system runs on loosely coupled commodity hardware, simplifying deployment on a larger scale in the future.

1 Introduction

This paper explores parallelization of combinatorial optimization tasks over graphical models like weighted or soft CSP problems and Bayesian networks. Specifically, we consider a state-of-the-art exact optimization algorithm, AND/OR Branch and Bound (AOBB). AOBB, which exploits independencies and unifiable subproblems, has demonstrated superior performance for these tasks compared with other state-of the art exact solvers [1] (e.g., it was ranked first or second in the UAI'06[1] and '08[2] evaluations).

To parallelize AOBB we use the established concept of parallel tree search [2] where the tree is explored centrally up to a certain depth and the remaining subtrees are solved in parallel. For graphical models this can be implemented straightforwardly by exploring the search space of partial instantiations up to a certain depth and solving the remaining conditioned subproblems in parallel. This approach has already proven successful for likelihood computation in Superlink-Online, which parallelizes cutset conditioning for linkage analysis tasks [3]. Our work differs in focusing on optimization and in exploiting the AND/OR paradigm, leveraging additional subproblem independence for parallelism. Moreover, we use the power of Branch and Bound in a central search space that manages (and prunes) the set of conditioned subproblems.

The main difference however is that, compared to likelihood computation, optimization presents far greater challenges with respect to load balancing. Hence the primary challenge in search tree parallelization is to determine the "cutoff", the *parallelization frontier*. Namely, we need a mechanism to decide when to terminate a branch in the central search space and send the corresponding subproblem to a machine on the network.

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

[1] http://ssli.ee.washington.edu/ bilmes/uai06InferenceEvaluation/

[2] http://graphmod.ics.uci.edu/uai08/

There are two primary issues: *(1)* Avoid *redundancies*: caching of unifiable subproblems is lost across the independently solved subproblems, hence some work might be duplicated; *(2)* Maintain *load balancing* among the grid resources, dividing the total work equally and without major idling periods. While introducing redundancy into the search space can be counterproductive for both tasks, load balancing is a far greater challenge for optimization, since the cost function is exploited in pruning the search space. Capturing this aspect is essential in predicting the size of a subproblem and thus the focus of this paper.

The contribution of this work is thus as follows: We suggest a parallel B&B scheme in a graphical model context and analyze some of its design trade-offs. We devise an estimation scheme that predicts the size of future subproblems based on cost functions and learns from previous subproblems to predict the extent of B&B pruning within future subproblems. We show that these complexity estimates enable effective load distribution (which was not possible via redundancy analysis only), and yield very good performance on several very hard practical problem instances, some of which were never solved before. Our approach assumes the most general master-worker scenario with minimal communication and can hence be deployed on a multitude of parallel setups spanning hundreds, if not thousands of computers worldwide. Our current empirical results were obtained on 20 networked desktop computers, but we believe the potential for scaling up is very promising.

Related work: The idea of parallelized Branch and Bound in general is not new, but existing work often assumes a shared-memory architecture or extensive inter-process communication [2, 4–7], or specific grid hierarchies [8]. Early results on estimating the performance of search go back to [9] and more recently [10], which predict the size of general backtrack trees through random probing. Similar schemes were devised for Branch and Bound algorithms [11]: B&B is run for a limited time and the partially explored tree is extrapolated. Our method, on the other hand, is not based on sampling or probing but only uses parameters available a priori and information learned from past subproblems which is facilitated through the use of depth-first branch and bound to explore the master search space.

The paper is organized as follows: Section 2 provides necessary definitions and concepts, while in Section 3 we outline our parallelized AOBB scheme and analyze its parameters through a set of initial experiments. Section 4 derives the complexity estimates required for load balancing, with which we obtain the experimental results in Section 5. Section 6 concludes.

2 Background

We assume the usual definitions of a *graphical model* as a set of functions over discrete variables, its *induced graph*, and *induced width*. In a *weighted constraint problem* (WCSP), for instance, we aim to find a complete assignment that minimizes the sum of all costs. Figure 1(a) depicts the primal graph of an example problem with six variables. The induced graph for the example problem along ordering $d = A, B, C, D, E, F$ is depicted in Figure 1(b), with two new induced edges, (B, C) and (B, E). Its induced width is 2. Note that different orderings will vary in their implied induced width; find-

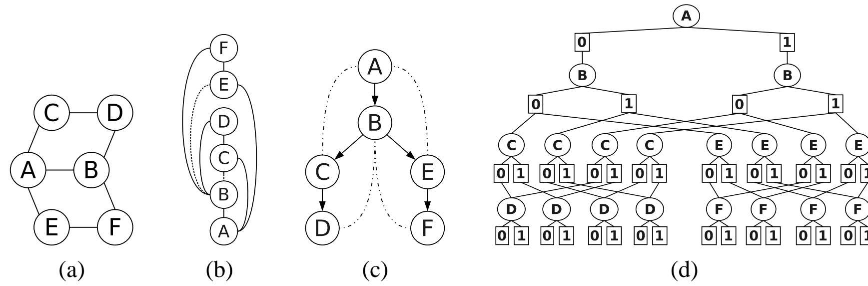

Fig. 1: (a) Example primal graph with six variables, (b) its induced graph along ordering $d = A, B, C, D, E, F$, (c) a corresponding pseudo tree, and (d) the resulting context-minimal AND/OR search graph.

ing an ordering of minimal induced width is known to be NP-hard, in practice heuristics like *minfill* are used to obtain approximations [12].

