Lazy Decomposition for Distributed Decision Procedures

Youssef Hamadi
Microsoft Research
Cambridge, UK
youssefh@microsoft.com

Joao Marques-Silva
University College
Dublin, IE
jpms@ucd.ie

Christoph M. Wintersteiger
Microsoft Research
Cambridge, UK
cwinter@microsoft.com

The increasing popularity of automated tools for software and hardware verification puts ever increasing demands on the underlying decision procedures. This paper presents a framework for distributed decision procedures (for first-order problems) based on Craig interpolation. Formulas are distributed in a lazy fashion, i.e., without the use of costly decomposition algorithms. Potential models which are shown to be incorrect are reconciled through the use of Craig interpolants. Experimental results on challenging propositional satisfiability problems indicate that our method is able to outperform traditional solving techniques even without the use of additional resources.

1 Introduction

Decision procedures for first-order logic problems, or fragments thereof, have seen a tremendous increase in popularity in the recent past. This is due to the great increase in performance of solvers for the propositional satisfiability (SAT) problem, as well as the increasing popularity of verification tools for both soft- and hardware which extensively use first-order decision procedures like SAT and SMT solvers.

As the decision problems that occur in large-scale verification problems become larger, methods for distribution and parallelization are required to overcome memory and runtime limitations of modern computer systems. Frequently, computing clusters and multi-core processors are employed to solve such problems. The inherent parallelism in these systems is often used to solve multiple problems concurrently, while distributed and parallel decision procedures would allow for much better performance. This has led to the development of distributed verification tools, for example through the distribution of Bounded-Model-Checking (BMC) problems (see, e.g., [9, 15]).

In the following sections we present a general method for distributed decision procedures which is applicable to decision procedures of first-order fragments. The key component in this method is Craig's interpolation theorem [11]. This theorem enables us to arbitrarily split formulas into multiple parts without any restrictions on the nature or size of the cut. We propose to use this lazy formula decomposition because it does not require analysis of the semantics of a formula prior to the distribution of the problem. In many other distributed algorithms, such a lazy decomposition clearly has a negative impact on the overall runtime of the decision procedure. However, when using an interpolation scheme, the abstraction provided by the interpolation algorithm is often strong enough to counterbalance this.

Through an experimental evaluation of our algorithm on propositional formulas, we are able to show that, at least in the propositional case, large speed-ups result from using a lazy decomposition when using a suitable interpolation algorithm.

2 Background

We are interested in satisfiability of first-order formulas. Formulas are assumed to be in conjunctive normal form, i.e., of the form $\phi = \psi_1(x_1, \dots, x_n) \wedge \dots \wedge \psi_m(x_1, \dots x_n)$, where $V_\phi = \{x_1, \dots, x_n\}$ are the free variables of ϕ and the ψ_i are clauses (disjunctions of atoms).

Craig's interpolation theorem provides a way to characterize the relationship between two formulas when one implies the other:

Theorem 1 (Craig Interpolation [11])**.** *Let ϕ and ψ be first-order formulas. If $\phi \Rightarrow \psi$ then there exists an* Interpolant *I such that $\phi \Rightarrow I \wedge I \Rightarrow \psi$ and $V_I \subseteq V_\phi \cap V_\psi$.*

Equivalently, there is an interpolant I such that $\phi \Rightarrow I \wedge I \Rightarrow \neg\psi$ whenever $\phi \wedge \psi$ is unsatisfiable, because $\phi \Rightarrow \neg\psi \equiv \neg(\phi \wedge \psi)$. Craig's theorem guarantees the existence of an interpolant, but does not provide an algorithm for obtaining it. However, such algorithms are known for many logics. We refer to two interpolation algorithms for propositional logic and to describe them, we require some definitions:

A literal l is either a propositional variable x or its negation $\neg x$. A clause is a disjunction of literals, denoted $\{l_1, \ldots, l_n\}$. A formula ϕ is assumed to be in conjunctive normal form (CNF), i.e., it is a conjunction of clauses, denoted $\phi = \{c_1, \ldots, c_n\}$.

A (partial) assignment α is a consistent set of clauses of size 1. The (propositional) SAT problem is to determine whether for a given formula ϕ there exists a total assignment such that $\phi \wedge \alpha \equiv \top$. Given two clauses of the form $C_1 \cup \{x\}$ and $C_2 \cup \{\neg x\}$, their resolution is defined as $C_1 \cup C_2$ and if it is not tautological the result is called a *resolvent* and the variable x is called the pivot variable. A resolution refutation is a sequence of resolution operations such that the final resolvent is empty, proving that the formula is unsatisfiable.

There are multiple techniques for propositional interpolation. Here, we refer to two popular systems, one by McMillan [26] and the other by Huang, Krajícek and Pudlák [20,24,30] (HKP). Both are methods that require time linear in the size of the resolution refutation of $\neg(\phi \wedge \psi)$. Both interpolation methods construct interpolants by associating an intermediate interpolant with every resolvent in the resolution refutation of $\phi \wedge \psi$. The interpolant associated with the final resolvent (the empty clause) constitutes an interpolant I for which Theorem 1 holds. For a characterization of these and other interpolation systems see, e.g., [13].

