To appear in Proceedings of 18th International Joint Conference on Artificial Intelligence (IJCAI-03), *August 2003.*

Practical Partition-Based Theorem Proving for Large Knowledge Bases

Bill MacCartney
Knowledge Systems Lab
Computer Science Dept.
Stanford University
bmac@ksl.stanford.edu

Sheila McIlraith
Knowledge Systems Lab
Computer Science Dept.
Stanford University
sam@ksl.stanford.edu

Eyal Amir
Computer Science Div.
University of California
Berkeley, CA
eyal@cs.berkeley.edu

Tomás E. Uribe
AI Center
SRI International
Menlo Park, CA
uribe@ai.sri.com

Abstract

Query answering over commonsense knowledge bases typically employs a first-order logic theorem prover. While first-order inference is intractable in general, provers can often be hand-tuned to answer queries with reasonable performance in practice. Appealing to previous theoretical work on *partition-based reasoning*, we propose resolution-based theorem proving strategies that exploit the structure of a KB to improve the efficiency of reasoning. We analyze and experimentally evaluate these strategies with a testbed based on the SNARK theorem prover. Exploiting graph-based partitioning algorithms, we show how to compute a *partition-derived ordering* for ordered resolution, with good experimental results, offering an automatic alternative to hand-crafted orderings. We further propose a new resolution strategy, *MFS resolution*, that combines partition-based message passing with focused sublanguage resolution. Our experiments show a significant reduction in the number of resolution steps when this strategy is used. Finally, we augment partition-based message passing, partition-derived ordering, and MFS by combining them with the set-of-support restriction. While these combinations are incomplete, they often produce dramatic improvements in practice. This work presents promising practical techniques for query answering with large and potentially distributed commonsense KBs.

1 Introduction

Theorem proving in first-order logic (FOL) is intractable in general. Nevertheless, first-order provers are commonly used for query answering over large knowledge bases (KBs) containing thousands of axioms, such as Cycorp's Cyc and the High Performance Knowledge Base (HPKB) [4]. To make headway in large KBs, theorem provers usually require KB-specific tuning and customization.

Partition-based reasoning (PBR) [2; 12] promises to speed up reasoning, without manual tuning, by exploiting the structure implicit in such large commonsense KBs, which typically contain loosely coupled clusters of domain knowledge. PBR uses graph-based techniques to automatically partition a logical theory into a network of subtheories minimally connected by links of shared vocabulary. Theorem proving is performed locally in each subtheory, and relevant results are propagated between partitions to achieve globally sound and complete collaborative reasoning.

Previous work on PBR has presented a theoretical framework and made claims about the potential for improving the efficiency of reasoning in practice. In this paper we validate these claims empirically, and introduce novel FOL resolution strategies that exploit PBR techniques to improve the efficiency of reasoning.

Outline: In Section 2, we review the theoretical framework of PBR. In Section 3 we explain how a generic theorem prover may be easily augmented with PBR, and describe the experimental testbed we developed using the SNARK theorem prover [18]. Using this testbed, in Section 4 we compare the performance of the PBR message-passing algorithm (MP) to that of popular resolution strategies [3]. MP far outperforms unrestricted resolution, fares comparably to ordered resolution with a default ordering, and sometimes beats set-of-support (SOS) resolution.

In Section 5 we show how automatic partitioning [1] can induce a *partition-derived ordering* (PDO) for use with ordered resolution. Ordering can be a highly efficient resolution strategy, but its success has previously depended on hand-crafted orderings tailored to a specific KB, often through trial and error. Our PDO is competitive with hand-crafted orderings and far outperforms SNARK's default ordering. This important result will let future theorem provers incorporate efficient automatic ordering.

In Section 6, we present a novel resolution strategy, MFS resolution, and show it to be sound and complete. MFS combines MP with a *focused support* restriction employed within partitions. Focused support is a novel resolution restriction that is complete for consequence-finding in a specified sublanguage. In experiments, MFS significantly reduces the number of resolution steps required to answer queries.

Finally, in Section 7 we examine combinations of each of these strategies (MP, PDO, and MFS) with SOS. While these combinations are, in general, incomplete, they perform well in experimental testing. PDO+SOS was found to outperform every other strategy and combination examined, and its theoretical incompleteness was never encountered in practice.

This paper introduces novel, easily implemented techniques for improving the efficiency of FOL theorem proving with large structured KBs such as commonsense KBs. It is the first to report experimental results for PBR, and the first to examine combinations of MP with other strategies in a theoretical or experimental setting.

2 Background: partition-based reasoning

The PBR framework has two components: graph-based algorithms for automatic partitioning of a theory, and message-passing algorithms for reasoning with the partitioned theory. For further details see [2; 12; 1].

2.1 Automatically partitioning a theory

We say that $\{\mathcal{A}_i\}_{i\leq n}$ is a *partitioning* of a logical theory $\mathcal{A}$ if $\mathcal{A} = \bigcup_i \mathcal{A}_i$. Each $\mathcal{A}_i$ is a set of axioms called a *partition*, $L(\mathcal{A}_i)$ is its signature (the set of non-logical symbols), and $\mathcal{L}(\mathcal{A}_i)$ is its language (the set of formulae built with $L(\mathcal{A}_i)$). Partitions may share symbols and axioms.

To partition a theory, we first construct a *symbols graph* from the axioms, where each vertex represents a symbol in the theory, and two vertices are joined by an edge iff the two symbols appear together in an axiom. We then use one of several *tree decomposition* algorithms [16; 1] to generate a tree where each node corresponds to a tightly connected cluster of symbols, defining a partition $\mathcal{A}_i$ consisting of the axioms in the original theory that contain only those symbols.