2.1 AND/OR Search Spaces

The concept of AND/OR search spaces has recently been introduced as a unifying framework for advanced algorithmic schemes for graphical models to better capture the structure of the underlying graph [13]. Its main virtue consists in exploiting conditional independencies between variables, which can lead to exponential speedups. The search space is defined using a *pseudo tree*, which captures problem decomposition:

Definition 1 (pseudo tree). *Given an undirected graph $G = (X, E)$, a* pseudo tree *of G is a directed, rooted tree $\mathcal{T} = (X, E')$ with the same set of nodes X, such that every arc of G that is not included in E' is a back-arc in $\mathcal{T}$, namely it connects a node in $\mathcal{T}$ to an ancestor in $\mathcal{T}$. The arcs in E' may not all be included in E.*

AND/OR Search Trees : Given a graphical model instance with variables X and functions F, its primal graph (X, E), and a pseudo tree $\mathcal{T}$, the associated *AND/OR search tree* consists of alternating levels of OR and AND nodes. OR nodes are labeled X_i and correspond to the variables in X. AND nodes are labeled $\langle X_i, x_i \rangle$, or just x_i and correspond to the values of the OR parent's variable. The structure of the AND/OR search tree is based on the underlying pseudo tree $\mathcal{T}$: the root of the AND/OR search tree is an OR node labeled with the root of $\mathcal{T}$. The children of an OR node X_i are AND nodes labeled with assignments $\langle X_i, x_i \rangle$ that are consistent with the assignments along the path from the root; the children of an AND node $\langle X_i, x_i \rangle$ are OR nodes labeled with the children of X_i in $\mathcal{T}$, representing conditionally independent subproblems. It was shown that, given a pseudo tree $\mathcal{T}$ of height h, the size of the AND/OR search tree based on $\mathcal{T}$ is $\mathcal{O}(n \cdot k^h)$, where k bounds the domain size of variables [13].

AND/OR Search Graphs : In an AND/OR search tree, different nodes may root identical subproblems. These nodes can be merged, yielding an *AND/OR search graph* of smaller size, at the expense of using additional memory during search. Some mergeable nodes can be identified by their *contexts*:

Definition 2 (context). *Given the pseudo tree $\mathcal{T}$ of an AND/OR search space, the context of an OR node X_i is the set of ancestors of X_i in $\mathcal{T}$, that are connected in the primal graph to X_i or its descendants (in $\mathcal{T}$). The context of X_i separates the subproblem below X_i from the rest of the network. Merging all context-mergeable nodes in the AND/OR search tree yields the* context minimal *AND/OR search graph [13].*

Proposition 1. *[13] Given a graphical model, its primal graph G, and a pseudo tree $\mathcal{T}$, the size of the context-minimal AND/OR search graph is $\mathcal{O}(n \cdot k^{w^*})$, where w^* is the induced width of G over a depth-first traversal of $\mathcal{T}$ and k bounds the domain size.*

Example 1. Figure 1(c) depicts a pseudo-tree extracted from the induced graph in Figure 1(b) and Figure 1(d) shows the corresponding context-minimal AND/OR search graph. Note that the AND nodes for B have two children each, representing independent subproblems and thus demonstrating problem decomposition. Furthermore, the OR nodes for D (with context $\{B, C\}$) and F (context $\{B, E\}$) have two edges converging from the AND level above them, signifying caching.

Weighted AND/OR Search Graphs : Given an AND/OR search graph, each edge from an OR node X_i to an AND node x_i can be annotated by *weights* derived from the set of cost functions F in the graphical model: the weight $l(X_i, x_i)$ is the sum of all cost functions whose scope includes X_i and is fully assigned along the path from the root to x_i, evaluated at the values along this path. Furthermore, each node in the AND/OR search graph can be associated with a *value*: the value $v(n)$ of a node n is the minimal solution cost to the subproblem rooted at n, subject to the current variable instantiation along the path from the root to n. $v(n)$ can be computed recursively using the values of n's successors [13].

2.2 AND/OR Branch and Bound

AND/OR Branch and Bound is a state-of-the-art algorithm for solving optimization problems over graphical models. Assuming a minimization task, it traverses the context-minimal AND/OR graph in a depth-first manner while keeping track of a current upper bound on the optimal solution cost. It interleaves forward node expansion with a backward cost revision or propagation step that updates node values (capturing the current best solution to the subproblem rooted at each node), until search terminates and the optimal solution has been found [13].

3 Parallel Setup and Scheme

We assume a very general parallel framework in which autonomous hosts are loosely connected over some network – in our case we use ten dual-core desktop computers, with CPU speeds between 2.33 and 3.0 GHz, on a local Ethernet, thus allowing experiments with up to 20 parallel nodes. We impose a *master-worker* hierarchy on the computers in the network, where a special *master* node runs a central process to coordinate the *workers*, which cannot communicate with each other. This general model is

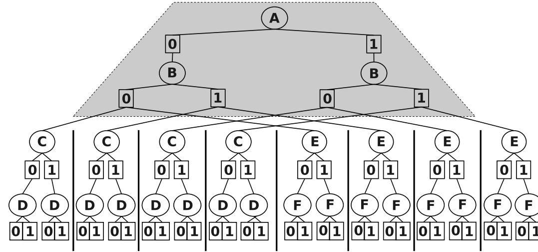

Fig. 2: Application of the parallelization scheme to the example problem from Figure 1, with the master search space (marked gray) and eight independent subproblems.

chosen to accommodate a wide range of parallel resources, where direct node communication is often either prohibitively slow or entirely impossible; it also facilitates flexible deployment on geographically dispersed, heterogeneous resources in the future.

