Dagster: Parallel Structured Search

Mark Alexander Burgess, Charles Gretton, Josh Milthorpe, Luke Croak, Thomas Willingham, Alwen Tiu

School of Computing, Australian National University, Canberra, Australia, 2600
(mark.burgess, charles.gretton, josh.milthorpe, thomas.willingham, alwen.tiu)@anu.edu.au
markburgess1989@gmail.com, luke.croak1@defence.gov.au

Abstract

We demonstrate DAGSTER, a system that implements a new approach to scheduling interdependent (Boolean) SAT search activities in high-performance computing (HPC) environments. Our system takes as input a set of disjunctive clauses (i.e., DIMACS CNF) and a labelled directed acyclic graph (DAG) structure describing how the clauses are decomposed into a set of interrelated problems. Component problems are solved using standard systematic backtracking search, which may optionally be coupled to (stochastic dynamic) local search and/or clause-strengthening processes. We demonstrate DAGSTER using a new *Graph Maximal Determinant* combinatorial case study. This demonstration paper presents a new case study, and is adjunct to the longer accepted manuscript at the Pacific Rim International Conference on Artificial Intelligence (2022).

Introduction & Related Work

A range of frameworks have been developed for solving SAT problems in parallel. Portfolio approaches run multiple algorithms on the same formula, often in parallel, and typically have a mechanism for sharing learnt clauses. For instance: PLINGELING (Biere 2010), HORDESAT and its cloud-ready counterpart (Balyo, Sanders, and Sinz 2015; Schreiber and Sanders 2021), the multi-core SYRUP and its hybrid counterpart D-SYRUP (Audemard et al. 2017).

Cube-and-conquer approaches do not work on the same formula in parallel, but break the search space into disjoint subspaces by resolving with an array of partial assignments. More generally, decompose and conquer procedures include: DPLL-TD (Habet, Paris, and Terrioux 2009), PMSAT(Gil, Flores, and Silveira 2009), TLINGELING(Biere 2016), and PARACOOBA (Heisinger, Fleury, and Biere 2020), DMC (Lagniez, Marquis, and Szczepanski 2018), PAINLESS (Frioux et al. 2017).

We demonstrate DAGSTER, a new tool that takes as input a set of disjunctive clauses (e.g., a DIMACS CNF file), and a graphical structure which together with those clauses defines the problem at hand. The CNF gives the available constraints/clauses, and the graphical structure is used to represent how those constraints are grouped into interdependent subproblems, which are solved in parallel using DAGSTER. The solution to the underlying problem is derived via the combination of subproblem solutions. In this way our tool blends and extends both portfolio and compositional techniques. Additionally, DAGSTER search tasks can be made to enumerate satisfying assignments; thus, the tool directly supports solving both decision (SAT) problems and counting (#SAT) problems.

System Description

DAGSTER[1] takes as input a CNF formula and a labelled graphical structure as a directed acyclic graph (DAG). Each node of the DAG represents a subset of the clauses of the CNF, and finding solutions to these subsets of clauses constitutes a subproblem (see Figure 1). Nodes are connected by edges labelled with a subset of shared variables present in the clauses of both nodes. The directed nature of the graph means that the variable valuations in satisfying solutions found for each node are passed along the edges to constrain the solutions of subsequent node(s). Given an edge between nodes, each satisfying assignment at the destination node must be consistent with the shared variable valuations passed from the source nodes.

DAGSTER uses a Master-Worker architecture, coded in C++ using MPI (Message Passing Interface Forum 2015). The master issues and processes the work to and from the workers, with each worker solving the subproblems using a CDCL procedure. DAGSTER workers currently support: *(i)* the lightweight TINISAT (Huang 2007a,b), and *(ii)* MINISAT (Eén and Sörensson 2003) CDCL procedures. The worker reports each unique satisfying assignment it finds to the master. By default this process continues until the worker finds all satisfying assignments. The master uses the shared variable assignments from each subproblem solution to seed computation on further nodes in the DAG.

We designed DAGSTER to be modular and agnostic about the underlying CDCL solver. However, along with a CDCL solver, each worker may also be configured to collaborate with one or more helper processes, to form a worker group. Helper processes may include: a *strengthener* performing concurrent clause strengthening (Wieringa and Heljanko 2013) to accelerate search; and/or many *stochastic lo-*

[1]Sourcecode: Zenodo(Burgess 2022), and GitHub, https://github.com/ANU-HPC/dagster

cal search (SLS) processes, which use the GNOVELTY+ dynamic local search algorithm (Pham et al. 2008).