The result is a *partition graph* $G : (V, E, l)$, where vertices correspond to partitions, and edges correspond to communication links between partitions, labeled with their *communication vocabulary*: the label $l(i, j)$ on the edge between $\mathcal{A}_i$ and $\mathcal{A}_j$ is the set of symbols shared by the two partitions, that is, $l(i, j) = L(\mathcal{A}_i) \cap L(\mathcal{A}_j)$. Efficient partition-based reasoning depends on finding a tree decomposition that keeps the communication languages small.

Partitioning Algorithms

A *tree decomposition* is a tree of nodes, where each node is a set of symbols and the tree satisfies the following property: if a symbol appears in two different nodes, then it appears in all the nodes and edges on the path between the two nodes. The *width* of a tree decomposition is the size (number of symbols) of its largest node (the largest set of symbols in a single node). An *optimal* tree decomposition is one which has the lowest width among all tree decompositions for the graph. This tree's width is the *treewidth* of the original graph. Tree decomposition algorithms are commonly used in a variety of AI applications, including constraint satisfaction problems and Bayes Nets.

We have performed experiments with two graph partitioning algorithms. The first algorithm is due to [16] and is based on an ordering heuristic named *min fill*. In this heuristic we iterate over the symbols in the symbols graph. At each step, we select a symbol in the graph, add all missing edges between the symbol's neighbors in the graph, and remove that symbol from the graph. The symbol is selected such that the number of added edges is minimal. The tree decomposition is extracted from this order. The experimental results reported here are based on this partitioning algorithm.

The second algorithm is due to [1] and uses a divide-and-conquer approach that is guaranteed to approximate the optimum decomposition by a factor of at most $O(\log t)$, where t is the treewidth of the symbols graph. Iteratively, the algorithm finds a vertex cut in the graph (a set of vertices whose removal separates the graph into two or more parts). This vertex cut, W, is sent recursively to each of the separated parts, and subsequent iterations are required to find a vertex cut that separates W in a *balanced* fashion (here, we may use algorithms such as [11; 9]). The tree decomposition is built from the final parts and the vertex-cuts' vertex sets recursively. In experiments, this algorithm generated partitions somewhat fewer in number and larger in size than the first algorithm, but yielded similar performance in answering queries.

2.2 Reasoning with message passing (MP)

Figure 1 displays Forward-Message-Passing (MP), a PBR algorithm proposed in [2]. It takes as input a partitioned theory $\mathcal{A}$, an associated partition graph $G = (V, E, l)$, and a query formula Q in $\mathcal{L}(\mathcal{A}_k)$, and returns YES if the query was entailed by $\mathcal{A}$. MP first directs all edges in the partition graph G toward the goal partition $\mathcal{A}_k$. Since G is a tree, this means each partition (except the goal) will have exactly one "outbound link" (leading to the next partition on the path to the goal) and an "outbound link vocabulary" L_{out}. MP then performs consequence finding independently in each partition, and propagates each derived formula over the outbound link toward the goal iff the formula's signature matches L_{out}.

```
PROCEDURE FORWARD-M-P (MP)($\{\mathcal{A}_i\}_{i\leq n}$, $G$, $Q$)
$\{\mathcal{A}_i\}_{i\leq n}$ a partitioning of the theory $\mathcal{A}$,
$G : (V, E, l)$ a graph with the connections between partitions,
$Q$ a query in $\mathcal{L}(\mathcal{A}_k)$ (for some $k \leq n$).
  1. Define $i \prec j$ iff $i$ is on the path between $j$ and $k$ in $G$.
  2. Concurrently,
      (a) Perform consequence finding for each of the partitions
          $\mathcal{A}_i$, $i \leq n$.
      (b) For every $(i, j) \in E$ such that $i \prec j$, for every conse-
          quence $\varphi$ of $\mathcal{A}_j$ found (or $\varphi$ in $\mathcal{A}_j$), if $\varphi \in \mathcal{L}(l(i, j))$,
          then add $\varphi$ to the set of clauses of $\mathcal{A}_i$.
      (c) If $Q$ is proven[a] in $\mathcal{A}_k$, return YES.
```

[a]We initially add $\neg Q$ to $\mathcal{A}_k$ and derive inconsistency.

Figure 1: A forward message-passing algorithm.

This algorithm was shown sound and complete when the partition graph corresponds to a tree decomposition, and the reasoning procedure employed in each partition is sound and complete for $\mathcal{L}$*-consequence generation*, where $\mathcal{L}$ is the language defined by the outbound link vocabulary L_{out}:

Definition 2.1. *Given a target language $\mathcal{L}$, a reasoning procedure $\mathcal{R}$ is* complete for $\mathcal{L}$-consequence generation *if every formula in $\mathcal{L}$ logically entailed by a set of axioms S is also entailed by the set of consequences* in $\mathcal{L}$ *that is generated by $\mathcal{R}$ from S.*

Ordered resolution is an example of a reasoning procedure that satisfies this condition [7].

3 Experimental setup

To evaluate the performance of various PBR strategies, we built an experimental platform around SNARK, a resolution-based FOL theorem prover developed by Mark Stickel at the SRI AI Center ([18]). Adding PBR capabilities to SNARK was straightforward. Three extensions were required:

- Associate a set of partitions with each clause.
- Restrict resolution to occur only within partitions.
- Ensure appropriate propagation of new clauses to neighboring partitions.