McMillan's interpolation system associates with every clause C in ϕ the intermediate interpolant $C \setminus \{v, \neg v | v \in V_\psi\}$, i.e., the restriction of C to the variables in ψ. Every clause in ψ is associated with the intermediate interpolant $\top$. Every other interpolant is calculated depending on the corresponding resolution step. Consider the derivation of resolvent R from clauses C_1 and C_2 with pivot variable x, where C_1 and C_2 have previously been associated with intermediate interpolants I_{C_1} and I_{C_2}. The resulting clause R is then associated with the intermediate interpolant

$$I_R := \begin{cases} I_{C_1} \vee I_{C_2} & \text{if } x \in V_\phi \wedge x \notin V_\psi \\ I_{C_1} \wedge I_{C_2} & \text{if } x \in V_\phi \wedge x \in V_\psi \\ I_{C_1} \wedge I_{C_2} & \text{if } x \notin V_\phi \wedge x \in V_\psi \end{cases} .$$

In the HKP system, every clause in ϕ is associated the intermediate interpolant $\perp$, while the clauses in ψ are associated $\top$. Every resolvent R obtained from clauses C_1 and C_2 with pivot variable x is associated with the intermediate interpolant

$$I_R := \begin{cases} I_{C_1} \vee I_{C_2} & \text{if } x \in V_\phi \wedge x \notin V_\psi \\ (x \vee I_{C_1}) \wedge (\neg x \vee I_{C_2}) & \text{if } x \in V_\phi \wedge x \in V_\psi \\ I_{C_1} \wedge I_{C_2} & \text{if } x \notin V_\phi \wedge x \in V_\psi \end{cases} .$$

For every propositional interpolation system that computes an interpolant for $\phi \Rightarrow \psi$, the *dual* system is defined by the computation of an interpolant for I for $\psi \Rightarrow \phi$, with the effect that $\neg I$ is an interpolant for $\phi \Rightarrow \psi$. It is known that the HKP system is self-dual and that McMillan's system is not [13].

3 Related Work

Our work is most closely related to parallel and distributed decision procedures. Many decision procedures exists for the propositional satisfiability and some of them exploit parallelism.

3.1 Parallel SAT Solving

In parallel SAT, the objective is to simultaneously explore different parts of the search space in order to quickly solve a problem. There are two main approaches to parallel SAT solving. First, the classical concept of divide-and-conquer, which divides the search space into subspaces and allocate each of them to sequential SAT solvers. The search space is divided thanks to guiding-path constraints (typically unit clauses). A formula is found satisfiable if one worker is able to find a solution for its subspace, and unsatisfiable if all the subspaces have been proved unsatisfiable. Workers usually cooperate through a load balancing strategy which performs the dynamic transfer of subspaces to idle workers, and through the exchange of conflict-clauses [10, 16].

In 2008, Hamadi et al. [18, 19] introduced the parallel portfolio approach. This method exploits the complementarity between different sequential DPLL strategies to let them compete and cooperate on the original formula. Since each worker deals with the whole formula, there is no need for load balancing, and the cooperation is only achieved through the exchange of learnt clauses. Moreover, the search process is not artificially influenced by the original set of guiding-path constraints like in the first category of techniques. With this approach, the crafting of the strategies is important, and the objective is to cover the space of the search strategies in the best possible way.

The main drawback of parallel SAT techniques comes from their required replication of the formula. This is obvious for the parallel portfolio approach. It is also true for divide-and-conquer algorithms whose guiding-path constraints do not produce significantly smaller subproblems (only $log_2 c$ variables have to be set to obtain c subproblems). This makes these techniques only applicable to problems which fit into the memory of a single machine.

In the last two years, portfolio-based parallel solvers became prominent and it has been used in SMT decision procedures as well [34]. We are not aware of a recently developed improvements on the divide-and-conquer approach (the latest being [16]). We give a brief description of the parallel solvers qualified for the 2010 SAT Race[1]:

- In `plingeling` [6], the original SAT instance is duplicated by a boss thread and allocated to worker threads. The strategies used by these workers are mainly differentiated around the amount of pre-processing, random seeds, and variables branching. Conflict clause sharing is restricted to units which are exchanged through the boss thread. This solver won the parallel track of the 2010 SAT Race.
- `ManySAT` [19] was the first parallel SAT portfolio. It duplicates the instance of the SAT problem to solve, and runs independent SAT solvers differentiated on their restart policies, branching heuristics, random seeds, conflict clause learning, etc. It exchanges clauses through various policies. Two versions of this solver were presented at the 2010 SAT Race, they finished second and third.
- In `SArTagnan`, [22] different SAT algorithms are allocated to different threads, and differentiated with respect to restart policies and VSIDS heuristics. Some threads apply a dynamic resolution process [5,6] or exploit reference points [23]. Some others try to simplify a shared clauses database by performing dynamic variable elimination or replacement. This solver finished fourth.

[1] `http://baldur.iti.uka.de/sat-race-2010`

- In `Antom` [32], the SAT algorithms are differentiated on decision heuristic, restart strategy, conflict clause detection, lazy hyper binary resolution [5, 6], and dynamic unit propagation lookahead. Conflict clause sharing is implemented. This solver finished fifth.

3.2 Distributed SAT Solving

Contrary to parallel SAT, in distributed SAT, the goal is to handle problems which are by nature distributed or, even more interestingly, to handle problems which are too large to fit into the memory of a single computing node. Therefore, the speed-up against a sequential execution is not necessarily the main objective, and in some cases (large instances) cannot even be measured.