The setup is similar to Superlink-Online [3], which has been very successful in using large-scale parallelism in likelihood algorithms for genetic linkage analysis, or SETI@home [14], which uses Internet-connected PCs around the world to search through enormous amounts of radio data. Like Superlink-Online, our system is implemented on top of the *Condor* grid middleware [15].

3.1 Parallel AND/OR Branch and Bound

Given a reasoning problem over a graphical model instance and a pseudo tree $\mathcal{T}$, a straightforward approach to parallelize the search process is to have the master process explore a *start pseudo tree*:

Definition 3 (start pseudo tree, parallelization frontier). *Given an undirected graph $G = (X, E)$, a directed rooted tree $\mathcal{T}_c = (X_c, E_c)$, where $X_c \subset X$, is a* start pseudo tree *if it has the same root as, and is a subgraph of some pseudo tree of G. Given a start pseudo tree T_c, we refer to the set of variables corresponding to the leaf nodes of T_c as the* parallelization frontier. *Each variable in the parallelization frontier roots a collection of subproblems characterized by value assignments along the path from the root to the variable.*

Example 2. Consider again the AND/OR search graph in Figure 1(d). Given a start pseudo tree having A and B, we can illustrate the parallelization scheme through Figure 2: the search space of the master process is marked in gray, and each of the eight independent subproblems rooted at C or E can be solved in parallel. (Notice, however, that some redundancy is introduced.)

3.2 Master Process Details

As a Branch and Bound algorithm, the master implements node expansion (or exploration) and propagation as outlined in the following (see [16] for details):

Table 1: Results on hard pedigree instances with $p = 15$ workers (timeout 24 hours). N is the number of problem variables, k the max. domain size, w the induced width along the chosen minfill ordering, and h the height of the corresponding pseudo tree. T_s is the solution time (in seconds) of sequential AOBB graph search on a single 3.0 GHz processor. The parallel solution time T_p is given for varying cutoff depth d.

						d=4	d=5	d=6	d=7	d=8	d=9	d=10	d=11
inst.	N	k	w	h	T_s	T_p	T_p	T_p	T_p	T_p	T_p	T_p	T_p
ped7	1068	4	32	90	19,114	21,334	8,343	4,038	**3,352**	3,610	4,560	4,675	7,073
ped13	1077	3	32	102	2,752	**379**	519	504	662	1,184	2,166	4,342	8,631
ped19	793	5	25	98	*time*	47,600	30,781	27,797	**27,327**	39,282	47,568	64,148	81,103
ped31	1183	5	30	85	77,580	68,472	47,089	46,837	43,097	41,286	22,582	**15,230**	15,313
ped41	1062	5	33	100	14,643	4,250	3,069	**2,173**	2,251	2,881	4,476	7,662	11,159
ped51	1152	5	39	98	*time*	*time*	79,131	**65,818**	*time*	*time*	*time*	72,218	83,011

Exploration. The master process explores the AND/OR graph in a depth-first manner guided by the start pseudo tree T_c. Upon expansion of a node n it consults a heuristic lower bound $lb(n)$ to make pruning decisions, where the computation of the upper bound $ub(n)$ can take into account previous subproblem solutions. If $lb(n) \geq ub(n)$, the current subtree can be pruned. Exploration is halted when the parallelization frontier is reached. The master then sends the respective subproblem, given by the subproblem root variable and its context instantiation, to a worker node.

Propagation. The master process also collects and processes subproblem solutions from the worker nodes. Upon receipt of a solved subproblem, its solution is assigned as the value of the respective node in the master search space and recursively propagated upwards towards the root, updating node values identical to sequential AOBB.

Assuming a fixed number of workers p, the master initially generates only the first p subproblems; whenever a worker finishes, its solution is propagated and the central exploration is resumed to generate the next subproblem.

3.3 Initial Experiments

The central decision is obviously where to place the *parallelization frontier*, i.e., at which point to cut off the master search space, which will determine subproblems sent to worker nodes. And while in the end the parallel scheme should make this decision automatically, we investigate the performance of the general parallelization approach through initial experiments with the cutoff set manually.

We consider two sets of hard problems, pedigree networks and mastermind game instances, both part of the UAI'08 evaluation[3]. Based on a pseudo tree computed from a minfill variable ordering, we enforce the parallelization frontier at constant depth d in the master search space. The *mini bucket* scheme is used to generate the heuristic function [17].

Haplotyping Problems : The first set of problems consists of pedigree networks from the area of human genetic analysis, specifically haplotyping problems. These can

[3] http://graphmod.ics.uci.edu/uai08/

Table 2: Results on Mastermind instances with $p = 10$ workers (columns as in Table 1).

