Improving the Efficiency of Reasoning Through Structure-Based Reformulation*

Eyal Amir[1] and Sheila McIlraith[2]

[1] Department of Computer Science, Stanford University, Stanford, CA 94305, eyal.amir@cs.stanford.edu
[2] Knowledge Systems Lab, Department of Computer Science, Stanford University, Stanford, CA 94305, sheila.mcilraith@cs.stanford.edu

Abstract. We investigate the possibility of improving the efficiency of reasoning through structure-based partitioning of logical theories, combined with partition-based logical reasoning strategies. To this end, we provide algorithms for reasoning with partitions of axioms in first-order and propositional logic. We analyze the computational benefit of our algorithms and detect those parameters of a partitioning that influence the efficiency of computation. These parameters are the number of symbols shared by a pair of partitions, the size of each partition, and the topology of the partitioning. Finally, we provide a greedy algorithm that automatically reformulates a given theory into partitions, exploiting the parameters that influence the efficiency of computation.

1 Introduction

There is growing interest in building large knowledge bases (KBs) of everyday knowledge about the world, teamed with theorem provers to perform inference. Three such systems are Cycorp's Cyc, and the High Performance Knowledge Base (HPKB) systems developed by Stanford's Knowledge Systems Lab (KSL) [21] and by SRI (e.g., [13]). These KBs comprise tens/hundreds of thousands of logical axioms. One approach to dealing with the size and complexity of these KBs is to structure the content in some way, such as into multiple domain- or task-specific KBs, or into microtheories. In this paper, we investigate how to reason effectively with partitioned sets of logical axioms that have overlap in content, and that may even have different reasoning engines. Furthermore, we investigate the problem of how to exploit structure inherent in a set of logical axioms to induce a partitioning of the axioms that will improve the efficiency of reasoning.

To this end, we propose *partition-based* logical reasoning algorithms, for reasoning with logical theories[1] that are decomposed into related partitions of axioms. Given a partitioning of a logical theory, we use Craig's interpolation theorem [16] to prove the soundness and completeness of a forward message-passing algorithm and an algorithm for propositional satisfiability. The algorithms are designed so that, without loss of generality, reasoning within a partition can be realized by an arbitrary consequence-finding

* Much of the material presented in this abstract appeared in [2].
[1] In this paper, every set of axioms is a *theory* (and vice versa).

engine, in parallel with reasoning in other partitions. We investigate the impact of these algorithms on resolution-based inference, and analyze the computational complexity for our partition-based SAT.

A critical aspect of partition-based logical reasoning is the selection of a *good* partitioning of the theory. The computational analysis of our partition-based reasoning algorithms provides a metric for identifying parameters of partitionings that influence the computation of our algorithms: the *bandwidth* of communication between partitions, the size of each partition, and the topology of the partitions graph. These parameters guide us to propose a greedy algorithm for decomposing logical theories into partitions, trying to optimize these parameters.

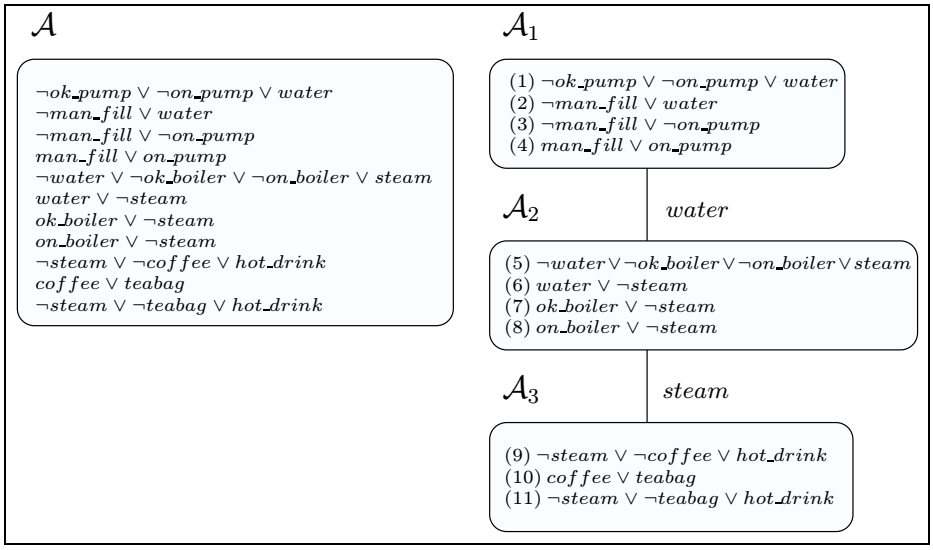

Fig. 1. A partitioning of $\mathcal{A}$ and its intersection graph.

Surprisingly, there has been little work on the specific problem of exploiting structure in theorem proving in the manner we propose. This can largely be attributed to the fact that theorem proving has traditionally examined mathematics domains, that do not necessarily have structure that supports decomposition. Nevertheless, there are many areas of related work, some of which we discuss at the end of this paper.

