Optimizing Proof Search in Model Elimination

John Harrison
Åbo Akademi University
Department of Computer Science
Lemminkäisenkatu 14a
20520 Turku
FINLAND
jharriso@abo.fi
http://www.abo.fi/~jharriso/

11th January 1996

Abstract

Many implementations of model elimination perform proof search by iteratively increasing a bound on the total size of the proof. We propose an optimized version of this search mode using a simple divide-and-conquer refinement. Optimized and unoptimized modes are compared, together with depth-bounded and best-first search, over the entire TPTP problem library. The optimized size-bounded mode seems to be the overall winner, but for each strategy there are problems on which it performs best. Some attempt is made to analyze why. We emphasize that our optimization, and other implementation techniques like caching, are rather general: they are not dependent on the details of model elimination, or even that the search is concerned with theorem proving. As such, we believe that this study is a useful complement to research on extending the model elimination calculus.

1 Model elimination and PTTP

For some time after its proposal by Loveland (1968), model elimination was pushed to the background by the intense flurry of activity in resolution theorem proving. It was given a new lease of life by the work of Stickel. A natural procedural implementation of model elimination calculi, Loveland's MESON procedure in particular, is a modest change to Prolog's standard search strategy, viz. backward chaining on Horn clauses with unification and backtracking. We assume that the first order formula to be proved is negated and reduced to clausal form, so the task is to refute, i.e. prove falsity ($\perp$) from, an implicitly conjoined set of clauses, each one of the form:

$$P_1 \vee \ldots \vee P_n$$

Here each P_i is a *literal*, meaning either an atomic formula or the negation of one. Variables occurring in each clause are implicitly universally quantified. Now from each input clause of the above form, a set of $n+1$ pseudo-Horn clause rules called 'contrapositives' is created.[1] We will write '$-$' for a (syntactic) negating

[1] Recall that a Horn clause is a clause which contains at most one unnegated literal. Here we just single out each literal in turn to act as the head clause in a Prolog-style search, even if this literal and some or all of its antecedants in the clause are negative.

operation on literals; that is $-(\neg P)$ is P, whereas $-P$ is $\neg P$ for atomic P. First there are n rules of the form:

$$-P_1 \wedge \ldots \wedge -P_{i-1} \wedge -P_{i+1} \wedge \ldots \wedge -P_n \Rightarrow P_i$$

and then there is one more of the form:

$$-P_1 \wedge \ldots \wedge -P_n \Rightarrow \bot$$

We could emphasize the Prolog connection by writing `P :- P1, ..., Pn` instead of $P_1 \wedge \ldots \wedge P_n \Rightarrow P$ (and putting any variables in specific instances in upper case). Anyway, the idea is to use these rules in a Prolog-style backward proof of the goal $\bot$. Stickel (1988) developed a Prolog Technology Theorem Prover (PTTP) based on just a few basic changes to a standard Prolog implementation:

- Perform sound unification. Most Prolog implementations omit an occurs check, allowing for example $f(X)$ and X to be unified. This is, according to the logic programming folklore, necessary for efficiency reasons, or desirable in order to permit cyclic structures.
- At each stage, retain a list of the ancestor goals (i.e. those which have already been expanded on the path between $\bot$ and the current goal), and as well as the input rules, allow unification of the current goal with the negation of one of its ancestors. This gives an alternative way of solving a goal, instead of expanding it using one of the rules. The ancestor goal $-P$ can be seen as a rule with P as conclusion and no hypotheses, except that its variables are fixed relative to those in the goal and the other ancestors.
- Replace Prolog's unbounded depth-first search with some complete alternative. The choice of alternative is the main topic of this paper, but we might note now that even in the propositional case, Prolog's standard search strategy could lead to an infinite loop. We can simply check the ancestor list for repetition — to do so is a worthwhile optimization anyway. However in general this is not sufficient. For example, a rule of the form $P(f(x)) \Rightarrow P(x)$ leads to larger and larger goals of the form $P(f^n(x))$, without limit.