To the best of our knowledge, the only relevant work in the area presents an architecture tailored for large distributed Bounded Model Checking [15]. The objective is to perform deep BMC unwindings thanks to a network of standard computers, where the SAT formulas become so large that they cannot be handled by any one of the machines. This approach uses a master/slaves topology, and the unrolling of an instance provides a natural partitioning of the problem in a chain of workers. Each worker has to reconcile its local solution with its neighbors. The master distributes the parts, and controls the search. First, based on proposals coming from the slaves, it selects a globally best variable to branch on. From that decision, each worker performs Boolean Constraint Propagation (BCP) on its subproblem, and the master performs the same on the globally learnt clauses. The master maintains the global assignment, and to ensure the consistency of the parallel BCP algorithms propagates to the slaves Boolean implications. The master also records the causality of these implications which allows him to perform conflict-analysis when required.

3.3 Interpolation

McMillan's propositional interpolation system [26] when employed in a suitable Model Checking algorithm, has been shown to perform competitively with algorithms based purely on SAT-solving, i.e., McMillan showed that the abstraction obtained through interpolation for Model Checking problems is at least as good as and sometimes better than previously known abstraction methods.

4 Lazy Decomposition

When considering distributed decision procedures, it is usually assumed that the formulas which are to be solved are too large to be solved on a single computing node. Under this premise, strategies for distributing a formula have to be employed. If there exists a quantifier elimination algorithm for the fragment considered, then it is straight-forward, but comparatively expensive to distribute the problem: Find sparsely connected partitions of the formula and eliminate the connections such that the partitions become independent. For example, let formula $\phi = \phi_1 \wedge \phi_2$ where the partitions ϕ_1 and ϕ_2 overlap on variables $X = V_{\phi_1} \cap V_{\phi_2}$. The elimination of X from $\exists X \,.\, \phi_1 \wedge \phi_2$ produces two independent parts ϕ_1' and ϕ_2', which, respectively, depend on variables $V_{\phi_1} \setminus X$ and $V_{\phi_2} \setminus X$ and therefore can be solved independently. While this distribution strategy is quite simple, it depends on the existence of a quantifier elimination algorithm. Furthermore, the performance of such an algorithm in practice depends greatly on the fact that the problem is sparsely connected, which is not generally a given. We therefore use a different and cheaper method for distribution:

Definition 1 (Lazy Decomposition)**.** *Let ϕ be in conjunctive normal form, i.e., $\phi = \phi_1 \wedge \ldots \wedge \phi_n$. A* lazy decomposition *of ϕ into k partitions is an equivalent set of formulas $\{\psi_1, \ldots, \psi_k\}$ such that each*

```
Input   : Formula $\phi$
Output: $\top$ if $\phi$ is satisfiable, $\bot$ otherwise
$\psi_1, \dots, \psi_k := decompose(\phi)$;
$G := \top$;
$flag :=$ true;
while $flag$ do
    if $G \equiv \bot$ then
        return $\bot$;
    else
        Let $m$ be a total model for $G$;
    end
    $flag :=$ false;
    foreach $i$ in $1 \dots k$ do
        if $\psi_i \wedge m \equiv \bot$ then
            Let $I$ be an interpolant for $\neg(\psi_i \wedge m)$ over $V_{\psi_i} \cap V_G$;
            $G := G \wedge I$;
            $flag :=$ true;
        end
    end
end
return $\top$;
```

Algorithm 1: A reconciliation algorithm.

ψ_i is equivalent to some conjunction of clauses from ϕ, i.e., there exist a, b $(a < b < n)$, such that $\psi_i = \phi_a \wedge \dots \wedge \phi_b$.

We call this a *lazy* decomposition, because no effort is made to ensure that partitions do not share variables. The formulas $\psi_1 \dots \psi_k$ may then be solved independently, but if the partitions happen to share variables, i.e., when $V_{\psi_i} \cap V_{\psi_j} \neq \emptyset$ for some $j \neq i$, then these (potentially global) solutions have to be reconciled.

Let $S_{\psi_i} = \bigvee_j m_{i,j}$ be the set of all models satisfying ψ_i and let $S = \{S_{\psi_1}, \dots, S_{\psi_k}\}$ be a set of all models of all partitions. The *reconciliation problem* is to determine whether $S_{\psi_1} \wedge \dots \wedge S_{\psi_k}$ is satisfiable, i.e., to determine whether there is a global model in S_ϕ which has a matching extension in each S_{ψ_i}. Clearly, any set of models is not required to be any smaller in representation than its corresponding partition; in fact, it may be exponentially larger. In practice it may therefore be more efficient to build the solution sets incrementally, avoiding any blowup wherever possible. To this end, we require the following lemma:

Lemma 1. *Let $\phi = \psi_1 \wedge \dots \wedge \psi_k$ and let m be a model for the shared variables $V := \bigcup_{i,j=1}^{k} V_{\psi_i} \cap V_{\psi_j}$ and let $1 \leq i \leq k$. If I is an interpolant for $\neg(\psi_i \wedge m)$ then $\phi \Rightarrow I$.*

Proof. I is an interpolant for $\neg(\psi_i \wedge m)$ or, equivalently, for $\psi_i \Rightarrow \neg m$. Therefore $\psi_i \Rightarrow I$. Since $\phi \Rightarrow \psi_i$, we also have $\phi \Rightarrow I$. □

Algorithm 1 presents a simple method that makes use of this Lemma to solve a decomposed formula. First, it extracts a model m for G, which is over the shared variables of the decomposition (the globals). It then attempts to extend the model to models satisfying each of the partitions and returns $\top$ if this was successful. Otherwise, it extracts an interpolant I from every unsatisfiable partition which is subsequently used to refine G. When G is found to be unsatisfiable the algorithm returns $\perp$ as there cannot be any model that is extensible to all partitions.