2 Partition-Based Theorem Proving

In this section we address the problem of how to reason with an already partitioned propositional or first-order logic (FOL) theory. In particular, we propose a forward message-passing algorithm, in the spirit of Pearl [34], and examine the effect of this algorithm on resolution-based inference.

$\{\mathcal{A}_i\}_{i\leq n}$ is a *partitioning* of a logical theory $\mathcal{A}$ if $\mathcal{A} = \bigcup_i \mathcal{A}_i$. Each individual $\mathcal{A}_i$ is called a *partition*, and $\mathcal{L}(A_i)$ is its signature (the non-logical symbols). Each such partitioning defines a labeled graph $G = (V, E, l)$, which we call the *intersection graph*. In the intersection graph, each node i represents an individual partition $\mathcal{A}_i$, ($V = \{1, ..., n\}$), two nodes i, j are linked by an edge if $\mathcal{L}(\mathcal{A}_i)$ and $\mathcal{L}(\mathcal{A}_j)$ have a symbol in common ($E = \{(i, j) \mid \mathcal{L}(\mathcal{A}_i) \cap \mathcal{L}(\mathcal{A}_j) \neq \emptyset\}$), and the edges are labeled with the set of symbols that the associated partitions share ($l(i, j) = \mathcal{L}(\mathcal{A}_i) \cap \mathcal{L}(\mathcal{A}_j)$). We refer to $l(i, j)$ as the *communication language* between partitions $\mathcal{A}_i$ and $\mathcal{A}_j$. We ensure that the intersection graph is connected by adding a minimal number of edges to E with empty labels, $l(i, j) = \emptyset$.

We illustrate the notion of a partitioning in terms of the simple propositional theory $\mathcal{A}$, depicted at the top of Figure 1. This set of axioms captures the functioning of aspects of an expresso machine. The top four axioms denote that if the machine pump is OK and the pump is on then the machine has a water supply. Alternately, the machine can be filled manually, but it is never the case that the machine is manually filling while the pump is on. The second four axioms denote that there is steam if and only if the boiler is OK and is on, and there is a supply of water. Finally, there is always either coffee or tea. Steam and coffee (or tea) result in a hot drink.

2.1 Forward Message Passing

In this section we propose a forward message-passing algorithm for reasoning with partitions of first-order and propositional logical axioms. Figure 2 describes our forward message-passing algorithm, FORWARD-M-P (MP) for finding the truth value of query formula Q whose signature is in $\mathcal{L}(\mathcal{A}_k)$, given partitioned theory $\mathcal{A}$ and graph $G = (V, E, l)$, possibly the intersection graph of $\mathcal{A}$, but not always so.

PROCEDURE FORWARD-M-P($\{\mathcal{A}_i\}_{i\leq n}$, G, Q)
$\{\mathcal{A}_i\}_{i\leq n}$ a partitioning of the theory $\mathcal{A}$, $G = (V, E, l)$ a graph describing the connections between the partitions, Q a query formula in the language of $\mathcal{L}(\mathcal{A}_k)$ ($k \leq n$).

1. Let $dist(i, j)$ ($i, j \in V$) be the length of the shortest path between i, j in G. Let $i \prec j$ iff $dist(i, k) < dist(j, k)$ ($\prec$ is a strict partial order).
2. Concurrently perform consequence finding for each of the partitions $\mathcal{A}_i$, $i \leq n$.
3. For every $(i, j) \in E$ such that $i \prec j$, if we prove $\mathcal{A}_j \models \varphi$ and φ's signature is in $\mathcal{L}(l(i, j))$, then add φ to the set of axioms of $\mathcal{A}_i$.
4. If we proved Q in $\mathcal{A}_k$, return YES.

Fig. 2. A forward message-passing algorithm.

This algorithm exploits consequence finding (step 2) to perform reasoning in the individual partitions. Consequence finding was defined by Lee [27] to be the problem of finding all the logical consequences of a theory or sentences that subsume them.

In MP, we can use any sound and complete consequence-finding algorithm. The *resolution rule* is complete for consequence finding (e.g., [27, 41]) and a the same is

Using FORWARD-M-P to prove hot_drink			
Part.	Resolve	Generating	
$\mathcal{A}_1$	(2) , (4)	$on\_pump \vee water$	(m1)
$\mathcal{A}_1$	$(m1), (1)$	$ok\_pump \vee water$	(m2)
$\mathcal{A}_1$	$(m2), (12)$	$water$	(m3)
		clause $water$ passed from $\mathcal{A}_1$ to $\mathcal{A}_2$	
$\mathcal{A}_2$	(m3) , (5)	$ok\_boiler \wedge on\_boiler \supset steam$	(m4)
$\mathcal{A}_2$	(m4) , (13)	$\neg on\_boiler \vee steam$	(m5)
$\mathcal{A}_2$	(m5) , (14)	$steam$	(m6)
		clause $steam$ passed from $\mathcal{A}_2$ to $\mathcal{A}_3$	
$\mathcal{A}_3$	(9) , (10)	$\neg steam \vee teabag \vee hot\_drink$	(m7)
$\mathcal{A}_3$	(m7) , (11)	$\neg steam \vee hot\_drink$	(m8)
$\mathcal{A}_3$	(m8) , $(m6)$	hot_drink	(m9)