instance	N	k	w	h	T_s	d=5 T_p	d=6 T_p	d=7 T_p	d=8 T_p	d=9 T_p	d=10 T_p
mm_03_08_05-0011	3612	2	37	89	9,715	1,698	**1,443**	1,540	1,510	2,464	3,765
mm_03_08_05-0012	3612	2	37	81	7,568	2,386	2,146	**1,430**	1,498	1,649	2,720
mm_04_08_04-0000	2616	2	37	79	10,620	1,594	1,322	**1,306**	1,358	1,539	2,045
mm_06_08_03-0000	1814	2	31	72	12,595	1,798	**1,797**	1,798	1,820	1,905	2,090
mm_10_08_03-0000	2606	2	47	99	26,102	4,593	4,417	4,481	**3,866**	4,259	4,357
mm_10_08_03-0011	2558	2	47	101	84,029	13,413	11,052	11,279	**11,044**	10,887	10,483
mm_10_08_03-0012	2558	2	47	82	5,630	2,420	**1,357**	1,362	1,361	1,473	1,294
mm_10_08_03-0013	2558	2	46	99	10,385	4,229	4,339	2,489	2,460	**2,413**	2,536

be translated into a MPE task over a Bayesian network [18] or, by moving to the log domain, a weighted CSP. Looking at the results in Table 1, the parallel scheme seems effective in almost all cases that we tested: for instance, ped13 can be solved in less than 10 minutes for several values of d, whereas the sequential algorithm takes 45 minutes. ped31 takes close to 22 hours on a single computer, while the parallel scheme with $d = 10$ solves the problem in 4 hours and 15 minutes. Moreover, the parallel scheme was able to solve two instances, ped19 and ped51, on which the sequential algorithm timed out after 24 hours.

Mastermind Problems : Table 2 documents experiments on some hard Mastermind instances, with T_s from a few hours to almost a day. Similar to the pedigree instances, parallelization enabled significant improvements in overall running time. For example, with $d = 7$, mm_04_08_04-0000 went from 3 hours to little over 22 minutes, while mm_10_08_03-0011 improved from more than 23 hours to about 3 hours for $d = 9$.

As evidenced by the results above, our parallel scheme carries high potential, yet its performance depends highly on the chosen parallelization frontier. In making this decision we can identify the following three objectives: *(1)* Minimize the redundancy and the associated blowup in the search space, that is inherent to the conditioning scheme (cf. Example 2). *(2)* Balance the workload over all processing units, each of which should be utilized equally to optimize efficiency and improve overall running time. *(3)* Minimize overhead resulting from grid communication and resource management. We address the issue of redundancy next.

3.4 Practical Impact of Redundancy

Recall that enforcing the parallelization frontier can introduce redundancies into the search space, since caching is not possible across subproblem boundaries. We have therefore developed fine-grained expressions which use the underlying structure of the graphical model to analytically capture the size of conditioned subproblems and the overall parallel search space as a function of the cutoff depth d – for space reasons we have to refer to [16] for details.

We point out, however, that this reasoning only yields an upper bound on the search space size since it accounts neither for the pruning in AOBB nor for determinism, which causes early backtracking. The latter can be partially incorporated as described in [19],

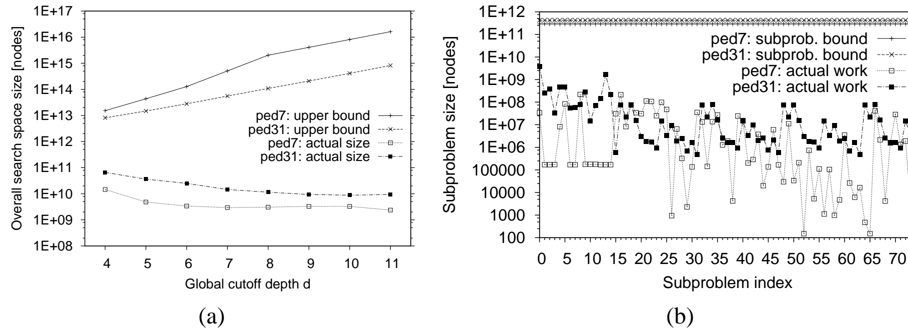

Fig. 3: (a) Search space upper bound vs. actual number of nodes, (b) subproblem bound and actual number of nodes for the first 75 subproblems of two pedigree instances.

yielding tighter bounds. As we show next, however, the main shortcoming of these bounds in the context of Branch and Bound actually lies in their disregard of the cost function, the guiding heuristic, and the associated pruning effect. Note that when we modified our scheme to compute likelihood (e.g., probability of evidence), in a very preliminary evaluation we observed a far better connection between our redundancy-based estimation and the true size of the search space. This suggests that structure-based analysis can play an important role for parallelizing likelihood queries, an aspect we plan to investigate in the future.

Redundancy : To investigate the practical impact of redundancy, we recorded the number of nodes generated in the master search space and across all subproblems for each parallel run; we also computed the upper bound from [16], extended to account for determinism [19]. Figure 3(a) contrasts these two measures as a function of d on two pedigrees: for both instances we observe that the exponential blowup of the upper bound with increasing d (due to redundancy) is not at all reflected in the actual number of nodes generated – evidently the pruning of AOBB is powerful enough to compensate for this and only a very small portion of the total search space is actually explored.

Subproblem Bounds : In order to evaluate the obtained subproblem bounds with respect to the size of the explored search space we recorded the number of nodes actually generated by the worker and contrast it with the respective precomputed subproblem bound. Exemplary results for two pedigree instances are shown in Figure 3(b), where the first 75 subproblems are plotted in order of their generation. We note that the upper bound doesn't change across subproblems; yet the actual size exhibits significant variance, going anywhere from a few thousand nodes (solvable in seconds) to many millions. Similar findings hold for other haplotyping and Mastermind instances.

In the following, we will thus focus on predicting the impact of pruning in AOBB, based on the problem's cost function and the resulting upper and lower bounds.

4 Predicting Subproblem Size Using the Cost Function

In this section we derive a scheme for estimating the size of the explored search space of a conditioned subproblem using parameters associated with the problem's cost function. Our goal is to discriminate between "easy" and "hard" subproblems to allow efficient load-balancing within our parallel scheme. In particular, we wish to enforce an upper bound on the complexity of subproblems (measured in the number of nodes expanded).