The maximum number of iterations require by Algorithm 1 depends on the number and the domain of the shared variables in the decomposition. Of course, this motivates the use of decomposition techniques that find partitions with little overlap and on the other hand, motivates the use of interpolation techniques that produce (logically) weak interpolants such that every interpolant covers as many (global) models as possible.

Theorem 2. *Algorithm 1 is sound.*

Proof. When the algorithm returns $\top$, there is a model m for G which has an extension in every ψ_i. Conversely, if the algorithm returns $\perp$, then every potential model m is contained in some interpolant which implies $\neg m$, which is an immediate consequence of Lemma 1. □

Theorem 3. *Algorithm 1 is complete for formulas over finite-domain variables.*

Proof. Every iteration of the algorithm excludes at least one possible model from G (otherwise the algorithm would terminate and return $\top$). For formulas over finite-domain variables there is only a finite number of potential variables. Therefore, G has to become unsatisfiable at some point, forcing $G \equiv \perp$ and therefore termination of the algorithm. □

5 Interpolation and Conflict Clauses

The DPLL procedure is an algorithm that solves the SAT problem (see, e.g., [28]). It does so by evaluating a series of partial assignments until a total assignment is found. When a partial assignment is found to be inconsistent with the input formula, DPLL backtracks to a previous (smaller) assignment. Modern incarnations of this algorithm use conflict-driven backtracking, which means that the conflicting state of the solver is analyzed and a *conflict clause* is derived. It is required that every conflict clause be implied by the original formula, that it is over the variables of the current assignment, and that it be inconsistent with the current assignment. Any conflict clause is therefore redundant, but it may help to prevent further conflicts when it is kept in the clause database (in which case it is called a *learnt* clause). We think it worthwile to characterize the relationship between conflict clauses and interpolants:

Corollary 1. *Every conflict clause for a propositional formula ϕ derived under the partial assignment α is an interpolant for $\phi \Rightarrow \neg\alpha$.*

Proof. According to the definition of a conflict clause C, it must be implied by ϕ and inconsistent with α. We therefore have $\phi \Rightarrow C$ and $\neg(C \wedge \alpha) \equiv C \Rightarrow \neg\alpha$, which makes C an interpolant for $\phi \Rightarrow \neg\alpha$ by Theorem 1. □

Currently, the most popular conflict resolution scheme for DPLL-style solvers is the so-called First-UIP method (for a definition see [7]). The corollary stated above raises the question whether other interpolation methods are able to improve upon this scheme. Note that the First-UIP scheme has some properties which make it very efficient in practice:

- a conflict clause can be computed in linear time and
- every such clause is *asserting*, i.e., it contains a unique literal which is unassigned after backtracking.

More general interpolation schemes like McMillan's or HKP also have the first of these properties since interpolants are usually computed in linear time from a resolution proof. Therefore, an interpolant may be computed in linear time, too, if the resolution proof of the current conflict is kept in the state of the solver. This, however, is much more expensive than keeping the reasons for implications as is done in the First-UIP scheme.

An interpolant is generally not of clause form. If it is to be kept as part of the problem (akin to a learnt clause) it therefore requires conversion. The straight-forward expansion to CNF may increase the size of the interpolant exponentially. The Tseitin transformation [33] increases the size of the interpolant only linearly, but introduces new variables. It is not clear which of these methods is to be preferred. In general, however, a *set* of conflict clauses is produced, instead of a single clause like in the First-UIP scheme.

An interpolant is also asserting in the sense that it is asserted to be true; however, it is not immediately asserting a specific literal like a First-UIP conflict clause. A preliminary experimental evaluation (of which the details are omitted) has shown that none of the known propositional interpolation methods performs better than the First-UIP scheme. This, however, may be due to the lack of an efficient interpolation algorithm that matches the performance of the algorithm for First-UIP conflict resolution.

6 Experimental Evaluation

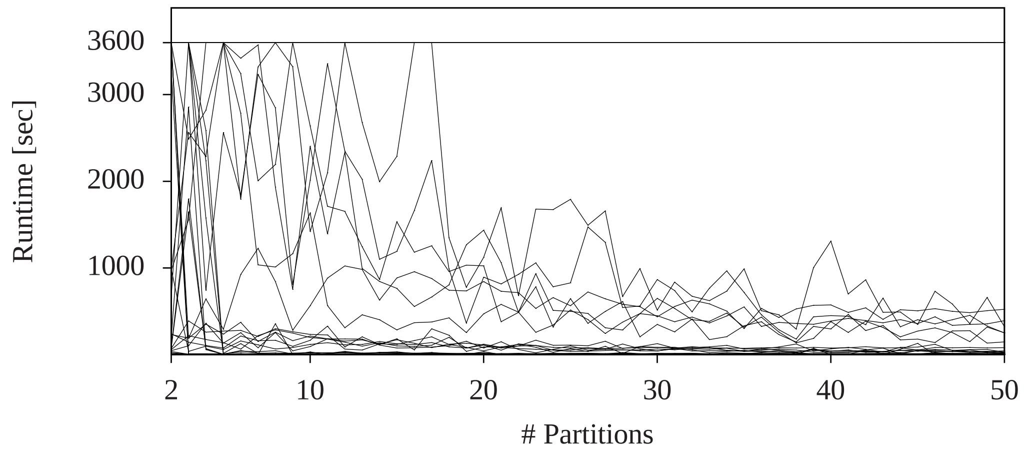
As a first step in evaluating our algorithm, we implemented a propositional satisfiability solver based on the MiniSAT solver. We restrict ourselves to the slightly outdated version 1.14p, because propositional interpolation methods require proof production, which is not available in more recent versions of MiniSAT [14]. Interpolants are produced by iterating over the resolution proof, which is saved (explicitly) in memory. We use Reduced Boolean Circuits (RBCs [1]) to represent interpolants such that recurring structure is exploited. Furthermore, Algorithm 1 permits the exploitation of state-of-the-art SAT solver technology, like incremental solving techniques in solving partitions. Furthermore, every assignment to the globals is a set of clauses of size 1, which means that facilities for solving a formula under assumptions may be made use of. The lazy decomposition used by our implementation is indeed quite trivial: it simply divides the clauses of the problem into a predefined number p of equally sized partitions. Clauses are ordered as they appear in the input file and each partition i is assigned the clauses numbered from $i \cdot \frac{n}{p}$ to $(i+1) \cdot \frac{n}{p}$, where n is the total number of clauses.