Fig. 3. A proof of hot_drink from $\mathcal{A}$ in Figure 1 after asserting ok_pump (12) in $\mathcal{A}_1$ and ok_boiler (13), on_boiler (14) in $\mathcal{A}_2$.

true for several *linear resolution* variants (e.g., [31, 25]). *Semantic resolution* and *set-of-support resolution* are complete for consequence finding, but only in a limited way [42]. Such consequence finders are used for prime implicate generation in applications such as diagnosis. Inoue [25] provides an algorithm for selectively generating consequences or *characteristic clauses* in a given sub-vocabulary. We can exploit this algorithm to focus consequence finding on axioms whose signature is in the communication language of the partition. Figure 3 illustrates an execution of MP using resolution.

Given a partitioning whose intersection graph forms an *undirected tree*, our MP algorithm is a sound and complete proof procedure. The completeness relies on Craig's Interpolation Theorem [16], as we prove in [2]. When the intersection graph is not a tree, the cycles in the graph must first be broken and then MP applied. In [2] we present an algorithm, BREAK-CYCLES that transforms the intersection graph into a tree by removing edges from the graph and adding their labels to some of the edges that are left. We then show that MP combined with BREAK-CYCLES is sound and complete.

Theorem 1 (Craig's Interpolation Theorem [16]). *If $\alpha \vdash \beta$, then there is a formula γ involving only symbols common to both α and β, such that $\alpha \vdash \gamma$ and $\gamma \vdash \beta$.*

It is important to notice that although MP was illustrated with respect to an example in propositional logic, it was designed primarily for first-order theorem proving. The results above are valid for first-order theories as well as propositional ones. A procedure solely for propositional satisfiability is presented in Section 3. We discuss the application and limitation of MP in the following section.

2.2 Resolution-Based Inference

We now analyze the effect of forward message-passing (MP) on the computational efficiency of resolution-based inference, and identify some of the parameters of influence. Current measures for comparing automated deduction strategies are insufficient for our purposes. Proof length (e.g., [24]) is only marginally relevant. More relevant is comparing the sizes of search spaces of different strategies (e.g., [35]). Both measures do not precisely address our needs, but we use them here, leaving better comparison for future work.

In a *resolution search space*, each node includes a set of clauses, and properties relevant to the utilized resolution strategy (e.g., clause parenthood information). Each arc is a resolution step allowed by the strategy. In contrast, in an *MP resolution search space* the nodes also include partition membership information. Further, each arc is a resolution step allowed by the utilized resolution strategy that satisfies either of: (1) the two axioms are in the same partition, or (2) one of the axioms is in partition $\mathcal{A}_j$, the second axiom is drawn from its communication language $l(i,j)$, and the query-based ordering allows the second axiom to be sent from $\mathcal{A}_i$ to $\mathcal{A}_j$. Legal sequence of resolutions correspond to paths in these spaces.

Proposition 1. *Let $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$ be a partitioned theory. Any path in the MP resolution search space of $\{\mathcal{A}_i\}_{i \leq n}$ is also a path in the resolution search space of the unpartitioned theory $\mathcal{A}$.*

From the point of view of proof length, it follows that the longest proof without using MP is as long or longer than the longest MP proof. Unfortunately, the shortest MP proof may be longer than the shortest possible proof without MP. This observation can be quantified most easily in the simple case of only two partitions $\mathcal{A}_1, \mathcal{A}_2$. The set of messages that need to be sent from $\mathcal{A}_1$ to $\mathcal{A}_2$ to prove Q is exactly the interpolant γ promised by Theorem 1 for $\alpha = A_1$, $\beta = \mathcal{A}_2 \Rightarrow Q$. The MP proof has to prove $\alpha \vdash \gamma$ and $\gamma \vdash \beta$. Carbone [12] showed that, if γ is a minimal interpolant, then for many important cases the proof length of $\alpha \vdash \gamma$ together with the proof length of $\gamma \vdash \beta$ is in $O(k^2)$ (for sequent calculus with cuts), where k is the length of the minimal proof of $\alpha \vdash \beta$.

In general, the size of γ itself may be large. In fact, in the propositional case it is an open question whether or not the size of the smallest interpolant can be polynomially bounded by the size of the two formulae α, β. A positive answer to this question would imply an important consequence in complexity theory, namely that $NP \cap coNP \subseteq P/poly$ [10]. Nevertheless, there is a good upper bound on the length of the interpolation formula as a function of the length of the minimal proof [26] : If α, β share l symbols, and the resolution proof of $\alpha \vdash \beta$ is of length k, then there is an interpolant γ of length $min(kl^{O(1)}, 2^l)$.