When considering a particular subproblem rooted at node n, we propose to estimate its complexity $N(n)$ (i.e., the number of nodes AOBB explores to solve it) as a function of the heuristic lower bound $L(n)$ as well as the upper bound $U(n)$, which can be computed based on earlier parts of the search space or through an approximation algorithm like local search; we will also use the height $h(n)$ of the subproblem pseudo tree. The general expression we propose has the form:

$$N(n) = b^{\frac{(U(n)-L(n))\cdot h(n)^{\alpha}}{inc}} \quad (1)$$

where b, inc, and α are constants. In the following we provide the rationale for this functional form and demonstrate how the free parameters can be learned as the search progresses.

4.1 Main Assumptions

We consider a node n that roots the subproblem $P(n)$. If the search space below n was a perfectly balanced tree of height D, with every node having exactly b successors, clearly the total number of nodes is $N = (b^{D+1} - 1)/(b-1) \approx b^D$.

However, even if the underlying search space is balanced, the portion expanded by B&B, guided by some heuristic evaluation function, is not: the more accurate the heuristic, the more focused around the optimal solution paths the search space will be. In state-based search spaces it is therefore common to measure effectiveness in post-solution analysis via the *effective branching factor* defined as $b = \sqrt[D]{N}$ where D is the length of the optimal solution path and N is the actual number of nodes generated [20].

Inspired by this approach, for a subproblem rooted at n we adopt the idea of approximating the explored search space by a balanced tree and express its size through $N(n) = b(n)^{D(n)}$. However, in place of the optimal solution path length (which corresponds to the pseudo tree height in our case), we propose to interpret $D(n)$ as the average leaf node depth $\bar{D}(n)$ defined as follows:

Definition 4 (Average leaf node depth). *Let $l_1, \ldots, l_j$ denote the leaf nodes generated when solving subproblem $P(n)$. We define the* average leaf node depth *of $P(n)$ to be $D(n) := \frac{1}{j}\sum_{k=1}^{j} d_n(l_k)$, where $d_n(l_i)$ denotes the depth of leaf node i relative to the subproblem root n.*

Figure 4 plots the number of nodes generated within a subproblem as a function of the average leaf node depth for ped13 ($d = 8$) and ped31 ($d = 10$), respectively. We can see a log-linear behavior (note the logarithmic vertical axis), thus supporting the general exponential form of $N(n) = b(n)^{D(N)}$.

We next aim to express $b(n)$ and $D(n)$ as functions of the subproblem parameters $L(n)$, $U(N)$, and $h(n)$ (using other parameters is subject to future research).

4.2 Estimating the Effective Branching Factor

For the sake of simplicity we assume an underlying, "true" effective branching factor b that is constant for all possible subproblems. We feel this is a reasonable assumption since all subproblems are conditioned within the same graphical model. Figure 4 exhibits some variance in complexity for fixed average leaf depth. This suggests modeling $b(n)$ as a random variable and assuming a normal distribution we can take the mean as the constant b. An obvious way to learn this parameter is then to average over the effective branching factors of previous subproblems, which is known to be the right statistic for estimating the true average of a population.

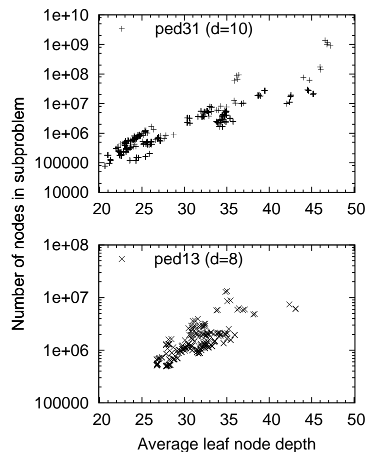

Fig. 4: Subproblem complexity vs. average leaf depth.

Estimating b for new Subproblem $P(n)$: Given a set of already solved subproblems $P(n_1), \ldots, P(n_r)$, we can compute $D(n_i)$ and derive effective branching degrees $b(n_i) = \sqrt[D(n_i)]{N(n_i)}$ for all i. We then estimate b through $\hat{b} = \frac{1}{r} \sum_{i=1}^{r} b(n_i)$.

4.3 Deriving and Predicting Average Leaf Depth

With each subproblem $P(n)$ rooted at a node n we associate a lower bound $L(n)$ based on the heuristic estimate and an upper bound $U(n)$ derived from the best solution from previous subproblems[4]. Both $L(n)$ and $U(n)$ are known before we start solving $P(n)$. We can assume $L(n) < U(n)$, since otherwise n itself could be pruned and $P(n)$ was trivially solved. We denote with $lb(n')$ and $ub(n')$ the lower and upper bounds of nodes n' within the subproblem $P(n)$ at the time of their expansion and similarly assert that $lb(n') < ub(n')$ for any expanded node n'.

