Dagster: Parallel Structured Search


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 Terriou 2009), PM-SAT(Gil, Flores, and Silveira 2009), TLINGELING(Biere 2016), and PARACOBA (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.

System Description

DAGSTER 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 node.

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 #TINSAT (Huang 2007a, b), and (ii) MINILSAT (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 ...

1 Sourcecode: Zenodo(Burgess 2022), and GitHub, https://github.com/ANU-HPC/dagster
(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 \((1 + v^T A^{-1} u)\) as:

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

Here, \( u \) is the elementary unit vector \( e_i \), and \( v \) the vector \( e_j \). 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.

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).

---

\(^2\)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.

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

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