The limits reported above are important for computational space considerations. The facts above imply a limit on the space used in the propositional case of MP: It may in general take exponential space, but only in as much as the underlying proof procedure does. It does not add more than a polynomial amount of space on top of a resolution theorem prover. A comparison between resolution theorem proving and a satisfiability search procedure is reported in [20, 37].

To conclude, we can guarantee low amounts of computation and space, if we make sure the communication language is minimal. Unfortunately, we do not always have control over the communication language, as in the case of multiple KBs that have extensive overlap. In such cases, the communication language between KBs may be large, possibly resulting in a large interpolant. In Section 4 we provide an algorithm for partitioning theories that attempts to minimize the communication language between partitions.

3 Propositional Satisfiability

The algorithm we propose in this section uses a SAT procedure as a subroutine and is back-track free. We describe the algorithm using database notation [45]. $\pi_{p_1,...,p_k}T$ is the *projection* operation on a relation T. It produces a relation that includes all the rows of T, but only the columns named $p_1, ..., p_k$ (suppressing duplicate rows). $S \bowtie R$ is the *natural join* operation on the relations S and R. It produces the cross product of S, R, selecting only those entries that are equal between identically named fields (checking $S.A = R.A$), and discarding those columns that are now duplicated (e.g., $R.A$ will be discarded).

The proposed algorithm shares some intuition with prime implicate generation (e.g., [29, 25]). Briefly, we first compute all the models of each of the partitions (akin to computing the implicates of each partition). We then use $\bowtie$ to combine the partition models into models for $\mathcal{A}$. The algorithm is presented in Figure 4.

PROCEDURE LINEAR-PART-SAT($\{\mathcal{A}_i\}_{i\leq n}$)
$\{\mathcal{A}_i\}_{i\leq n}$ a partitioning of the theory $\mathcal{A}$,

1. $G_0 \leftarrow$ the intersection graph of $\{\mathcal{A}_i\}_{i\leq n}$. $G \leftarrow BREAK\text{-}CYCLES(G_0)$.
2. $\forall i \leq n$, let $L(i) = \bigcup_{(i,j)\in E} l(i,j)$.
3. $\forall i \leq n$, for every truth assignment A to $L(i)$, find satisfying truth assignments of $\mathcal{A}_i \cup A$, storing the result in a table $T_i(A)$.
4. Let $dist(i,j)$ ($i,j \in V$) be the length of the shortest path between i, j in G. Let $i \prec j$ iff $dist(i,1) < dist(j,1)$ ($\prec$ is a strict partial order).
5. Iterate over $i \leq n$ in reverse $\prec$-order (the last i is 1). $\forall j \leq n$ such that $(i,j) \in E$ and $i \prec j$, perform:
   - $T_i \leftarrow T_i \bowtie (\pi_{L(i)}T_j)$ (*Join* T_i with those columns of T_j that correspond to $L(i)$). If $T_i = \emptyset$, return FALSE.
6. Return TRUE.

Fig. 4. An algorithm for SAT of a partitioned propositional theory.

The iterated join that we perform takes time proportional to the size of the tables involved. We keep table sizes below $2^{|L(i)|}$ ($L(i)$ computed in step 2), by *projecting* every table before *joining* it with another. Soundness and completeness follow by an argument similar to that given for MP, which can be found in [2].

Let $\mathcal{A}$ be a partitioned propositional theory with n partitions. Let $m = |\mathcal{L}(\mathcal{A})|$, $L(i)$ the set of propositional symbols calculated in step 2 of LINEAR-PART-SAT, and $m_i = |\mathcal{L}(\mathcal{A}_i) \setminus L(i)|$ ($i \leq n$). Let $a = |\mathcal{A}|$ and k be the length of each axiom.

Lemma 1. *The time taken by LINEAR-PART-SAT to compute SAT for $\mathcal{A}$ is*

$$Time(n, m, m_1, ..., m_n, a, k, |L(1)|, ..., |L(n)|) =$$

$$O(a * k^2 \ + \ n^4 * m \ + \ \sum_{i=1}^{n} (2^{|L(i)|} * f_{SAT}(m_i))),$$

where f_{SAT} is the time to compute SAT. Furthermore, if $P \neq NP$ and in G all the partitions $\mathcal{A}_i$ have the same number of propositional symbols, then LINEAR-PART-SAT computes SAT for $\mathcal{A}$ in time

$$Time(m, n, l, d) = O(n * 2^{d*l} * f_{SAT}(\frac{m}{n})).$$

where $d = max_{v \in V} d(v)$ ($d(v)$ is the degree of node v) and $l = max_{i,j \leq n} |l(i, j)|$.

For example, if we partition a given theory $\mathcal{A}$ into only two partitions ($n = 2$), sharing l propositional symbols, the algorithm will take time $O(2^l * f_{SAT}(\frac{m}{2}))$. Assuming $P \neq NP$, this is a signifi cant improvement over a simple SAT procedure, for every l that is small enough ($l < \frac{\alpha m}{2}$, and $\alpha \leq 0.582$ [38, 14]).