A set of clauses is contradictory iff there is a proof of $\bot$ by the usual Prolog backward chaining (with the above modifications), the appropriate variable instantiations being discovered by unification and backtracking. Soundness of the procedure is easy to prove. Plaisted (1990) proposes the following interpretation in terms of sequents. A goal P in the context of an ancestor list $P_1, \ldots, P_n$ can be seen as a sequent goal $-P_1, \ldots, -P_n \vdash P$. Now the rule for unification with the negation of an ancestor is evidently sound; it just amounts to $\Gamma, P \vdash P$. And if we are trying to prove $\Gamma \vdash P$ given a rule $P_1 \wedge \ldots \wedge P_n \Rightarrow P$, we can perform case analysis on P; if it is true we are finished, otherwise it is false, so we may add $-P$ to the list of assumptions, giving subgoals $\Gamma, -P \vdash P_i$ for each $1 \leq i \leq n$. Hence the process of adding ancestor goals to the list is also justified. Plaisted (1990) goes on to prove completeness, even given two refinements:

1. It is only necessary to use the second kind of rule (with $\bot$ as conclusion) for certain 'support' clauses C. Informally, these are the ones which contribute to the inconsistency of the original clause set S. More formally, it is only necessary to try C if there is an $S_1 \subseteq S$ such that S_1 is inconsistent yet $S_1 - \{C\}$ is consistent.[2] Of course this can't be decided in general, but what we *do* know is that any set of clauses where each clause contains a positive literal

[2] This was probably known to earlier workers, but was not made explicit.

is satisfiable (choose an interpretation which maps each predicate to *true*). So it is enough to try each all-negative clause as a support. Quite often there is only one such clause; for example when trying to derive an equation from the axioms of group theory, the only all-negative clause will be the negation of the desired equation. Hence the first subgoal will be the 'conclusion' of the desired theorem, making the search appealingly goal-directed.

2. The process of unifying with ancestors need only be tried for *negative* goals with *positive* ancestors. (Or vice versa, or based on various other semantically-based ways of splitting the literals in two, but this seems the most useful.) The proofs given this 'positive refinement' may occasionally be longer, but this is often more than repaid by the cutting back of the search space. Therefore it is not even necessary to add negative ancestors to the list; although it's still useful to store them to check for repetitions, they can otherwise be ignored.

2 An example

The following theorem was proposed by Łoś, as an example of a relatively simple purely logical assertion which is nevertheless not obvious.[3] It was introduced to the automated theorem proving community and used by Rudnicki (1987) as an example of an assertion which is indeed not obvious, in a certain technical sense of 'obvious'. It is now often referred to by some name such as `nonobv` in the theorem proving literature; it is problem `MSC006-1` in the TPTP Problem library (Suttner and Sutcliffe 1995).

$$\begin{array}{l}(\forall x\ y\ z.\ P(x,y) \land P(y,z) \Rightarrow P(x,z)) \land \\ (\forall x\ y\ z.\ Q(x,y) \land Q(y,z) \Rightarrow Q(x,z)) \land \\ (\forall x\ y.\ Q(x,y) \Rightarrow Q(y,x)) \land \\ (\forall x\ y.\ P(x,y) \lor Q(x,y)) \\ \Rightarrow (\forall x\ y.\ P(x,y)) \lor (\forall x\ y.\ Q(x,y))\end{array}$$

Translating the negation of this formula into clausal form, introducing Skolem constants a, b, c and d, we get the following clauses:

$$\begin{array}{r}\neg P(x,y) \lor \neg P(y,z) \lor P(x,z) \\ \neg Q(x,y) \lor \neg Q(y,z) \lor Q(x,z) \\ \neg Q(x,y) \lor Q(y,x) \\ P(x,y) \lor Q(x,y) \\ \neg P(a,b) \\ \neg Q(c,d)\end{array}$$

Now all the contrapositives are generated, yielding the following rules:

$$\begin{array}{rcll}P(y,z) \land \neg P(x,z) & \Rightarrow & \neg P(x,y) & (1) \\ P(x,y) \land \neg P(x,z) & \Rightarrow & \neg P(y,z) & (2) \\ P(x,y) \land P(y,z) & \Rightarrow & P(x,z) & (3) \\ Q(y,z) \land \neg Q(x,z) & \Rightarrow & \neg Q(x,y) & (4) \\ Q(x,y) \land \neg Q(x,z) & \Rightarrow & \neg Q(y,z) & (5) \\ Q(x,y) \land Q(y,z) & \Rightarrow & Q(x,z) & (6)\end{array}$$

[3] "You may say it is trivial yet you will not say it is nothing".