These modifications should be easy to implement in any resolution-based prover.

3.1 Implementing PBR in SNARK

In SNARK, a *row* is a data structure that stores a clause along with metadata such as the inference method used to derive it. We extend this data structure to include a (non-empty) set of partition IDs indicating the partitions in which the clause resides. A separate data structure holds a description of the partition graph: the partitions, the links that connect them, and the vocabulary for each partition and link.

Resolution is restricted to occur only within partitions: two rows may be resolved only if the intersection of their partition sets is non-empty. The resolvent inherits this intersection as its own partition set, so that it appears in the same partitions as its parents. (This restriction may be superimposed on top of other resolution restrictions, such as set-of-support.)

Finally, to implement message passing between partitions, we examine the vocabulary of every new resolvent. If it matches the outbound link language L_{out} of any partition in its partition set, the partition set is expanded to include the adjacent partition reached by that link.

To generate the initial partitioning, we first load the KB axioms into SNARK and convert them to clausal form. The clauses are passed to an external partitioning tool, which generates the partition graph and defines the vocabulary for each partition and link, according to the particular partitioning strategy selected (see Section 2.1). The partition graph is essentially static, and can be generated just once for each KB, so that the cost of partitioning is amortized over many queries. However, the links between partitions are reoriented by MP on a per-query basis, to ensure that all communication links lead toward the goal partition.

3.2 Sample KBs, evaluation metrics

The greatest problem we encountered was obtaining large KBs of FOL suitable for use as test data. Test suites for theorem provers such as TPTP [19] are biased toward propositional logic (PL) or small problems in FOL, while the minority of test problems containing significant numbers of FOL axioms usually describe an abstract domain and employ only a small number of symbols, making them unsuitable for partitioning. The best application of partitioning will be to commonsense KBs that contain large numbers of both axioms and symbols. Unlike most mathematical theorem proving problems, these KBs exhibit structure that can be exploited by partitioning to improve the efficiency of reasoning. However, the few such KBs publically available tend to employ proprietary extensions to FOL, making their direct comparison in a common testbed problematic.

In the end, we used a subset of the Cyc KB containing 730 axioms (and 150 symbols) concerning spatial relationships, along with a set of queries supplied by an outside source. We are continuing our efforts to remedy the paucity of experimental data by adding other large KBs. We are currently testing queries against the Suggested Upper Merged Ontology (SUMO) [13]; experimental results will be reported at the project website[1].

We collected a variety of statistics on each KB partitioning task and each query trial, including runtimes and elapsed times. But we are most interested in metrics that allow us to compare one trial with another by the size of the search space explored. Processor runtime is an imperfect proxy, since it depends greatly on implementation details that are not our primary concern. Thus, in the remainder of this paper, we focus on *number of resolution steps* as the best measure of the quantity of work required for each trial.

4 Baseline evaluation of MP

As a starting point for evaluating the effectiveness of MP, we compare it with two common resolution restriction strategies: *set-of-support* (SOS) and *ordered resolution*. SOS places the negated query into a designated "set of support" and allows only resolutions involving at least one clause from the set of support, to which newly derived clauses are added. In ordered resolution, a global ordering among predicates is given (by the user), and clauses are resolved upon only on their highest literals (in the predicate ordering).

To assess the effectiveness of a resolution strategy, we examine how much work is required to answer a query using the strategy, relative to using *no* strategy (that is, allowing any possible resolution). As described in Section 3.2, we use the number of resolution steps as a measure of the work done in answering a query. The absolute number of resolution steps is typically in the thousands, but since this number may vary widely from one query to the next, we must normalize in order to make meaningful comparisons across queries. Thus, for each query and strategy, we report the number of resolution steps required to answer the query as a percent of the number of resolution steps required using no strategy.

Results for a representative selection of queries are shown in the first three histograms of Figure 2. Relative to using no strategy, MP reduces the numbers of steps required to answer most queries by one-third to two-thirds.

However, MP is significantly outperformed by SOS on many queries (though not all). Why? This "vanilla" version of MP operates only at the global level: it restricts resolution *between* partitions, but allows unrestricted resolution *within* partitions, equivalent to using no strategy at the local level. In the following sections, we show how to complement MP with smart local strategies for enhanced performance.

The trials for ordered resolution used a default (arbitrary) ordering, so the unimpressive results are no surprise. In Sec-

[1]`http://www.ksl.stanford.edu/projects/Partitioning`

tion 5, we introduce a mechanism for automatically inducing efficient orderings from a partition structure.

5 Partition-derived ordering

A major obstacle to the effective use of ordered resolution is finding a good predicate ordering, which will minimize inference. SNARK provides a default ordering, but as we saw in Section 4, it is undistinguished compared to MP and SOS, though better than using no strategy. Efficient ordered resolution is usually achieved by arduous hand-crafting of a predicate ordering by a theorem-proving expert.

In this section, we propose to use automatic partitioning, and in particular PBR tree-decomposition algorithms, to automatically produce a predicate ordering. This is an important contribution as it paves the way to improving theorem-proving software for ordered resolution, providing more effective ordered resolution for the non-expert, and relieving the expert from arduous hand-coding of predicate ordering.

Figure 4 describes the P2O algorithm for inducing a symbol ordering from a computed partitioning. It takes as input a partition graph $G : (V, E, l)$ for partitioned theory $\mathcal{A} : \bigcup_i \mathcal{A}_i$, and a query Q in $\mathcal{L}(\mathcal{A}_k)$, and outputs $Ord(\mathcal{A})$, a symbol ordering on $\mathcal{A}$. We call this ordering a *partition-derived ordering*. When combined with ordered resolution, we call the strategy PDO. Recall that ordered resolution uses an ordering on predicate symbols, so PDO in fact uses $Ord(\mathcal{A})$ restricted to predicate symbols and ignores the ordering imposed on constant and function symbols.