It is important to notice that both the MP procedure (Figure 2) and the LINEAR-PART-SAT procedure (Figure 4) focus on structured problems and not random ones. In structured problems the labels of the links are small, leading to only a small overhead in space. Lemma 1 and Section 2.2 show that the size of tables and size of messages sent is exponentially dependent on the size of links between partitions. In a random problem it is possible that in any decomposition the links may be large, leading to possibly exponential computational space. In structured problems the links are small, thus avoiding such risk.

4 Decomposing a Logical Theory

The algorithms presented in previous sections assumed a given partitioning. In this section we address the critical problem of automatically decomposing a set of propositional or FOL clauses into a partitioned theory. Guided by the results of previous sections, we propose guidelines for achieving a good partitioning, and present a greedy algorithm that decomposes a theory following these guidelines.

4.1 A Good Partitioning

Given a theory, we wish to fi nd a partitioning of that theory that minimizes the formula derived in Lemma 1. To that end, assuming $P \neq NP$, we want to minimize the following parameters for all $i \leq n$.

1. $|L(i)|$ - the total number of symbols contained in all links to/from node i. If G_0 is already a tree, this is the number of symbols shared between the partition $\mathcal{A}_i$ and the rest of the theory $\mathcal{A} \setminus \mathcal{A}_i$.
2. m_i - the number of symbols in a partition, less those in the links, i.e., in $\mathcal{A}_i \setminus L(i)$. Typically, having more partitions causes m_i to become smaller.
3. n - the number of partitions.

Also, a simple analysis shows that given *fixed* values for l, d in Corollary 1, the maximal n that maintains l, d such that also $n \leq ln2 * \alpha * m$ ($\alpha = 0.582$ [38, 14]) yields an optimal bound for LINEAR-PART-SAT. In Section 2.2 we saw that the same parameters influence the number of derivations we can perform in MP: $|L(i)|$ influences the interpolant size and thus the proof length, and m_i influences the number of deductions/resolutions we can perform. Thus, we would like to minimize the number of symbols shared between partitions and the number of symbols in each partition less those in the links.

The question is, how often do we get large n (many partitions), small m_i's (small partitions) and small $|L(i)|$'s (weak interactions) in practice. We believe that in domains that deal with engineered physical systems, many of the domain axiomatizations have these structural properties. Indeed, design of engineering artifacts encourages modularization, with minimal interconnectivity (see [1, 28, 13]). More generally, we believe axiomatizers of large corpora of real-world knowledge tend to try to provide structured representations following some of these principles.

4.2 Vertex Min-Cut in the Graph of Symbols

To exploit the partitioning guidelines proposed in the previous subsection, we represent our theory $\mathcal{A}$ using a *symbols graph* that captures the features we wish to minimize. $G = (V, E)$ is a symbols graph for theory $\mathcal{A}$ such that each vertex $v \in V$ is a symbol in $\mathcal{L}(\mathcal{A})$, and there is an edge between two vertices if their associated symbols occur in the same axiom of $\mathcal{A}$, i.e., $E = \{(a, b) \mid \exists \alpha \in \mathcal{A}$ s.t. a, b appear in $\alpha\}$.

Figure 5 illustrates the symbols graph of theory $\mathcal{A}$ (top) from Figure 1 and the connected symbols graphs (bottom) of the individual partitions $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$. The symbols ok_p, on_p, m_f, w, ok_b, on_b, s, c, t, h_d are short for ok_pump, on_pump, man_fill, $water$, ok_boiler, on_boiler, $steam$, $coffee$, $teabag$, hot_drink, respectively. Notice that each axiom creates a clique among its constituent symbols. To minimize the number of symbols shared between partitions (i.e., $|L(i)|$), we must find partitions whose symbols have minimal *vertex separators* in the symbols graph.

We briefly describe the notion of a vertex separator. Let $G = (V, E)$ be an undirected graph. A set S of vertices is called an (a, b) *vertex separator* if $\{a, b\} \subset V \setminus S$ and every path connecting a and b in G passes through at least one vertex contained in S. Thus, the vertices in S split the path from a to b. Let $N(a, b)$ be the least cardinality of an (a, b) vertex separator. The *connectivity* of the graph G is the minimal $N(a, b)$ for any $a, b \in V$ that are not connected by an edge.

Figure 6 presents a greedy recursive algorithm that uses Even's algorithm to find *sets of vertices* that together separate a graph into partitions. The algorithm returns a set of symbols sets that determine the separate subgraphs. Different variants of the algorithm

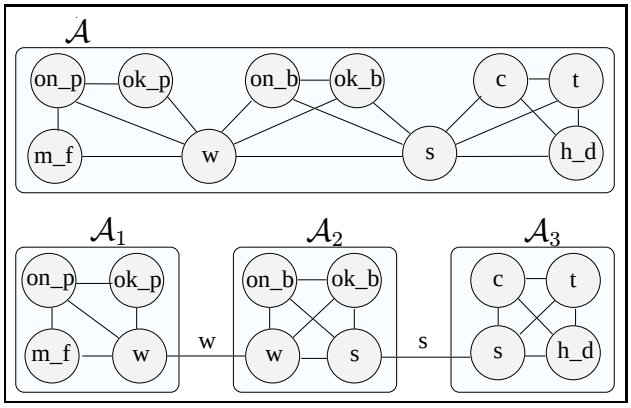

Fig. 5. Decomposing $\mathcal{A}$'s symbols graph.

yield different structures for the intersection graph of the resulting partitioning. As is, SPLIT returns sets of symbols that result in a chain of partitions. We obtain arbitrary *trees*, if we change step 2(c) to find a minimum separator that does not include a, b (not required to separate a, b). We obtain arbitrary *graphs*, if in addition we do not aggregate R into r in step 4.

