Parallel Combinatorial Optimization with Decision Diagrams

David Bergman[1], Andre A. Cire[2], Ashish Sabharwal[3], Horst Samulowitz[3], Vijay Saraswat[3], and Willem-Jan van Hoeve[2]

[1] School of Business, University of Connecticut, Stamford, CT 06901
david.bergman@business.uconn.edu
[2] Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA 15213
{acire,vanhoeve}@andrew.cmu.edu
[3] IBM Watson Research Center, Yorktown Heights, NY 10598
{samulowitz,ashish.sabharwal,vsaraswa}@us.ibm.com

Abstract. We propose a new approach for parallelizing search for combinatorial optimization that is based on a recursive application of approximate Decision Diagrams. This generic scheme can, in principle, be applied to any combinatorial optimization problem for which a decision diagram representation is available. We consider the maximum independent set problem as a specific case study, and show how a recently proposed sequential branch-and-bound scheme based on approximate decision diagrams can be parallelized efficiently using the X10 parallel programming and execution framework. Experimental results using our parallel solver, DDX10, running on up to 256 compute cores spread across a cluster of machines indicate that parallel decision diagrams scale effectively and consistently. Moreover, on graphs of relatively high density, parallel decision diagrams often outperform state-of-the-art parallel integer programming when both use a single 32-core machine.

1 Introduction

In recent years, hardware design has increasingly focused on multi-core systems and parallelized computing. In order to take advantage of these systems, it is crucial that solution methods for combinatorial optimization be effectively parallelized and built to run not only on one machine but also on a large cluster.

Different combinatorial search methods have been developed for specific problem classes, including mixed integer programming (MIP), Boolean satisfiability (SAT), and constraint programming (CP). These methods represent (implicitly or explicitly) a complete enumeration of the solution space, usually in the form of a branching tree where the branches out of each node reflect variable assignments. The recursive nature of branching trees suggests that combinatorial search methods are amenable to efficient parallelization, since we may distribute sub-trees to different compute cores spread across multiple machines of a compute cluster. Yet, in practice this task has proved to be very challenging. For example, Gurobi, one of the leading commercial MIP solvers, achieves

H. Simonis (Ed.): CPAIOR 2014, LNCS 8451, pp. 351–367, 2014.

an average speedup factor of 1.7 on 5 machines (and only 1.8 on 25 machines) when compared to using only 1 machine [18]. Furthermore, during the 2011 SAT Competition, the best parallel SAT solvers obtained a average speedup factor of about 3 on 32 cores, which was achieved by employing an algorithm portfolio rather than a parallelized search [20]. In our experimentation, the winner of the parallel category of the 2013 SAT Competition also achieved a speedup of only about 3 on 32 cores. Constraint programming search appears to be more suitable for parallelization than search for MIP or SAT: different strategies, including a recursive application of search goals [24], work stealing [14], problem decomposition [25], and a dedicated parallel scheme based on limited discrepancy search [23] all exhibit good speedups (sometimes near-linear) of the CP search in certain settings, especially those involving infeasible instances or scenarios where evaluating search tree leaves is costlier than evaluating internal nodes. Yet, recent developments in CP have moved towards more constraint learning during search, for which efficient parallelization becomes increasingly more difficult.

In general, search schemes relying heavily on learning during search (such as learning new bounds, activities for search heuristics, cuts for MIP, nogoods for CP, and clauses for SAT) tend to be more difficult to efficiently parallelize. *It remains a challenge to design a robust parallelization scheme for solving combinatorial optimization problems* which must necessarily deal with bounds.

Recently, a branch-and-bound scheme based on *approximate decision diagrams* was introduced as a promising alternative to conventional methods (such as integer programming) for solving combinatorial optimization problems [5, 7]. In this paper, our goal is to study how this branch-and-bound search scheme can be effectively parallelized. The key observation is that relaxed decision diagrams can be used to partition the search space, since for a given layer in the diagram each path from the root to the terminal passes through a node in that layer. We can therefore *branch on nodes in the decision diagram* instead of branching on variable-value pairs, as is done in conventional search methods. Each of the subproblems induced by a node in the diagram is processed recursively, and the process continues until all nodes have been solved by an exact decision diagram or pruned due to reasoning based on bounds on the objective function.

When designing parallel algorithms geared towards dozens or perhaps hundreds of workers operating in parallel, the two major challenges are *i)* balancing the workload across the workers, and *ii)* limiting the communication cost between workers. In the context of combinatorial search and optimization, most of the current methods are based on either parallelizing the traditional tree search or using portfolio techniques that make each worker operate on the entire problem. The former approach makes load balancing difficult as the computational cost of solving similarly sized subproblems can be orders of magnitude different. The latter approach typically relies on extensive communication in order to avoid duplication of effort across workers.

In contrast, using decision diagrams as a starting point for parallelization offers several notable advantages. For instance, the associated branch-and-bound method applies relaxed and restricted diagrams that are obtained by limiting the