The underlying intuition is as follows: in MP, we perform resolution and message passing starting with the partitions farthest away from the goal partition. All predicates are resolved *except* those in the outbound link language, which are resolved in the next partition downstream, unless they continue to appear in subsequent outbound link languages. In this way, the ordering of partitions along paths toward the goal determines an ordering among predicates resolved.

```
PROCEDURE P2O($G$, $Q$)
$G : (V, E, l)$ is a partition graph for the partitioned theory $\mathcal{A} : \{\mathcal{A}_i\}_{i \le n}$ and $Q$ is a query in $\mathcal{L}(\mathcal{A}_k)$ $(k \le n)$.
  1. Define $j \prec i$ iff $j$ is on the path between $i$ and $k$ in $G$.
  2. Let $\mathcal{L}_{out_i} = \mathcal{L}(l(i, j))$ for $j$ such that $(i, j) \in E$ and $j \prec i$
     (there is at most one such $j$; if there is none, then let $\mathcal{L}_{out_i} = \emptyset$). $\mathcal{L}_{out_i}$ is the output language of partition $\mathcal{A}_i$.
  3. Let $\mathcal{L}_{in_i} = \bigcup_h \mathcal{L}(l(h, i))$ for $h$ such that $(h, i) \in E$ and
     $i \prec h$ (there may be none or multiple such $h$'s). $\mathcal{L}_{in_i}$ is the input language of partition $\mathcal{A}_i$.
  4. Initialize $Ord(\mathcal{A})$ = empty list.
  5. Traverse the tree from furthest leaves to root following $\prec$, in a decreasing order, for every $i \le n$.
     $Ord(\mathcal{A}) =$
         append$(Ord(\mathcal{A}), \mathcal{L}_{in_i} \setminus \mathcal{L}_{out_i}, \mathcal{L}(\mathcal{A}_i) \setminus \mathcal{L}_{out_i})$.
  6. $Ord(\mathcal{A}) =$ append$(Ord(\mathcal{A}), \mathcal{L}(\mathcal{A}_k),)$.
```

Figure 4: An algorithm to compute a predicate ordering.

Theorem 5.1. *Let $\mathcal{A} = \bigcup\{\mathcal{A}_i\}_{i \le n}$ be a partitioning of $\mathcal{A}$ based on predicates only (so all function and constant symbols appear in all partitions), let G be a tree decomposition corresponding to $\mathcal{A}$, and let Q be a query. Then, ordered resolution with order $Ord(\mathcal{A})$ is equivalent to MP, where inference within each partition uses ordered resolution with an order compatible with $Ord(\mathcal{A})$.*

Proof. For every $p \in L(\mathcal{A})$, let $P(p)$ be the $\prec$-smallest partition, i, such that $p \in \tilde{L}(i)$. There is exactly one such partition; otherwise there would be two that are not connected, contradicting the fact that G is a tree decomposition (it guarantees that if a symbol appears in two partitions then it appears on the edges on the path between them).

Define $<_0$, an order between symbols, by $p <_0 q$ iff p precedes q in $Ord(\mathcal{A})$ (equivalently, $P(p) \prec P(q)$). Define $<_{\mathcal{A}}$ to be a total order that agrees with $<_0$. Let $\{p_i\}_{i \le m}$ be an enumeration of the predicate symbols of $\mathcal{A}$ according to $<_{\mathcal{A}}$. Notice that if ordered resolution with $<_{\mathcal{A}}$ resolved two clauses upon p_i, then p_i is the highest symbol in both clauses.

We show that ordered resolution is equivalent to MP by showing that ordered resolution step can be done by MP, and vice versa. Let C_1, C_2 be two clauses resolved on p by ordered resolution using the order $<_{\mathcal{A}}$. Let i_1, i_2 be such that C_1 is in partition i_1 and C_2 is in partition i_2. Assume that MP cannot resolve the two clauses. That means that they are in different partitions and that at least one of the clauses, say C_2, should be sent to another partition, $\mathcal{A}_j$, with $j \prec i_2$, in order for the two clauses to be resolved (j is the $\prec$-largest partition such that $j \prec i_1$ and $j \prec i_2$).

Since C_2 was not sent to partition j, its vocabulary is not in the label of one of the edges on the path between i_2 and p in G. Let q be a symbol that appears in C_2 but is not on one of the edges in the path between partitions i_2 and j. p is the $<_{\mathcal{A}}$-largest symbol for both C_1 and C_2, meaning that p appears on the path between i_1, i_2 (thus it also appears on the path between i_2, j, which is a subset of the path between i_1, i_2) and that $q <_{\mathcal{A}} p$ (because q does not appear on that path (thus, it is not p) and p is the $<_{\mathcal{A}}$-largest in C_2).

Since q is not in $\tilde{L}(j)$ (otherwise it would be on the path between i_2, j) and the path between j, i_2 includes p, it must be that $P(p) \preceq j$, and $P(p) \preceq j \prec P(q)$ implies that $p <_{\mathcal{A}} q$. This contradicts the previous observation that $q <_{\mathcal{A}} p$. Thus, q appears on the path between i_2, j and C_2 must be sent to partition j. The same holds for C_1. Thus, MP will resolve the two clauses.