$$
\begin{aligned}
Q(x,y) &\Rightarrow Q(y,x) && (7)\\
\neg Q(y,x) &\Rightarrow \neg Q(x,y) && (8)\\
\neg P(x,y) &\Rightarrow Q(x,y) && (9)\\
\neg Q(x,y) &\Rightarrow P(x,y) && (10)\\
& \quad\ \neg P(a,b) && (11)\\
& \quad\ \neg Q(c,d) && (12)\\
P(a,b) &\Rightarrow \bot && (13)\\
Q(c,d) &\Rightarrow \bot && (14)
\end{aligned}
$$

Here is a MESON proof of $\bot$. Each step is marked either with its rule number or with 'A' indicating unification with the negation of an ancestor.

$$
\dfrac{\dfrac{
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg Q(c,d)}\,12}{P(c,d)}\,10 \quad \dfrac{}{\neg P(a,d)}\,A}{\neg P(a,c)}\,1}{Q(a,c)}\,9}{Q(c,a)}\,7 \quad \dfrac{}{\neg Q(c,d)}\,12}{\neg Q(a,d)}\,5}{P(a,d)}\,10
\quad
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg Q(c,d)}\,12}{\neg Q(d,c)}\,8}{P(d,c)}\,10 \quad \dfrac{}{\neg P(d,b)}\,A}{\neg P(c,b)}\,2}{Q(c,b)}\,9 \quad \dfrac{}{\neg Q(c,d)}\,12}{\neg Q(b,d)}\,5}{\neg Q(d,b)}\,8}{P(d,b)}\,10
}{P(a,b)}\,3}{\bot}\,13
$$

3 Search strategies

As we have already noted, the usual Prolog depth-first left-to-right search will go into an infinite loop on most nontrivial problems. In the Łoś example, the first subgoal $P(a,b)$ would result in application of the transitivity rule for P to give $P(a,X)$ and $P(X,b)$. Then $P(a,X)$ would get similarly expanded in its turn, and so ad infinitum. In order to achieve completeness, i.e. guarantee eventually finding a proof such as the one given above, it is necessary to force enumeration of all possible expansions so that each one is explored eventually. How is this to be done?

The most obvious way is to use breadth-first search. Each level in the space of proof trees gets fully explored before further expansion is attempted. Hence a proof of depth n will always get found at the n^{th} level of expansion.

A refinement of this idea is to use *best-first search*, as implemented by Paulson in the Isabelle theorem prover (Paulson 1994).[4] That is, based on some heuristic such as size and number of subgoals, a particular goal state is tried first, and all the possible subgoal states are inserted into a priority queue. This is then repeated until a solution is found. Since the strict enumeration of levels is no longer guaranteed, the danger of infinite looping is reintroduced. However this can easily be circumvented by appropriately including the number of ancestor goals in the heuristic.

We will give some results for best-first search later. However, breadth-first and best-first search have the substantial disadvantage that they require storage of all the goal states explored. In nontrivial examples this requires large amounts of

[4]Isabelle differs somewhat from most other theorem provers in the LCF family (Gordon, Milner, and Wadsworth 1979) in that its subgoaling mechanism, used for backward proof, allows multiple possible goal states with backtracking, and can perform Prolog-style unification (in fact higher order unification) of variables. Hence Prolog-like search methods are very easy to implement.

memory. Moreover, it moves the system further away from a conventional Prolog implementation. Instead, Stickel (1988) originally used *depth-first iterative deepening*. This means that depth-first search is performed in the usual Prolog style, but failing immediately if it ever gets beyond a certain depth; complete failure at a given depth bound results in the bound's being increased and the entire search attempted again. As Stickel remarks, this is very much like breadth-first search, except that intermediate levels of expansion are recalculated on demand rather than stored. At first sight this looks wasteful, but since the number of possible proofs tends to grow exponentially with the depth bound, the final level usually dominates and recalculation increases total computation by only a modest constant factor. Memory usage is practically nil and the implementation need only tweak a standard Prolog system to carry the depth bound as it expands goals.

Despite the talk of *depth*, Stickel's original implementation did not use a bound on the depth (height) of the proof tree, but rather on the number of nodes in the tree (= inferences in the proof). The blowup in the number of possibilities is often more graceful than it is with depth, and it avoids a bias towards highly symmetrical proof trees. Nevertheless, Letz, Schumann, Bayerl, and Bibel (1992) have discovered that actually using a depth bound seems to be better on average.[5] Let us see how these different methods perform on the Łoś example. We have implemented all the different search strategies, but tried to use the same code where possible. More implementation details are given below.