Case Study: Graph Maximal Determinant

We consider the model counting problem of finding all the undirected graphs, of size n edges, whose adjacency matrices have the maximum determinant. This is a constrained case of Hadamard's maximal determinant problem.[2] To encode our problem effectively as a #SAT problem, we considered calculating a similar class of adjacency matrices whose determinants could not be increased by changing any one of their elements. This class of matrices can be efficiently and succinctly encoded as a #SAT problem. Once computed, the matrices belonging to this set can be subsequently refined to efficiently find those matrices with maximum determinant.

The encoding of this set is efficient in #SAT, particularly via the Matrix Determinant Lemma, which renders the required condition as a relationship between any change in the elements of a matrix and the elements of its inverse. If a matrix element $A_{i,j}$ changes from 0 to 1, its determinant magnitude will increase by a factor $(A^{-1})_{i,j}$ as:

$$\det(A + uv^T) = (1 + v^T A^{-1} u)\det(A).$$

Here, u is the elementary unit vector e_j, and v the vector e_i. Conversely, if element $A_{i,j}$ changes from 1 to 0, then the determinant magnitude will decrease by the same factor.

Because of this relationship, the primary constraint in the #SAT formulation is the matrix integer inverse relation, $AB = aI$, where a is a positive integer. This constraint implies that B will then be some integer multiple of the inverse A^{-1}, and thus the sign relationship between the elements of the adjacency matrix A, and its scaled inverse B, determine the criterion for belonging to the targeted superset.

This #SAT problem is encoded using vertex symmetry breaking constraints (Codish et al. 2013). It is decomposed for DAGSTER according to a DAG of two parts (as illustrated in Figure 2). Here, the task of enumerating all possible adjacency matrices belonging to the upper $n-2$ submatrices is conducted, yielding partial solutions. Subsequently, for each partial solution a process finds all completions satisfying the full set of problem constraints. Thus, model counting is thereby distributed to be performed in parallel.

[2]Unanswered problem: https://mathoverflow.net/questions/386168/which-graphs-on-n-vertices-have-the-largest-determinant

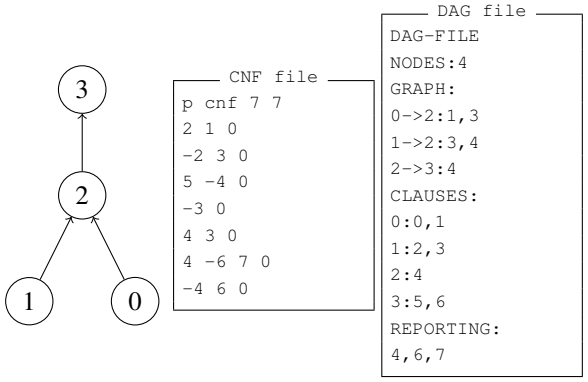

Figure 1: Example DAGSTER inputs. The graph (left) is described in the "DAG file", the "CLAUSES" block identifies subproblems by indexes of clauses in the "CNF file". The "GRAPH" block indicates what variables are shared between subproblems. "REPORTING" identifies variables of interest in resulting solution/s.

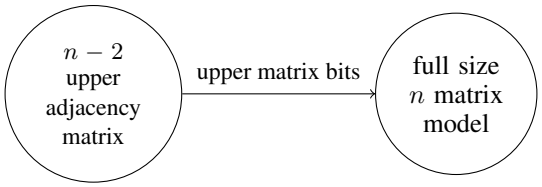

Figure 2: DAG structure for the determinant case study.

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Figure 3: An example of an $n = 5$ adjacency matrix with top $n-2$ upper submatrix highlighted.

Comparative runtime results are shown in Figure 4. The performance of parallel tools with 192 cores, including DAGSTER, is compared against *DMC* parallel model counter (Lagniez, Marquis, and Szczepanski 2018), *countAntom* parallel model counter (Burchard, Schubert, and Becker 2015), as well as serial model counters *sharpSAT* (Thurley 2006) and *GANAK* (Sharma et al. 2019).

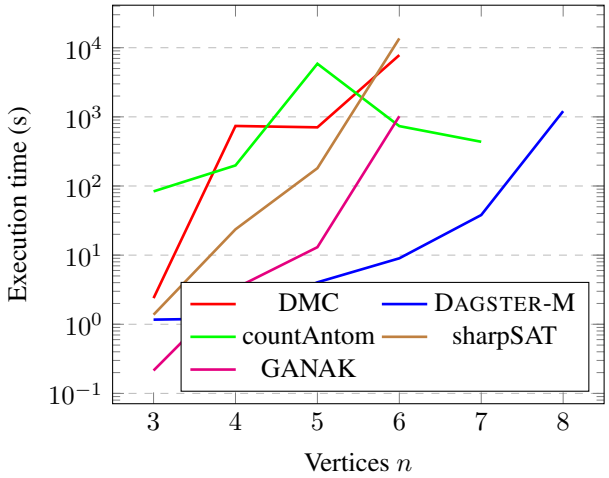

Figure 4: Runtime performance of model counting in case study problem of size n (vertices). Timeout is 4 hours.