Our implementation is evaluated on set of formulas which are small but hard to solve [2][2]. They are known to contain symmetries, which potentially can be exploited by interpolation. For this evaluation, our implementation uses only a single processing element, i.e., the evaluation of the partitions of a decomposition is sequentialized. Through this, we are able to show that our algorithm performs well even when using the same resources as a traditional solver. Preliminary experiments have shown that an actual (shared-memory) parallelization of our algorithm performs better than the sequentialized version, but not significantly so, which is due to the lack of a load-balancing mechanism to balance the runtime of partition evaluation.

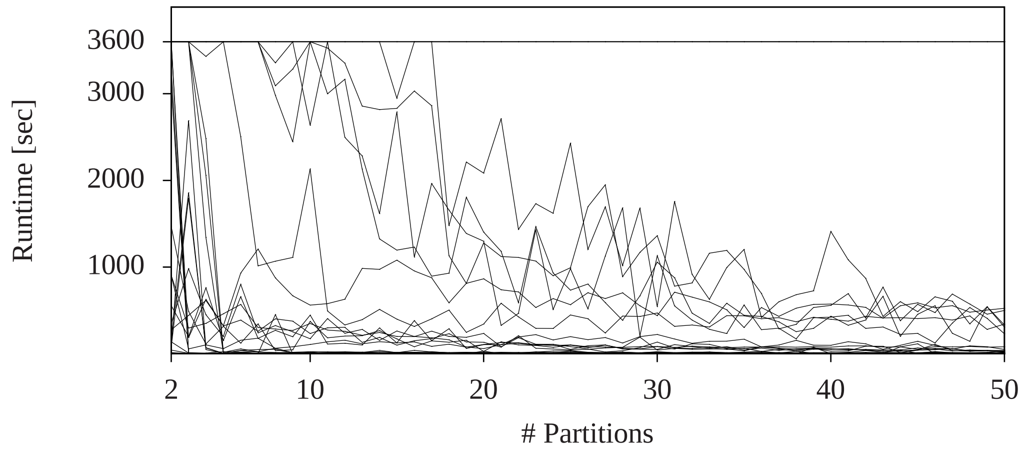
All our experiments are executed on a Windows HPC cluster of dual Quad-Xeon 2.5 GHz processors with 16 GB of memory each, using a timeout of 3600 seconds and a memory limit of 2 GB.

To assess the impact of the decomposition on the solver performance, we investigate all decompositions into 2 to 50 partitions for each of the three interpolation methods (McMillan's, Dual McMillan's

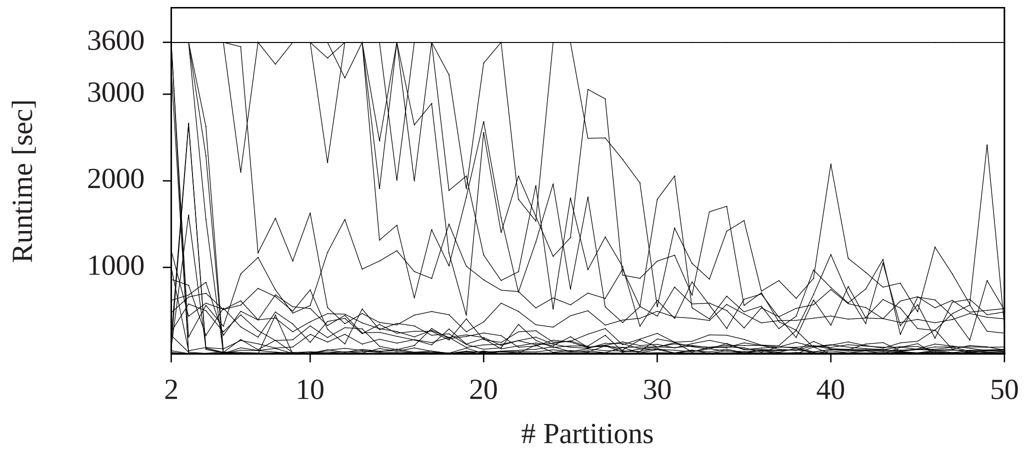
[2]`http://www.aloul.net/benchmarks.html`



(a) McMillan interpolants



(b) HKP interpolants



(c) Dual McMillan interpolants

Figure 1: Decomposition into $\{2,\ldots,50\}$ partitions, using different interpolation systems.