Since the upper bound is derived from the best solution found so far it can only improve throughout the search process. Furthermore, assuming a monotonic heuristic function (that provides for any node n' a lower bound on the cost of the best solution path going through n'), the lower bounds along any path in the search space are non-decreasing and we can state that any node n' expanded within $P(n)$ satisfies:

$$L(n) \leq lb(n') < ub(n') \leq U(n)$$

Consider now a single path within $P(n)$, from n down to leaf node l_k , and denote it by $\pi_k = (n'_0, \ldots, n'_{d_n(l_k)})$, where $n'_0 = n$ and $d_n(l_k)$ is again the depth of l_k with respect to n (and hence $n'_{d_n(l_k)} = l_k$). We will write lb_i for $lb(n'_i)$ and ub_i for $ub(n'_i)$, respectively, and can state that $lb_i \geq lb_{i-1}$ and $ub_i \leq ub_{i-1}$ for all $1 \leq i \leq d_n(l_k)$ (note

[4] In the following we assume a graphical model with addition as the combination operator (a weighted CSP, for instance). Adaption to multiplication is straightforward.

that $lb_0 = L(n)$ and $ub_0 = U(n)$). An internal node n' is pruned iff $lb(n') \geq ub(n')$ or equivalently $ub(n') - lb(n') \leq 0$, hence we consider the (non-increasing) sequence of values $(ub_i - lb_i)$ along the path π_k ; in particular we are interested in the average change in value from one node to the next, which we capture as follows:

Definition 5 (Average path increment). *The* average path increment of π_k *within* $P(n)$ *is defined by the expression:*

$$inc(\pi_k) = \frac{1}{d_n(l_k)} \sum_{i=1}^{d_n(l_k)} ((ub_i - lb_i) - (ub_{i-1} - lb_{i-1})) \tag{2}$$

We note that l_k is a leaf node and assume $(ub_{d_n(l_k)} - lb_{d_n(l_k)}) = 0$, so the sum reduces to $(U(n) - L(n))$. Thus rewriting Expression 2 for $d_n(l_k)$ and averaging to get $D(n)$ as in Definition 4 yields:

$$D(n) = (U(n) - L(n)) \frac{1}{j} \sum_{k=1}^{j} \frac{1}{inc(\pi_k)} \tag{3}$$

We now define $inc(n)$ of $P(n)$ through $inc(n)^{-1} = \frac{1}{j} \sum_{k=1}^{j} \frac{1}{inc(\pi_k)}$, with which Expression 3 becomes $D(n) = (U(n) - L(n)) \cdot inc(n)^{-1}$, namely an expression for $D(n)$ as a ratio of the distance between the initial upper and lower bounds and $inc(n)$. Note that in post-solution analysis $D(n)$ is known and $inc(n)$ can be computed directly, without considering each π_j .

One more aspect that has been ignored in the analysis so far, but which is likely to have an impact, is the actual height $h(n)$ of the subproblem pseudo tree. We therefore propose to scale $D(n)$ by a factor of the form $h(n)^\alpha$; in our experiments we found $\alpha = 0.5$ to yield good results[5]. The general expression we obtain is thus:

$$\frac{D(n)}{h(n)^\alpha} = \frac{U(n) - L(n)}{inc(n)} \tag{4}$$

Predicting $D(n)$ for New Subproblem $P(n)$: Given previously solved subproblems $P(n_1), \ldots, P(n_r)$, we need to estimate $inc(n)$ in order to predict $D(n)$. Namely, we compute $inc(n_i) = (U(n_i) - L(n_i)) \cdot h(n_i)^\alpha \cdot D(n_i)^{-1}$ for $1 \leq i \leq r$. Assuming again that $inc(n)$ is a random variable distributed normally we take the sample average to estimate $inc^* = \frac{1}{r} \sum_{i=1}^{r} inc(n_i)$. Using Equation 4, our prediction for $D(n)$ is:

$$\hat{D}(n) = \frac{(U(n) - L(n)) \cdot h(n)^\alpha}{\hat{inc}} \tag{5}$$

Predicting $N(n)$ for a New Subproblem $P(n)$: Given the estimates $\hat{b}$ and $\hat{inc}$ as derived above, we will predict the number of nodes $N(n)$ generated within $P(n)$ as:

$$\hat{N}(n) = \hat{b}^{\,\hat{D}(n)} \tag{6}$$

The assumption that inc and b are constant across subproblems is clearly too strict, more complex dependencies will be investigated in the future. For now, however, even this basic approach has proven to yield good results, as we demonstrate in Section 5.

[5] Eventually α could be subject to learning as well.

Table 3: Results of the automated parallel scheme (ped: $p=15$, mm: $p=10$). T_s is the parallel time from Tables 1/2, T_{SLS} the time of sequential AOBB with one iteration of SLS to find an initial bound, T_p^* the best-performing fixed-depth cutoff from Tables 1/2, and T_{auto} the overall solution time of the automated parallel scheme.

instance	T_s	T_{SLS}	T_p^*	T_{auto}	instance	T_s	T_{SLS}	T_p^*	T_{auto}
ped7	19,114	19,309	3,352	2,783	ped31	77,580	37,844	15,230	3,910
ped13	2,752	2,796	379	359	ped41	14,643	13,999	2,173	2,251
ped19	*time*	*time*	27,372	10,611	ped51	*time*	*time*	65,818	59,915
mm_03_08_05-0011	9,715	2,943	1,443	1,085	mm_10_08_03-0000	26,102	9,876	3,866	7,604
mm_03_08_05-0012	7,568	2,030	1,430	1,584	mm_10_08_03-0011	84,920	39,761	10,044	6,846
mm_04_08_04-0000	10,620	7,807	1,306	3,076	mm_10_08_03-0012	5,630	2,489	1,357	754
mm_06_08_03-0000	12,595	259	1,797	228	mm_10_08_03-0013	10,385	5,337	2,413	2,128

4.4 Parameter Initialization

To find an initial estimate of both the effective branching factor as well as the average increment, the master process performs 15 seconds of sequential search. It keeps track of the largest subproblem $P(n_0)$ solved within that time limit and extracts $b(n_o)$ as well as $inc(n_0)$, which will then be used as initial estimates for the first set of cutoff decisions. As an additional preprocessing step, we perform a brief run of stochastic local search [21], which returns a solution that is not necessarily optimal, but in practice usually close to it. This is given as an additional input to the master process to provide initial lower bounds for the subproblem estimation.