Proposition 2. *Procedure SPLIT takes time* $O(|V|^{\frac{5}{2}} * |E|)$.

Finally, to partition a theory $\mathcal{A}$, create its symbols graph G and run SPLIT(G, M, l, nil, nil). For each set of symbols returned, define a partition $\mathcal{A}_i$ that includes all the axioms of $\mathcal{A}$ in the language defined by the returned set of symbols.

We know of no easy way to find an optimal selection of l (the limit on the size of the links) and M (the limit on the number of symbols in a partition) without having prior knowledge of the dependency between the number (and size) of partitions and l. However, we can find out when a partitioning no longer helps computation (compared to the best time bound known for SAT procedures [38, 14]). Our time bound for the procedure is lower than $\Theta(2^{\alpha m})$ when $l \leq \frac{\alpha m - \alpha m_i - lgn}{d}$ ($i = \mathit{argmax}_j m_j$). In particular, if $l > \frac{m}{2}$, a standard deterministic SAT procedure will be better. Hence, l and M are perhaps best determined experimentally.

5 Related Work

Many AI researchers have exploited structure to improve the efficiency of reasoning (e.g., Bayes Nets [34], Markov decision processes [11], CSPs [18], and model-based diagnosis [17]). There is also a vast literature in both clustering and decomposition techniques.

Decomposition has not been exploited in theorem proving until recently (see [6, 7]). We believe that part of the reason for this lack of interest has been that theorem proving has focused on mathematical domains that do not necessarily have structure that

PROCEDURE SPLIT(G, M, l, a, b)
$G = (V, E)$ is an undirected graph. M is the limit on the number of symbols in a partition. l is the limit on the size of links between partitions. a, b are in V or are nil.

1. If $|V| < M$ then return the graph with the single symbol set V.
2. (a) If a and b are both nil, find a minimum vertex separator R in G. (b) Otherwise, if b is nil, find a minimum vertex separator R in G that does not include a. (c) Otherwise, find a minimum vertex separator R in G that separates a and b.
   If $R > l$ then return the graph with the single symbol set V.
3. Let G_1, G_2 be the two subgraphs of G separated by R, with R included in both subgraphs.
4. Create G'_1, G'_2 from G_1, G_2, respectively, by aggregating the vertices in R into a single vertex r, removing all self edges and connecting r with edges to all the vertices connected by edges to some vertices in R.
5. Set $V^1 = SPLIT(G'_1, M, l, r, a)$ and $V^2 = SPLIT(G'_2, M, l, r, b)$.
6. Replace r in V^1, V^2 by the members of R. Return V^1, V^2.

Fig. 6. An algorithm for generating symbol sets that define partitions.

supports decomposition. Work on theorem proving has focused on decomposition for parallel implementations [8, 5, 15, 43] and has followed decomposition methods guided by lookahead and subgoals, neglecting the types of structural properties we used here. Another related line of work focuses on combining logical systems (e.g., [32, 40, 3, 36, 44]). Contrasted with this work, we focus on interactions between theories with overlapping signatures, the effi ciency of reasoning, and automatic decomposition.

Decomposition for propositional SAT has followed different tracks. Perhaps the most relevant work to ours is [19], which presented algorithms for reasoning with decomposed CSPs. These can be used for SAT, using a given decomposition. In comparison, the algorithm we presented for partitioned SAT does not produce all the models possible in each partition, as proposed in [19]. Instead, it fi nds the truth values for propositions on the links that are extendible to a satisfying truth assignment for the whole partition. This reduces our computation time and makes it more dependent on the links' sizes rather than on partition sizes. Other work focused on heuristics for clause weighting or symbol ordering (e.g., [39, 20]). Concurrently to our work, Rish and Dechter [37] have proposed an algorithm similar to our MP for the case of propositional ordered resolution. Aside from looking at only a limited case (ordered resolution, propositional logic), they allow excessive computation (they do the equivalent of performing all possible resolutions in each partition, twice) thus possibly using exponential amounts of space and time over and above MP in the same settings.

Other SAT decomposition methods include [33] which suggested a decomposition procedure that represents the theory as a hypergraph of clauses and divides the propositional theory into two partitions (heuristically minimizing the number of hyperedges), modifying ideas described in [22]. [15] developed an algorithm that partitions a propositional theory into connected components. Both [15, 33] performed experiments that demonstrated a decrease in the time required to prove test sets of axioms unsatisfi able.