The other direction (every resolution in MP is a resolution in ordered resolution) is simpler, and follows from Theorem 5.2 below. □

It is interesting to observe that we can, conversely, generate a theory partitioning from a symbol ordering, should one be given by an expert. The algorithm is as follows: Let S be a stack containing the predicate symbols in logical theory $\mathcal{A}$ in the designated order $Ord(S)$. Let Z be an empty stack. Pop a symbol s from S. Create a set of symbols that includes s, and all the symbols still in S that appear with s in some axiom in $\mathcal{A}$. If this set of symbols is contained by another set on Z, discard it. Otherwise, push it onto Z. Continue until S is empty. Then, set $G : (V, E)$ to be an empty graph, and iteratively pop sets of symbols from Z. For a set of symbols Z_i, add it to V as a neighbor (in G) to a set of symbols in

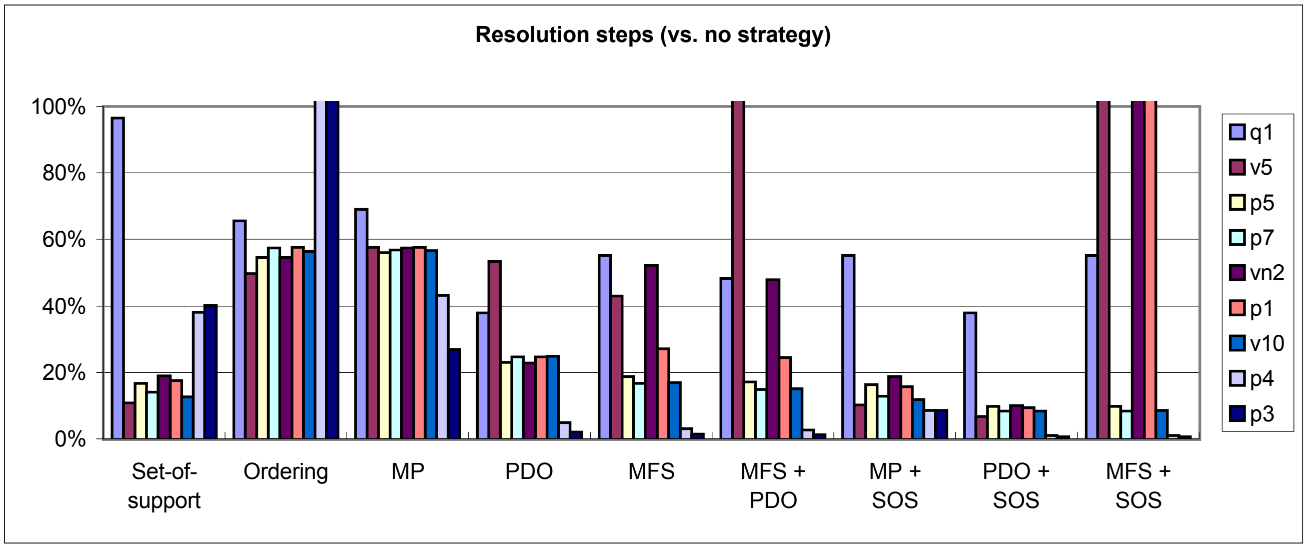

Figure 2: Performance comparison of reasoning strategies: set-of-support (SOS), ordered resolution, vanilla MP, partition-derived ordering (PDO), MP with focused support (MFS), and several combinations of strategies. For a representative selection of queries, we show the number of resolution steps required to answer the query using each strategy, expressed as a percent of the number of steps required using no strategy. Columns exceeding 100% represent timeouts.

V with which it shares the largest number of symbols. When Z is empty, create a partition $\mathcal{A}_i$ for every set of symbols $Z_i \in V$ such that $\mathcal{A}_i$ includes all the axioms from $\mathcal{A}$ that are in $\mathcal{L}(Z_i)$. Call the resulting set of partitions $\mathcal{P}(Ord(S))$, and the graph $G(Ord(S))$.

Theorem 5.2. *Assume that Ord is an order on the nonlogical symbols of theory $\mathcal{A}$, and that Q is a query such that $\mathcal{L}(Q)$ are the final symbols in order Ord. Then, ordered resolution with order Ord is equivalent to MP with partitions $\mathcal{P}(Ord)$ and graph $G(Ord)$, where inference within each partition uses ordered resolution with an order compatible with $Ord(\mathcal{A})$.*

Proof Sketch. To see that MP does not perform more resolutions than ordered resolution with order $\leq_{\mathcal{A}}$, we only need to notice that every partition corresponds to a *bucket* in the ordered resolution algorithm. If MP resolves C_1, C_2 on predicate P, then C_1, C_2 were together in some partition j, and were resolved on their highest predicate. Thus, C_1, C_2 will also be resolved by ordered resolution with order $\leq_{\mathcal{A}}$. □

Proposition 5.3. *PDO is sound and complete.*

In our experimental evaluation, illustrated in Figure 2, PDO made a strong showing. Its performance was comparable with that of SOS for most queries; for a few queries, PDO was up to two orders of magnitude more efficient than SOS. Unsurprisingly, PDO invariably outperformed ordered resolution with SNARK's default ordering. PDO also outperformed vanilla MP, which by itself does not focus reasoning at the local level. When MP is augmented with focused support (the MFS combination), PDO's comparative advantage largely disappears. In Section 7, we will see that the performance of PDO can be enhanced even further by combining it with other strategies.