5 Experiments

We ran our parallel AOBB scheme on the same set of problem instances as in Section 3.3, using the above prediction scheme to make the cutoff decision. The master process, however, now makes the cutoff decisions fully automatically: for each node n in the master search space, the complexity of $P(n)$ is estimated in $N^*(n)$ as described above; if this predicted value is less than a given threshold T, the subproblem below this node is submitted to the grid for solving; if $N^*(n) > T$, the children of n are generated within the master process and the estimation is recursively applied. For this set of experiments we set the threshold $T = 12 \cdot 10^8$, which corresponds to roughly 20 minutes of processing time and was deemed to be a good compromise between subproblem granularity and parallelization overhead.

Pedigree Networks : Results on the set of complex pedigree instances are given in Table 3. We can see that in all cases the automatic scheme does at least as good as the best fixed cutoff, in some cases even better. Again it is important to realize that T_p^* in Table 3 is the result of trying various fixed cutoff values d and selecting the best one, whereas T_{auto} requires no such "trial and error". In case of ped31 the SLS initialization is quite effective for the sequential algorithm, cutting computation from 21 to approx. 10 hours – yet the automated scheme improved upon this by a factor of almost 10, to just above one hour. Furthermore, for ped51 and in particular ped19, both of which could not be solved sequentially, T_{auto} marks a good improvement over T_p^* .

Mastermind Networks : Table 3 also includes results of the automated scheme for the set of Mastermind instances. Here the SLS preprocessing has a larger impact in

general, improving the sequential solution times considerably. And again, in most cases the automated scheme performs at least as well as the best fixed cutoff (determined after trying various depths). There are, however, some notable exceptions like mm_04_08_04, where T_{auto} is about two times T_p^* – our analysis here showed that the initial parameters for the subproblem prediction were too far off. We are therefore confident that a future, improved initialization scheme would alleviate these issues.

5.1 Subproblem Statistics

Figures 5(a) and (b) contain detailed subproblem statistics for the first 75 subproblem generated by the automated parallelization scheme on ped31 and ped51, respectively. Each plot shows actual and predicted number of nodes as well as the (constant) threshold that was used in the parallelization decision. The cutoff depth of the subproblem root is depicted against a separate scale to the right.

We see that the prediction scheme does not give perfect predictions (which was expected), but it reliably captures the trend. Furthermore, the actual subproblem complexities are all contained within an interval of roughly one order of magnitude, which is significantly more balanced than the results for fixed cutoff depths (cf. Figure 3(b)).

We also note that "perfect" load balancing is impossible to obtain in practice, because subproblem complexity can vary greatly from one depth level to the next along a single path. In particular, if a subproblem at depth d is deemed too complex, most of this complexity might stem from only one of its child subproblems at depth $d+1$, with the remaining ones relatively simple – yet solved separately. In light of this, we consider the above results very promising.

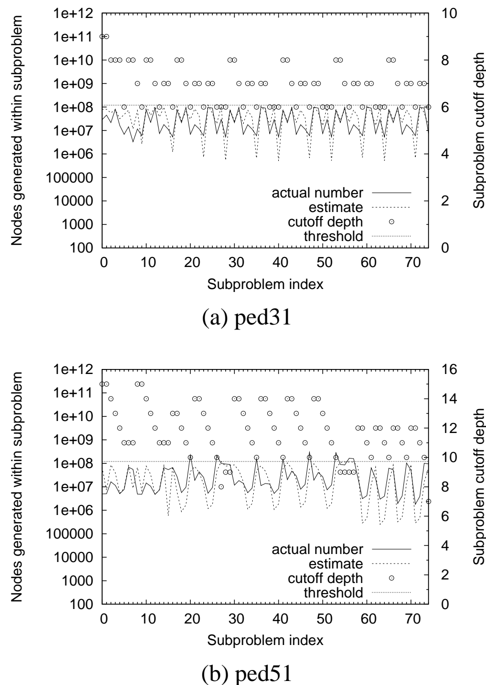

Fig. 5: Subproblem statistics for the first 75 subproblem of ped31 and ped51.

5.2 Performance Scaling

At this time we only have a limited set of computational resources at our disposal, yet we wanted to perform a preliminary evaluation of how the system scales with p, the number of workers. We hence ran the automated parallel scheme with $p \in \{5, 10, 15, 20\}$ workers and recorded the overall solution time in each case.

Figure 6 plots the relative speedup of the overall solution in relation to $p = 5$ workers. For nearly all instances the behavior is as expected, at times improving linearly with the number of workers, although not always at a 1:1 ratio. It is evident that relatively complex problem instances profit more from more resources; in particular ped51 sees a two-, three-, and fourfold improvement going to twice, thrice, and four times the number of workers, respectively. For simpler instances, we think the subproblem threshold of approx. 20 minutes is too close to the overall problem complexity, thereby inhibiting better scaling.

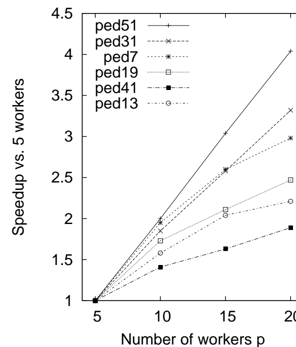

Fig. 6: Performance relative to $p = 5$ workers.