Compared to work on automated decomposition for reasoning in Bayes-networks and CSPs (e.g., [4]), our work is the first to address the problem of defining guidelines and parameters for good decompositions of sets of axioms for the purpose of logical reasoning. Earlier work assumes that reasoning inside a given partition takes time $O(2^m)$ (m is the number of propositions in the partition), which is not necessarily the case in logical reasoning (in either model finding or proof finding). This has led to a decomposition algorithm that focuses on minimal links rather than minimal partitions.

Finally, work on formalizing and reasoning with *context* (e.g., [30]) can be related to partition-based logical reasoning by viewing the contextual theories as interacting sets of theories. Unfortunately, to introduce explicit contexts, a language that is more expressive than FOL is needed. Consequently, a number of researchers have focused on context for propositional logic, while much of the reasoning work has focused on proof checking (e.g., GETFOL [23]). There have been few reported successes with automated reasoning; [9] presents one example.

6 Conclusions

We have shown that structured logical theories can be reformulated into partitioned logical theories such that reasoning over those partitions has computational advantages for theorem provers and SAT solvers. Theorem proving strategies, such as resolution, can use such decompositions to constrain search. Partition-based reasoning will improve the efficiency of propositional SAT solvers if the theory is decomposable into partitions that share only small numbers of symbols. We have provided sound and complete algorithms for reasoning with partitions of related logical axioms, both in propositional and FOL. Further, we analyzed the effect of partition-based logical reasoning on resolution-based inference, both with respect to proof search space size, and with respect to the length of a proof. We also analyzed the performance of our SAT algorithm and showed that it takes time proportional to SAT solutions on individual partitions and an exponent in the size of the links between partitions. Both algorithms can gain further time efficiency through parallel processing.

Guided by the analysis of our SAT algorithm, we suggested guidelines for achieving a good partitioning and proposed an algorithm for the automatic decomposition of theories that tries to minimize identified parameters. This algorithm generalizes previous algorithms used to decompose CSPs by finding single-vertex separators.

Acknowledgments

We wish to thank Rada Chirkova and Tom Costello for helpful comments on the contents of this paper. We would also like to thank Mike Genesereth, Nils Nilsson and John McCarthy for many interesting discussions on more general topics related to this work. Finally, we thank the SARA2000 reviewers for their thorough and helpful reviews. This research was supported in part by DARPA grant N66001-97-C-8554-P00004 and by AFOSR grant AF F49620-97-1-0207.

References

1. E. Amir. (De)composition of situation calculus theories. In *Proc. National Conference on Artificial Intelligence (AAAI '00)*. AAAI Press/MIT Press, 2000. To appear.
2. E. Amir and S. McIlraith. Partition-based logical reasoning. In *Intl. Conf. on Knowledge Representation and Reasoning (KR '2000)*, pages 389–400, 2000.
3. F. Baader and K. U. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. In *11th Intl. conf. on automated deduction*, volume 607 of *LNAI*, pages 50–65. Springer-Verlag, 1992.
4. A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal junction trees. In *Proc. Twelfth Conference on Uncertainty in Artificial Intelligence (UAI '96)*, pages 81–89. Morgan Kaufmann, 1996.
5. M. P Bonacina. Experiments with subdivision of search in distributed theorem proving. In Markus Hitz and Erich Kaltofen, editors, *Proceedings of the Second International Symposium on Parallel Symbolic Computation (PASCO97)*, pages 88–100. ACM Press, 1997.
6. M. P. Bonacina. A taxonomy of theorem-proving strategies. In *Artificial Intelligence Today – Recent Trends and Developments*, volume 1600 of *LNAI*, pages 43–84. Springer, 1999.
7. M. P Bonacina and J. Hsiang. Parallelization of deduction strategies: an analytical study. *Journal of Automated Reasoning*, 13:1–33, 1994.
8. M. P Bonacina and J. Hsiang. Distributed deduction by Clause-Diffusion: distributed contraction and the Aquarius prover. *J. of Symbolic Computation*, 19:245–267, March 1995.
9. P.E. Bonzon. A reflective proof system for reasoning in contexts. In *Proc. Nat'l Conf. on Artificial Intelligence (AAAI '97)*, pages 398–403, 1997.
10. R. Boppana and M. Sipser. The complexity of finite functions. In *Handbook of Theoretical Computer Science*, volume 1. Elsevier and MIT Press, 1990.
11. C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *14th Intl. Joint Conf. on Artificial Intelligence (IJCAI '95)*, pages 1104–1111, 1995.
12. A. Carbone. Interpolants, cut elimination and flow graphs for the propositional calculus. *Annals of Pure and Applied Logic*, 83(3):249–299, 1997.
13. P. Cohen, R. Schrag, E. Jones, A. Pease, A. Lin, B. Starr, D. Gunning, and M. Burke. The DARPA high-performance knowledge bases project. *AI Magazine*, 19(4):25–49, 1998.
14. S. A. Cook and D. G. Mitchell. Finding hard instances of the satisfiability problem: a survey. In *Dimacs Series in Discrete Math. and Theoretical Comp. Sci.*, volume 35. AMS, 1997.
15. R. Cowen and K. Wyatt. BREAKUP: A preprocessing algorithm for satisfiability testing of CNF formulas. *Notre Dame J. of Formal Logic*, 34(4):602–606, 1993.
16. W. Craig. Linear reasoning. a new form of the herbrand-gentzen theorem. *J. of Symbolic Logic*, 22:250–268, 1957.
17. A. Darwiche. Model-based diagnosis using structured system descriptions. *Journal of Artificial Intelligence Research*, 8:165–222, 1998.
18. R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, 1990.
19. R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.
20. R. Dechter and I. Rish. Directional resolution: The davis-putnam procedure, revisited. In *Intl. Conf. on Knowledge Representation and Reasoning (KR '94)*, pages 134–145. Morgan Kaufmann, 1994.
21. R. Fikes and A. Farquhar. Large-scale repositories of highly expressive reusable knowledge. *IEEE Intelligent Systems*, 14(2), 1999.
22. G. Gallo and G. Urbani. Algorithms for testing the satisfiability of propositional formulae. *J. of Logic Programming*, 7:45–61, 1989.