6 A new restriction strategy: MFS resolution

MP operates at the global level to focus reasoning by restricting between-partition resolution and passing relevant results toward the goal partition. However, MP has no impact on local reasoning, that is, reasoning within partitions. In fact, this is one of the strengths of MP: individual partitions may employ heterogeneous reasoning methods, including special-purpose reasoners optimized for particular domains. (As described in Section 2.2, the global soundness and completeness of MP follow from the soundness and completeness of the reasoners used in each partition.) MP can thus be used to orchestrate collaborative reasoning between disparate systems, provided that they share a common vocabulary. Nevertheless, the absence of a local strategy limits gains in efficiency. In this section we exploit PBR to propose a new restriction strategy, *MFS resolution*, that combines global MP with a local strategy for focused sublanguage resolution.

With vanilla MP, all possible resolutions are performed within a partition, but resolvents can be propagated to other partitions only if they are in the outbound link language. We propose a local resolution strategy, *focused support*, that takes inspiration from strategies described by Slagle [17], SOL restriction [8], and SFK-resolution [7].

Definition 6.1 (Focused Support Restriction). *Let $\mathcal{T}$ be a clausal theory and let L be a designated subset of $L(\mathcal{T})$. Initialize S to be the set of clauses in $\mathcal{T}$ that non-exhaustively include symbols from L. Two clauses may be resolved only if one of them is in S and the resolved predicate is not in L. The*

Query	no strategy	ordering	SOS	MP	PDO	MFS	MP+SOS	PDO+SOS	MFS+SOS
Number of resolution steps									
q1	29	19	28	20	11	16	16	11	16
v5	4,343	2,157	467	2,504	2,318	1,869	441	294	–
p5	3,575	1,948	598	1,999	826	673	585	348	350
p7	3,958	2,273	556	2,250	972	664	507	332	334
vn2	3,681	2,007	696	2,115	839	1,922	690	364	–
p1	4,182	2,410	729	2,413	1,029	1,131	653	391	–
v10	3,862	2,182	489	2,187	963	656	458	326	328
p4	–	–	13,191	15,004	1,715	1,088	2,938	382	381
p3	–	–	38,943	25,956	1,942	1,285	8,243	594	566
Runtime in seconds									
q1	0.1	0.1	0.1	0.1	0.0	0.1	0.1	0.1	0.1
v5	31.9	10.2	1.7	27.0	12.1	9.8	4.6	1.2	–
p5	24.8	9.8	2.4	18.5	3.3	3.3	5.8	1.7	2.6
p7	26.5	11.6	2.2	21.4	3.9	3.2	5.4	1.6	1.7
vn2	25.1	9.5	2.7	12.4	3.5	13.3	2.7	1.7	–
p1	27.3	14.0	2.8	20.1	4.0	8.2	6.5	1.8	–
v10	26.4	10.8	2.9	13.7	3.7	3.4	2.0	1.5	1.6
p4	–	–	217.5	454.3	9.6	7.1	32.0	1.7	1.8
p3	–	–	1,191.0	1,067.4	8.8	7.9	100.8	2.1	2.1

Figure 3: Absolute performance metrics for several reasoning strategies: no strategy, ordered resolution using a default ordering, set-of-support (SOS), vanilla MP, partition-derived ordering (PDO), MP with focused support (MFS), and combinations. For a representative selection of queries, we show the number of resolution steps and the runtime required to answer the query using each strategy. Blanks indicate timeouts.

resolvent is added to S.

When focused support is used as a local strategy in PBR, clausal theory $\mathcal{T}$ is an individual partition $\mathcal{A}_i$ from a partitioned theory, and L is the output communication language of that partition $\mathcal{A}_i$.

Focused support is sound and complete for generating consequences in the restricted vocabulary of L. The proof relies on the following lemma:

Lemma 6.2 ([17]). *If P is an ordering of the predicate symbols in a finite, unsatisfiable set U of clauses and if M is a model, then there is a semantic P-deduction of the empty clause from U.*

Roughly, P-deduction resolves clauses C_1, C_2 on predicate R only if C_1 satisfies M, C_2 does not satisfy M, and R is the largest among the predicates in C_2, in some preset order.

Theorem 6.3. *Focused support is sound and complete for L-consequence generation (c.f. Definition 2.1).*

Proof. Let $\mathcal{A}$ be a clausal theory and $D \in \mathcal{L}(L)$ a clause such that $\mathcal{A} \models D$. We show that focused support eventually generates D as a result of $\mathcal{A}$. Assume that $\mathcal{A}$ is consistent. Thus, there is a model M that satisfies $\mathcal{A}$. Let S be as in the definition of the focused support strategy for L.

Lemma 6.2 says that there is a P-resolution of $\{\}$ from $\mathcal{A}$ for the model M and for every ordering P of the predicate symbols. Since everything D resolves with is already in S, we can remove $\neg D$ from the resolution graph of this proof and yield D from the P-deduction of $\mathcal{A}$. This shows that there is a P-deduction of D from $\mathcal{A}$.

To show that there is a focused support deduction of D from $\mathcal{A}$ we still need to show that restricting resolution to occur only on non-L literals does not prevent proving D. Choose P to be an order such that the symbols not in L come before those in L. Sentences that are not in S have no symbols from L. When we resolve sentences from S with those not in S, then clearly the focused support condition holds. We move all the clauses of S that are completely in $\mathcal{L}(L)$ (i.e., have no symbols outside L) into a distinguished set, $\bar{S}$. Now, when we resolve two sentences from S, both of them have symbols that are not in L. Those symbols take precedence over the ones from L, so the focused support condition is held. If the resolvent has only literals from L, then we put it in $\bar{S}$. Otherwise, we put it in S.