6 Conclusion & Future Work

This paper presents a new framework for parallelization of AND/OR Branch and Bound (AOBB), a state-of-the-art optimization algorithm over graphical models. In extending the known idea of parallel tree search to AOBB, we show that generating independent subproblems can itself be done through an AOBB procedure, where previous subproblem solutions are dynamically used as bounds for pruning new subproblems. The underlying parallel framework is very general and makes minimal assumptions about the available parallel infrastructure, making this approach viable on many different parallel and distributed resources pools (e.g. just a set of networked commodity hardware).

We addressed the two central objectives of any parallelization scheme – minimizing redundancy and optimizing load balancing – in the context of our AOBB algorithm. In particular we analyzed the relation of the above aspects to the *cutoff frontier*, the main parallelization parameter of our scheme. The very restricted communication in the assumed parallel architecture makes this the central decision, in contrast to more forgiving, "work-stealing" approaches that can still compensate later on [6, 7].

Experiments have shown that analytic expressions quantifying redundancy based only on the structure of the underlying search space are not effective in practice, since performance is dominated by the pruning power of AOBB. Our focus in this paper is therefore on deriving predictions that better capture the performance of AOBB using the problem's cost function, which underlies the algorithm's pruning decisions. We proposed to estimate the size of the explored search space using an exponential functional form using certain subproblem parameters. We then proposed a scheme for learning this function's free parameters from previously solved subproblems. We have demonstrated empirically the effectiveness of the estimates, leading to far better workload balancing and improved solution times on hard problems.

We acknowledge that this initial estimation scheme, while justified and effective, still includes some ad hoc aspects. We aim to advance the scheme by taking into account additional parameters and by providing firm theoretical grounds for our approach.

Besides extending the scheme itself, future work will also more thoroughly investigate the issue of scaling, using larger grid setups than what we had access to so far. We

also need to conduct more experiments on hard problems from various domains. Suitable ones must not be too easy to yield meaningful results with our advanced scheme, yet not too complex to run experiments in a reasonable time frame, which has proven somewhat elusive.

References

1. Marinescu, R., Dechter, R.: AND/OR Branch-and-Bound search for combinatorial optimization in graphical models. Artif. Intell. **173**(16-17) (2009) 1457–1491
2. Grama, A., Kumar, V.: State of the art in parallel search techniques for discrete optimization problems. IEEE Trans. Knowl. Data Eng. **11**(1) (1999) 28–35
3. Silberstein, M., Geiger, D., Schuster, A., Livny, M.: Scheduling mixed workloads in multigrids: The grid execution hierarchy. In: HPDC. (2006) 291–302
4. Gendron, B., Crainic, T.G.: Parallel branch-and-bound algorithms: Survey and synthesis. Operations Research **42**(6) (1994) 1042–1066
5. Anstreicher, K., Brixius, N., Goux, J.P., Linderoth, J.: Solving large quadratic assignment problems on computational grids. Mathematical Programming **91**(3) (2002) 563–588
6. Chu, G., Schulte, C., Stuckey, P.J.: Confidence-based work stealing in parallel constraint programming. In Gent, I., ed.: CP. Volume 5732 of Lecture Notes in Computer Science., Lisbon, Portugal, Springer-Verlag (September 2009) 226–241
7. Jurkowiak, B., Li, C.M., Utard, G.: A parallelization scheme based on work stealing for a class of sat solvers. J. Autom. Reasoning **34**(1) (2005) 73–101
8. Aida, K., Natsume, W., Futakata, Y.: Distributed computing with hierarchical master-worker paradigm for parallel branch and bound algorithm. In: CCGRID. (2003) 156–163
9. Knuth, D.E.: Estimating the efficiency of backtrack programs. Mathematics of Computation **29**(129) (1975) 121–136
10. Kilby, P., Slaney, J., Thiébaux, S., Walsh, T.: Estimating search tree size. In: AAAI, AAAI Press (2006) 1014–1019
11. Cornuéjols, G., Karamanov, M., Li, Y.: Early estimates of the size of branch-and-bound trees. INFORMS Journal on Computing **18**(1) (2006) 86–96
12. Kjaerulff, U.: Triangulation of graphs – algorithms giving small total state space. Technical report, Aalborg University (1990)
13. Dechter, R., Mateescu, R.: AND/OR search spaces for graphical models. Artif. Intell. **171**(2-3) (2007) 73–106
14. Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D.: Seti@home: an experiment in public-resource computing. Commun. ACM **45**(11) (2002) 56–61
15. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the Condor experience. Concurrency - Practice and Experience **17**(2-4) (2005) 323–356
16. Otten, L., Dechter, R.: Towards parallel search for optimization in graphical models. In: ISAIM. (2010)
17. Dechter, R., Rish, I.: Mini-buckets: A general scheme for bounded inference. Journal of the ACM **50**(2) (2003) 107–153
18. Fishelson, M., Dovgolevsky, N., Geiger, D.: Maximum likelihood haplotyping for general pedigrees. Human Heredity **59** (2005) 41–60
19. Otten, L., Dechter, R.: Refined bounds for instance-based search complexity of counting and other #P problems. In: CP. (2008) 576–581
20. Nilsson, N.J.: Artificial Intelligence: A New Synthesis. Morgan Kaufmann (1998)
21. Hutter, F., Hoos, H.H., Stützle, T.: Efficient stochastic local search for MPE solving. In: IJCAI. (2005) 169–174