	Minisat		**Decomposition (# partitions)**						
Filename	**2.2.0**	**1.14p**	**5**	**10**	**20**	**30**	**40**	**50**	**Best**
chnl10_11.cnf	169.39	33.52	3.00	3.20	1.92	2.14	1.29	**1.22**	**0.98**
chnl10_12.cnf	165.02	20.95	2.46	2.07	4.17	2.07	**1.53**	2.15	**0.98**
chnl10_13.cnf	204.77	15.23	2.93	**1.87**	3.40	2.57	2.40	2.07	**0.89**
chnl11_12.cnf	T/O	291.61	**5.23**	6.05	17.02	14.29	6.46	7.16	**4.45**
chnl11_13.cnf	T/O	960.72	5.69	**4.70**	18.28	12.62	7.55	6.07	**4.45**
chnl11_20.cnf	T/O	2346.30	13.31	15.07	**5.05**	10.50	10.33	12.39	**4.96**
fpga10_8_sat.cnf	**0.00**	0.02	0.02	0.02	0.03	0.03	0.03	0.06	**0.00**
fpga10_9_sat.cnf	**0.02**	**0.02**	**0.02**	**0.02**	**0.02**	0.03	0.03	0.03	**0.02**
fpga12_8_sat.cnf	**0.00**	0.02	0.02	0.02	0.03	0.03	0.03	0.05	**0.00**
fpga12_9_sat.cnf	0.03	**0.02**	0.03	**0.02**	0.05	0.05	0.05	0.05	**0.02**
fpga12_11_sat.cnf	0.02	**0.00**	0.03	0.03	0.03	0.03	0.05	0.06	**0.00**
fpga12_12_sat.cnf	**0.00**	0.02	0.05	0.05	0.03	0.03	0.03	0.05	0.02
fpga13_9_sat.cnf	**0.00**	**0.00**	0.03	0.02	0.03	0.06	0.05	0.08	0.02
fpga13_10_sat.cnf	**0.00**	**0.00**	0.05	0.03	0.05	0.05	0.05	0.06	0.02
fpga13_12_sat.cnf	**0.00**	**0.00**	0.06	0.08	0.11	0.09	0.12	0.12	0.02
hole7.cnf	0.08	**0.05**	0.08	0.11	0.08	0.08	0.11	0.11	0.06
hole8.cnf	0.37	0.33	**0.31**	0.51	0.37	0.53	0.37	0.47	**0.30**
hole9.cnf	6.51	3.59	1.56	3.03	2.31	1.93	**1.40**	1.65	**1.22**
hole10.cnf	247.11	26.75	**10.09**	22.76	11.22	12.93	12.50	10.65	**4.34**
hole11.cnf	T/O	509.86	**59.12**	116.42	109.61	81.88	75.93	80.84	**37.03**
hole12.cnf	T/O	T/O	**293.08**	564.38	843.20	435.18	572.90	522.65	**187.70**
s3-3-3-1.cnf	0.44	**0.16**	2.98	0.30	0.20	0.23	0.27	0.39	**0.16**
s3-3-3-3.cnf	1.20	**0.02**	1.90	0.33	0.23	0.28	0.50	0.39	0.12
s3-3-3-4.cnf	0.44	1.50	3.51	0.23	**0.14**	0.19	0.42	0.51	**0.12**
s3-3-3-8.cnf	0.34	0.80	0.98	0.37	**0.20**	0.48	0.56	0.42	**0.19**
s3-3-3-10.cnf	0.47	0.81	1.23	0.41	**0.36**	0.48	0.67	0.55	**0.28**
Urq3_5.cnf	310.22	**58.27**	T/O	1633.50	470.33	448.89	386.04	391.56	243.55
Urq4_5.cnf	T/O	T/O	T/O	T/O	T/O	T/O	T/O	T/O	T/O
Urq5_5.cnf	T/O	T/O	T/O	T/O	T/O	T/O	T/O	T/O	T/O
Urq6_5.cnf	T/O	T/O	T/O	T/O	T/O	T/O	T/O	T/O	T/O
Urq7_5.cnf	T/O	T/O	M/O	T/O	T/O	T/O	T/O	T/O	T/O
Urq8_5.cnf	T/O	T/O	T/O	T/O	T/O	T/O	T/O	T/O	T/O
fpga10_8_sat_rcr.cnf	**0.00**	**0.00**	0.03	0.03	0.03	0.03	0.05	0.05	0.02
fpga10_9_sat_rcr.cnf	**0.00**	**0.00**	0.02	0.05	0.03	0.05	0.06	0.08	**0.00**
fpga12_8_sat_rcr.cnf	**0.00**	0.02	0.03	0.02	0.08	0.03	0.09	0.06	0.02
fpga12_9_sat_rcr.cnf	**0.02**	**0.02**	**0.02**	0.03	0.05	0.05	0.05	0.08	**0.02**
fpga12_10_sat_rcr.cnf	0.02	**0.00**	0.03	0.05	0.05	0.08	0.09	0.08	0.02
fpga12_11_sat_rcr.cnf	**0.02**	0.03	**0.02**	0.06	0.06	0.09	0.08	0.09	**0.02**
fpga12_12_sat_rcr.cnf	0.02	**0.00**	0.05	0.50	0.42	0.09	0.14	0.19	0.02
fpga13_9_sat_rcr.cnf	**0.00**	0.02	0.03	0.03	0.05	0.05	0.06	0.09	0.02
fpga13_10_sat_rcr.cnf	**0.03**	0.05	**0.03**	0.05	0.14	0.08	0.08	0.14	**0.02**
fpga13_11_sat_rcr.cnf	**0.00**	**0.00**	0.05	0.03	0.06	0.09	0.12	0.11	0.03
fpga13_12_sat_rcr.cnf	**0.00**	**0.00**	0.08	30.83	34.30	0.12	0.25	0.12	0.02
fpga10_11_uns_rcr.cnf	173.04	32.09	235.20	213.47	110.25	41.68	41.23	**29.39**	**26.97**
fpga10_12_uns_rcr.cnf	484.76	109.31	137.37	197.23	120.71	41.65	25.79	**13.81**	**13.81**
fpga10_13_uns_rcr.cnf	1279.94	110.32	132.26	92.70	74.96	65.26	**38.77**	42.46	**20.12**
fpga10_15_uns_rcr.cnf	T/O	215.50	271.72	232.78	42.85	83.32	26.13	**20.64**	**8.25**
fpga10_20_uns_rcr.cnf	3089.90	696.29	76.05	164.60	87.72	125.91	65.74	**28.24**	**28.24**
fpga11_12_uns_rcr.cnf	3204.84	449.22	T/O	2404.97	1435.07	648.40	380.16	**251.60**	**134.86**
fpga11_13_uns_rcr.cnf	T/O	1933.98	T/O	2638.34	1023.83	348.48	295.11	**146.20**	**132.34**
fpga11_14_uns_rcr.cnf	T/O	3199.81	2562.86	2011.51	893.40	866.23	450.22	**250.13**	**179.15**
fpga11_15_uns_rcr.cnf	T/O	T/O	T/O	1417.94	1121.73	512.84	1309.82	**321.85**	**294.67**
fpga11_20_uns_rcr.cnf	T/O	T/O	T/O	2828.49	2141.49	1068.31	577.14	**352.92**	**154.24**