References

Audemard, G.; Lagniez, J.; Szczepanski, N.; and Tabary, S. 2017. A Distributed Version of Syrup. In Gaspers, S.; and Walsh, T., eds., *Theory and Applications of Satisfiability Testing - SAT 2017*, volume 10491 of *Lecture Notes in Computer Science*, 215–232. Springer.

Balyo, T.; Sanders, P.; and Sinz, C. 2015. HordeSat: A Massively Parallel Portfolio SAT Solver. In *Theory and Applications of Satisfiability Testing - SAT 2015*.

Biere, A. 2010. Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. *FMV Technical Report 10/1, Johannes Kepler University, Linz, Austria*.

Biere, A. 2016. Splatz, Lingeling, PLingeling, Treengeling, YalSAT. In *Proceedings of the SAT Competition*, 44–45.

Burchard, J.; Schubert, T.; and Becker, B. 2015. Laissez-Faire Caching for Parallel #SAT Solving. In Heule, M.; and Weaver, S., eds., *Theory and Applications of Satisfiability Testing – SAT 2015*, 46–61. Cham: Springer International Publishing. ISBN 978-3-319-24318-4.

Burgess, M. A. 2022. Dagster - Parallel Structured Search - Source Code (DOI:10.5281/zenodo.7016083).

Codish, M.; Miller, A.; Prosser, P.; and Stuckey, P. J. 2013. Breaking Symmetries in Graph Representation. In Rossi, F., ed., *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 510–516. IJCAI/AAAI.

Eén, N.; and Sörensson, N. 2003. An extensible SAT-solver. In *International conference on theory and applications of satisfiability testing*, 502–518. Springer.

Frioux, L. L.; Baarir, S.; Sopena, J.; and Kordon, F. 2017. PaInleSS: A Framework for Parallel SAT Solving. In *Theory and Applications of Satisfiability Testing - SAT 2017*.

Gil, L.; Flores, P. F.; and Silveira, L. M. 2009. PMSat: a parallel version of MiniSAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 6(1-3): 71–98.

Habet, D.; Paris, L.; and Terrioux, C. 2009. A Tree Decomposition Based Approach to Solve Structured SAT Instances. In *2009 21st IEEE International Conference on Tools with Artificial Intelligence*, 115–122. IEEE Computer Society. ISBN 978-0-7695-3920-1.

Heisinger, M.; Fleury, M.; and Biere, A. 2020. Distributed Cube and Conquer with Paracooba. In Pulina, L.; and Seidl, M., eds., *Theory and Applications of Satisfiability Testing - SAT 2020*, volume 12178 of *Lecture Notes in Computer Science*, 114–122. Springer.

Huang, J. 2007a. A case for simple SAT solvers. In *International Conference on Principles and Practice of Constraint Programming*, 839–846. Springer.

Huang, J. 2007b. The Effect of Restarts on the Efficiency of Clause Learning. In *20th International Joint Conference on Artificial Intelligence (IJCAI)*, 2318–2323.

Lagniez, J.; Marquis, P.; and Szczepanski, N. 2018. DMC: A Distributed Model Counter. In Lang, J., ed., *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, 1331–1338. ijcai.org.

Message Passing Interface Forum. 2015. *MPI: A Message-Passing Interface Standard Version 3.1*.

Pham, D. N.; Thornton, J.; Gretton, C.; and Sattar, A. 2008. Combining Adaptive and Dynamic Local Search for Satisfiability. *Journal on Satisfiability, Boolean Modeling and Computation*, 4(2-4): 149–172.

Schreiber, D.; and Sanders, P. 2021. Scalable SAT Solving in the Cloud. In *Theory and Applications of Satisfiability Testing - SAT 2021*.

Sharma, S.; Roy, S.; Soos, M.; and Meel, K. S. 2019. GANAK: A Scalable Probabilistic Exact Model Counter. In Kraus, S., ed., *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 1169–1176. ijcai.org.

Thurley, M. 2006. sharpSAT - Counting Models with Advanced Component Caching and Implicit BCP. In Biere, A.; and Gomes, C. P., eds., *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, 424–429. Springer.

Wieringa, S.; and Heljanko, K. 2013. Concurrent Clause Strengthening. In *Theory and Applications of Satisfiability Testing - SAT 2013*.