23. F. Giunchiglia. Getfol manual - getfol version 2.0. Technical Report DIST-TR-92-0010, DIST - University of Genoa, 1994. http://ftp.mrg.dist.unige.it/pub/mrg-ftp/92-0010.ps.gz.
24. A. Haken. The intractability of resolution. *theoretical computer science*, 39:297–308, 1985.
25. K. Inoue. Linear resolution for consequence finding. *Artificial Intelligence*, 56(2-3):301–353, 1992.
26. J. Krajiček. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *J. of Symbolic Logic*, 62(2):457–486, 1997.
27. R. C. Lee. *A Completeness Theorem and a Computer Program for Finding Theorems Derivable from Given Axioms*. PhD thesis, University of California, Berkeley, 1967.
28. D. B. Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.
29. P. Marquis. Knowledge compilation using theory prime implicates. In *14th Intl. Joint Conf. on Artificial Intelligence (IJCAI '95)*, pages 837–843, 1995.
30. J. McCarthy and S. Buvač. Formalizing Context (Expanded Notes). In A. Aliseda, R.J. van Glabbeek, and D. Westerståhl, editors, *Computing Natural Language*, volume 81 of *CSLI Lecture Notes*, pages 13–50. Center for the Study of Language and Information, Stanford U., 1998.
31. E. Minicozzi and R. Reiter. A note on linear resolution strategies in consequence-finding. *Artificial Intelligence*, 3:175–180, 1972.
32. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. on Programming Languages and Systems*, 1(2):245–257, 1979.
33. T. J. Park and A. Van Gelder. Partitioning methods for satisfiability testing on large formulas. In *Proc. Intl. Conf. on Automated Deduction (CADE-13)*, pages 748–762. Springer-Verlag, 1996.
34. J. Pearl. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann, 1988.
35. D. A. Plaisted. The search efficiency of theorem proving strategies. In *Proc. Intl. Conf. on Automated Deduction (CADE-12)*, pages 57–71, 1994.
36. C. Ringeissen. Cooperation of decision procedures for the satisfiability problem. In F. Baader and K.U. Schulz, editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany)*, Applied Logic, pages 121–140. Kluwer, March 1996.
37. I. Rish and R. Dechter. Resolution versus search: two strategies for SAT. *J. of Approximate Reasoning*, To appear, 2000.
38. I. Schiermeyer. Pure literal look ahead: an $O(1,497^n)$ 3-satisfiability algorithm (extended abstract). Technical report, University of Köln, 1996. Workshop on the Satisfiability Problem, Siena April 29-May 3.
39. B. Selman and H. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *13th Intl. Joint Conf. on Artificial Intelligence (IJCAI '93)*, 1993.
40. R. E. Shostak. Deciding combinations of theories. *J. of the ACM*, 31:1–12, 1984.
41. J. R. Slagle. Interpolation theorems for resolution in lower predicate calculus. *J. of the ACM*, 17(3):535–542, July 1970.
42. J. R. Slagle, C.-L. Chang, and R. C. T. Lee. Completeness theorems for semantic resolution in consequence-finding. In *1st Intl. Joint Conf. on Artificial Intelligence (IJCAI '69)*, pages 281–285, 1969.
43. C. B. Suttner. SPTHEO. *Journal of Automated Reasoning*, 18:253–258, 1997.
44. C. Tinelli and M. T. Harandi. A new correctness proof of the Nelson–Oppen combination procedure. In F. Baader and K.U. Schulz, editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany)*, Applied Logic, pages 103–120. Kluwer, March 1996.
45. J. D. Ullman. *Principles of Database and knowledge-base systems*, volume 1. Computer Science Press, 1988.