Table 1: Runtime comparison with MiniSAT (versions 2.2.0 and 1.14p), using McMillan's interpolation system. Bold numbers indicate the smallest runtime in the decompositions shown here or the best decomposition into $\{2,\ldots,50\}$ partitions (right-most column).

and HKP). Each line in Figures 1a, 1b, and 1c corresponds to one benchmark file and indicates the change in runtime required to solve the benchmark when using from 2 up to 50 partitions in the decomposition. These figures provide strong evidence for an improvement of the runtime behavior as the number of partitions increase. Note that, as mentioned before, the evaluation of partitions was sequentialized for this experiment, i.e., this effect is *not* due to an increasing number of resources being utilized. The graphs in Figures 1a, 1b, and 1c provide equally strong evidence for the utility of McMillan's interpolation system: the impact on the runtime is the largest and most consistent of all three interpolation systems.

Finally, Table 1 provides a comparison of the runtime of MiniSAT versions 2.2.0 and 1.14p with a selection of different decompositions, the right-most column indicating the time of the best decomposition found among all those evaluated. It is clear from this table that no single partitioning can be identified as the best overall method. However, some decompositions, like the one into 50 partitions, perform consistently well and almost always better than either versions of MiniSAT.

7 Conclusion

We present the concept of lazy distribution for first-order decision procedures. Formulas are decomposed into partitions without the need for quantifier elimination or any other method for logical disconnection of the partitions. Instead, local models for the partitions are reconciled globally through the use of Craig interpolation. Experiments using different interpolation systems and decompositions for propositional formulas indicate that our approach performs better than traditional solving methods even when sequentialized, i.e., when no additional resources are used. At the same time, our algorithm provides straight-forward opportunities for parallelization and distribution of the solving process.

References

[1] Parosh Aziz Abdulla, Per Bjesse & Niklas Eén (2000): *Symbolic Reachability Analysis Based on SAT-Solvers*. In: *Proc. of TACAS*, *LNCS* 1785, Springer, pp. 411–425, doi:10.1007/3-540-46419-0_28.

[2] Fadi A. Aloul, Arathi Ramani, Igor L. Markov & Karem A. Sakallah (2002): *Solving difficult SAT instances in the presence of symmetry*. In: *Proc. of DAC*, ACM, pp. 731–736, doi:10.1145/513918.514102.

[3] Bowen Alpern, Roger Hoover, Barry K. Rosen, Peter F. Sweeney & F. Kenneth Zadeck (1990): *Incremental Evaluation of Computational Circuits*. In: *Proc. of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, ACM, pp. 32–42.

[4] Bengt Aspvall, Michael F. Plass & Robert Endre Tarjan (1979): *A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas*. *Inf. Process. Lett.* 8(3), pp. 121–123, doi:10.1016/0020-0190(79)90002-4.

[5] Armin Biere (2009): *Lazy hyper binary resolution*. Technical Report, Dagstuhl Seminar 09461.

[6] Armin Biere (2010): *Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010*. Technical Report 10/1, FMV Reports Series.

[7] Armin Biere, Marijn Heule, Hans van Maaren & Toby Walsh, editors (2009): *Handbook of Satisfiability*. *Frontiers in Artificial Intelligence and Applications* 185, IOS Press.

[8] Per Bjesse, James H. Kukula, Robert F. Damiano, Ted Stanion & Yunshan Zhu (2004): *Guiding SAT Diagnosis with Tree Decompositions*. In: *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003, Selected Revised Papers*, *Lecture Notes in Computer Science* 2919, Springer, pp. 315–329, doi:10.1007/978-3-540-24605-3_24.

[9] S. Campos, J. Neves, L. Zarate & M. Song (2009): *Distributed BMC: A Depth-First Approach to Explore Clause Symmetry*. In: *Proc. of the Intl. Conf. and Workshop on the Engineering of Computer Based Systems (ECBS 2009)*, IEEE Press, pp. 89–94, doi:10.1109/ECBS.2009.26.

[10] Wahid Chrabakh & Rich Wolski (2003): *GrADSAT: A Parallel SAT Solver for the Grid*. Technical Report, UCSB Computer Science.