Finally, assume that $\bar{S} \not\models D$. Then, $\bar{S} \cup \{\neg D\}$ is consistent. However, $\bar{S} \cup S \models D$ (as shown above with the P-deduction), so $\bar{S} \cup S \cup \{\neg D\}$ is not consistent. However, for sentences in $S, \bar{S}$ resolution with our semantic support condition is equivalent to using ordered resolution with the order only specifying that literals in L are resolved after literals not in L. Since this is known to be complete for L-consequence finding [20; 7], we know that $\bar{S} \models D$, and the proof is done. $\square$

MFS resolution is the combination of MP with the focused support restriction within partitions. As discussed in Section 2.2, MP is sound and complete provided that each partition uses a reasoning procedure that is sound and complete for L-consequence finding. Since focused support is such a procedure, MFS resolution is also sound and complete:

Corollary 6.4 (Soundness and Completeness of MFS). *MP with the focused support restriction* (MFS resolution) *is sound and complete.*

The relative performance of MFS, as depicted in Figure 2, is encouraging. For most queries, the performance of MFS is comparable to that of SOS; and for a minority of queries MFS is one or even two orders of magnitude more efficient.

We also examined the performance of MFS combined with PDO (MFS+PDO), which was slightly superior to that of MFS alone. However, unlike MFS, MFS+PDO is incomplete. Section 7 discusses the incomplete strategy MFS+SOS, which exhibited even better performance.

Theorem 6.5 (Incompleteness of MFS+PDO). *PDO with the focused support restriction* (MFS+PDO) *is incomplete.*

The incompleteness of MFS+PDO can be seen by noticing that focused support is stricter than each of SOS and semantic resolution. When we add PDO to this restriction, we eliminate resolutions with any clause that does not share predicates with the clauses in the negated query. Interestingly, MFS+PDO is very close to being complete, in the following sense. A close examination of the proof of Theorem 6.3 reveals that we can apply a predicate order restriction to clauses that do not contain predicates from L and still maintain completeness for L-consequence finding. Thus, a simple modification of MFS+PDO, allowing clauses in the focused support of a partition to resolve on any non-L literal (without considerations of ordering) is complete.

7 Combinations of strategies

Preceding sections examined three reasoning strategies that are shown to be complete: MP, PDO, and MFS. In this section we examine combinations of these strategies with SOS. Though none of these combinations is complete, they have some interesting theoretical properties, and in practice showed the best performance of all strategies evaluated. Results of experimental evaluation are summarized in Figure 5.

To combine MP with SOS, we maintain a global set of support, which initially contains only the negated query, and we allow resolution between two clauses only if they are in the same partition and at least one of them is in the support. Because messages are sent only toward the goal partition in MP, never away, resolution will occur only in the goal partition unless we place the negated query in every partition whose language includes it. In either case, SOS resolution will occur in partitions in which the negated query appears.

Clearly, the MP+SOS combination is not complete. Information crucial to proving the query may reside in partitions that do not participate in the computation. However, the algorithm determines the clauses that are most relevant to proving the query, in a simple and dynamic way. This sort of inference is used extensively with large KBs, where the selection of the KB fragments needed to answer the query is done by hand. Our automatic selection mechanism may suffice for many applications, and our experiments bear this out.

The argument given for the incompleteness of MP+SOS also shows that PDO+SOS is incomplete. This follows from the relationship between MP and ordered resolution presented

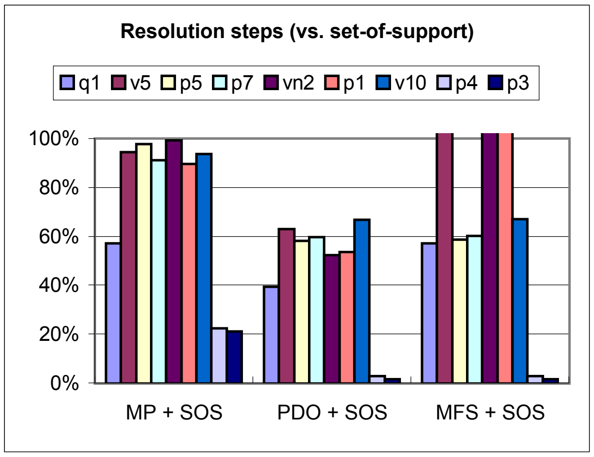

Figure 5: Performance comparison of three combinations of reasoning strategies. In contrast to Figure 2, the vertical axis is normalized relative to SOS.

in Theorem 5.2. When the PDO predicate ordering is determined by the same partitioning being used in MP, then PDO+SOS restricts resolution to only those clauses that include predicates that appear in the support. Despite its incompleteness, PDO+SOS performed extremely well in experimental evaluation. Incompleteness was never encountered in practice, and the PDO+SOS combination answered every query in fewer steps than any other strategy or combination of strategies.

The combination of MFS with SOS (MFS+SOS) is incomplete as well, since MFS is a restricted form of MP, and MP+SOS is incomplete. This incompleteness was revealed in experimental testing, when the MFS+SOS combination failed to answer a few queries. However, when it did not fail, MFS+SOS was highly efficient, answering most queries in as few steps as PDO+SOS.

8 Summary and related work

We propose, analyze, and experimentally evaluate a variety of strategies that improve the efficiency of resolution-based FOL theorem provers. We also propose an algorithm to automatically induce predicate orderings for ordered resolution. Experimental results confirm significant performance improvements using our techniques. This work makes promising practical contributions to the theorem proving community, presented in a manner designed to facilitate replication. (For software and data, see `http://www.ksl.stanford.edu/projects/Partitioning`.) Our results are particularly significant for query answering with commonsense KBs, whose size, structure, and sometimes distributed nature make them well-suited to our structure-based efficiency improvements.