[11] William Craig (1957): *Linear Reasoning. A New Form of the Herbrand-Gentzen Theorem*. *J. Symb. Log.* 22(3), pp. 250–268, doi:10.2307/2963593.

[12] Martin Davis, George Logemann & Donald Loveland (1962): *A machine program for theorem-proving*. *Commun. ACM* 5, pp. 394–397, doi:10.1145/368273.368557.

[13] Vijay D'Silva (2010): *Propositional Interpolation and Abstract Interpretation*. In: *Proc. of ESOP*, *LNCS* 6012, Springer, pp. 185–204, doi:10.1007/978-3-642-11957-6_11.

[14] Niklas Eén & Niklas Sörensson (2003): *An Extensible SAT-solver*. In: *Proc. of SAT*, *LNCS* 2919, Springer, pp. 502–518, doi:10.1007/978-3-540-24605-3_37.

[15] Malay K. Ganai, Aarti Gupta, Zijiang Yang & Pranav Ashar (2006): *Efficient distributed SAT and SAT-based distributed Bounded Model Checking*. *Intl. J. on Software Tools for Technology Transfer (STTT)* 8(4-5), pp. 387–396, doi:10.1007/s10009-005-0203-z.

[16] Chu Gorey & Peter J. Stuckey (2008): *PMiniSAT: a parallelization of MiniSAT 2.0*. Technical Report, SAT Race.

[17] Bernhard Haeupler, Telikepalli Kavitha, Rogers Mathew, Siddhartha Sen & Robert Endre Tarjan (2008): *Faster Algorithms for Incremental Topological Ordering*. In: *Proc. of ICALP*, *LNCS* 5125, Springer, pp. 421–433, doi:10.1007/978-3-540-70575-8_35.

[18] Youssef Hamadi, Said Jabbour & Lakhdar Sais (2008): *ManySAT: Solver Description*. *Microsoft Research Technical Report MSR-TR-2008-83* .

[19] Youssef Hamadi, Said Jabbour & Lakhdar Sais (2009): *ManySAT: a Parallel SAT Solver*. *J. on Satisfiability, Boolean Modeling and Computation* 6, pp. 245–262.

[20] Guoxiang Huang (1995): *Constructing Craig Interpolation Formulas*. In: *Proc. of the Intl. Conf. on Computing and Combinatorics (COCOON)*, *LNCS* 959, Springer, pp. 181–190.

[21] Antti Eero Johannes Hyvärinen, Tommi A. Junttila & Ilkka Niemelä (2010): *Partitioning SAT Instances for Distributed Solving*. In: *Proc. of LPAR*, *LNCS* 6397, Springer, pp. 372–386, doi:10.1007/978-3-642-16242-8_27.

[22] Stephan Kottler (2010): *SArTagnan: Solver Description*. Technical Report, SAT Race 2010.

[23] Stephan Kottler (2010): *SAT Solving with Reference Points*. In: *Proc. of SAT*, *LNCS* 6175, Springer, pp. 143–157, doi:10.1007/978-3-642-14186-7_13.

[24] Jan Krajícek (1997): *Interpolation Theorems, Lower Bounds for Proof Systems, and Independence Results for Bounded Arithmetic*. *J. Symb. Log.* 62(2), pp. 457–486, doi:10.2307/2275541.

[25] Alberto Marchetti-Spaccamela, Umberto Nanni & Hans Rohnert (1996): *Maintaining a Topological Order Under Edge Insertions*. *Inf. Process. Lett.* 59(1), pp. 53–58, doi:10.1016/0020-0190(96)00075-0.

[26] Kenneth L. McMillan (2003): *Interpolation and SAT-Based Model Checking*. In: *Proc. of CAV*, *LNCS* 2725, Springer, pp. 1–13.

[27] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang & Sharad Malik (2001): *Chaff: Engineering an Efficient SAT Solver*. In: *Proc. of DAC*, ACM, pp. 530–535.

[28] Robert Nieuwenhuis, Albert Oliveras & Cesare Tinelli (2006): *Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T)*. *J. ACM* 53(6), pp. 937–977, doi:10.1145/1217856.1217859.

[29] David J. Pearce & Paul H. J. Kelly (2006): *A dynamic topological sort algorithm for directed acyclic graphs*. *ACM J. of Experimental Algorithmics* 11, doi:10.1145/1187436.1210590.

[30] Pavel Pudlák (1997): *Lower Bounds for Resolution and Cutting Plane Proofs and Monotone Computations*. *J. Symb. Log.* 62(3), pp. 981–998, doi:10.2307/2275583.

[31] Liam Roditty & Uri Zwick (2008): *Improved Dynamic Reachability Algorithms for Directed Graphs*. *SIAM J. Comput.* 37(5), pp. 1455–1471, doi:10.1137/060650271.

[32] Tobias Schubert, Matthew Lewis & Bernd Becker (2010): *Antom: Solver Description*. Technical Report, SAT Race.

[33] G.S. Tseitin (1968): *On the complexity of derivation in propositional calculus*. In A.O. Slisenko, editor: *Structures in Constructive Mathematics and Mathematical Logic, Part II, Seminars in Mathematics (translated from Russian)*, Steklov Mathematical Institute, pp. 115–V125.

[34] Christoph M. Wintersteiger, Youssef Hamadi & Leonardo de Moura (2009): *A Concurrent Portfolio Approach to SMT Solving*. In: *Proc. of CAV*, *Lecture Notes in Computer Science* 5643, Springer, pp. 715–720, doi:10.1007/978-3-642-02658-4_60.