There is a diversity of related work. Our algorithm to induce a predicate ordering is the first to use graphical structure, and in particular treewidth approximation algorithms, with FOL resolution. Current automated approaches to ordering predicates use properties of those predicates, such as their arity (e.g., the Knuth-Bendix method [10] when applied to a uniform weight function, as in [21]), or generate an arbitrary default ordering (e.g., lexical, as in [18]). Our work is most significantly distinguished from work on CSPs (e.g., [6]) and propositional reasoning (e.g., [5; 15]) in that we partition (and subsequently order) a graph that includes all the nonlogical symbols in the theory, whereas propositional methods order nodes in a graph that correspond to propositional symbols and ordering is often dynamic.

Our *focused support* restriction (on which MFS is based) resembles SFK resolution [7] and SOL resolution [8] in its computation of resolvents in a target language. However, in contrast to SFK resolution, our target language is not closed under subsumption. Further, in [7] there is no clear way to determine a predicate ordering. MFS resolution combines focused support with predicate ordering imposed by MP, generating a significant improvement in performance.

Finally, there has been little experimental work studying the behavior of theorem proving strategies on large KBs, and none on commonsense KBs. The success rate of leading theorem provers, such as SPASS, Otter, Setheo, Protein and 3TAP, in formal verification problems with hundreds of axioms, is shown in [14] to depend strongly on how good they are at finding the few relevant axioms needed in the proofs. Our work presents a principled method to successfully elicit such a set of relevant axioms, the PDO+SOS strategy being most notably successful.

In the future, we plan to continue our experimental evaluation with additional KBs. We also hope to demonstrate the use of MP for collaborative theorem proving among distributed KBs and diverse (perhaps special-purpose) reasoners. Finally, we note that PBR naturally enables parallelization of the FOL theorem proving task. The benefits of partition-driven parallelism can be evaluated with little further implementation effort.

Acknowledgements

We thank Mark Stickel, Vinay Chaudhri, Adam Pease, and Richard Waldinger for their assistance, feedback and comments. This research was supported in part by DARPA grant N66001-97-C-8554-P00004, NAVY grant N66001-00-C-8027, DARPA grant N66001-00-C-8018, ARDA grant 2002*H768000*000, ONR MURI Fund N00014-01-1-0890, ONR MURI Fund N00014-00-1-0637, and NSF grant ECS-9873474.

References

[1] E. Amir. Efficient approximation for triangulation of minimum treewidth. In *Proc. UAI '01*, pages 7–15. MK, 2001.

[2] E. Amir and S. McIlraith. Paritition-based logical reasoning. In *Proc. KR '2000*, pages 389–400. MK, 2000.

[3] C.-L. Chang and R. C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[4] 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.

[5] A. Darwiche. Utilizing knowledge-based semantics in graph-based algorithms. In *Proc. AAAI '96*, pages 607–613, 1996.

[6] R. Dechter and J. Pearl. Tree clustering schemes for constraint processing. In *Proc. AAAI '88*, 1988.

[7] A. del Val. A new method for consequence finding and compilation in restricted language. In *Proc. AAAI '99*, pages 259–264. AAAI Press/MIT Press, 1999.

[8] K. Inoue. Linear resolution for consequence finding. *Artificial Intelligence*, 56(2-3):301–353, Aug. 1992.

[9] G. Karypis. Multilevel algorithms for multi-constraint hypergraph partitioning. Technical Report #99-034, University of Minnesota, Department of Computer Science, Minneapolis, MN, 1999.

[10] D. E. Knuth and P. B. Bendix. *Simple word problems in universal algebras*, pages 263–297. Pergamon Press, Oxford, 1970.

[11] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. of the ACM*, 46(6):787–832, 1999.

[12] S. McIlraith and E. Amir. Theorem proving with structured theories. In *IJCAI '01*, pages 624–631. MK, 2001.

[13] I. Niles and A. Pease. Towards a standard upper ontology. In *2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, pages 17–19, Oct. 2001. `http://ontology.teknowledge.com`.

[14] W. Reif and G. Schellhorn. Theorem proving in large theories. In *Proc. FTP'97*, pages 119–124, 1997.

[15] I. Rish and R. Dechter. Resolution versus search: two strategies for SAT. *J. of Automated Reasoning*, 24(1-2):225–275, 2000.

[16] D. J. Rose. Triangulated graphs and the elimination process. *J. of Discrete Mathematics*, 7:317–322, 1974.

[17] J. R. Slagle. Automatic theorem proving with renamable and semantic resolution. *J. of the ACM*, 14(4):687–697, 1967.

[18] M. E. Stickel, R. J. Waldinger, and V. K. Chaudhri. A guide to SNARK. Technical report, SRI International, 2000.

[19] G. Sutcliffe and C. B. Suttner. The TPTP problem library: CNF release v1.2.1. *J. of Automated Reasoning*, 21(2):177–203, 1998. `http://www.cs.miami.edu/~tptp`.

[20] P. Tison. Generalization of consensus theory and application to the minimization of boolean functions. *IEEE Transactions on Electronic Computers*, EC-16:446–456, 1967.

[21] C. Weidenbach, U. Brahm, T. Hillenbrand, E. Keen, C. Theobald, and D. Topic. SPASS version 2.0. In *Proc. CADE-18*, pages 275–279. Springer, 2002.