1. Best first search performs best of all; it takes just 1.0 seconds of user CPU time,[6] performs 1,378 successful unifications (we will follow the tradition in calling these 'inferences') and accumulates just 519 goal states in its priority queue. Against that, the proof it finds is rather larger than the above. The heuristic used was:

   $$SIZE = \Sigma_{g=1}^{subgoals} 10 size_g + |ancestors_g|$$

   that is, each goal is allocated a size which is the number of its ancestors plus 10 times the 'size' (roughly, the total number of function symbols and variables in the formula) of the main goal. These sizes are then added together.

2. Depth-bounded iterative deepening takes only slightly longer: 1.2 seconds of CPU. The proof (almost the one given above) is found at level 7, after 7,627 exploratory inferences.[7]

3. Inference-bounded iterative deepening performs very badly. It takes over half an hour (1,829.3 seconds) of CPU time to find the proof given above, and performs 5,360,067 inferences.

Why does inference-bounded search perform so badly? Well, most obviously because the proof is sufficiently long that by the time it is found, the search space has blown up too much. Depth-bounded search succeeded quickly because there is a relatively symmetrical proof which is therefore not too deep. The success of

[5]Of course, inference-bounded search will always find the 'shortest' proof, in a reasonable sense. The size of the proof is usually minuscule in comparison with the number of possibilities explored, so this has little relevance to the runtime. However our original interest in MESON was as a subsystem whose proofs would be translated into LCF inferences. From this point of view, short proofs are nice to have, though even then there is little difference in speed.

[6]All times given in this paper are user CPU times on a Sparc 4 with 48M of physical memory.

[7]The overhead of maintaining the priority queue and conducting more careful checks for ancestor repetition slow down the inference rate of best-first search; so many more 'inferences per second' are performed here.

best-first search is rather harder to understand; in some cases it performs well, in others badly.

So why is the proof so long? The model elimination calculus is rather weak in that it does not permit the multiple instantiation of lemmas. If two instances of $\phi[x]$ are used in the proof, then the proofs of $\phi[a_1]$ and $\phi[a_2]$ (say) must be given separately. This is often expressed by saying that the model elimination calculus is 'cut-free', since a characteristic of cut-free sequent proofs (or *normal* natural deduction proofs) is that a universal formula cannot be nontrivially deduced and then specialized. By contrast, resolution does allow multiple instantiations of lemmas.[8]

A careful look at the MESON proof above reveals that there are essentially two proofs of the fact $\neg Q(x, d)$; they are not precisely identical because the symmetry rules are applied in different places, but they could be made exactly identical except that one uses a for x, the other b. Indeed, a proof which a human would naturally find is the following. If $\forall x\ y.\ P(x, y)$, then we are finished. Otherwise there are a, b with $\neg P(a, b)$, and hence $Q(a, b)$. Now if $\forall x.\ Q(a, x)$, then by transitivity and symmetry of Q, we have $\forall x\ y.\ Q(x, y)$ as required. Otherwise there is a c with $\neg Q(a, c)$ and therefore $P(a, c)$. Now either (i) $P(c, b)$, so by transitivity $P(a, b)$; or (ii) $Q(c, b)$, so by symmetry and transitivity $Q(a, c)$; in either case we have a contradiction. The MESON analog requires a duplication to pass from $\forall z.\ Q(a, z)$ to $\forall x\ y.\ Q(x, y)$.

There are essentially two ways in which the model elimination procedure can be beefed up to allow multiple instantiation of lemmas.

1. The underlying calculus can itself be changed. For example, SETHEO researchers (Letz, Mayr, and Goller 1994) have recently been experimenting with incorporating so-called 'folding up' and 'folding down'. The present author is not really *au fait* with the technical details, but the results seem very promising.

2. The implementation technique can be altered to remember lemmas and avoid re-proving them. Such ideas have been explored by Astrachan and Stickel (1992), with promising results. The Łoś theorem is one of the best examples, where the use of lemmas cuts back runtimes by a factor of several hundred.

However, we will stick to the basic model elimination calculus, and try to understand how inference-bounded search can be improved. Such optimization might, of course, turn out to be all the better in conjunction with one or both of the above extensions.

4 A divide-and-conquer optimization

Suppose that when solving a goal g, we have used a rule which gives rise to two subgoals, g_1 and g_2, and that we have n inferences left. Now if we are to solve both g_1 and g_2 without overstepping the inference limit of n, we know that *one or the other of g_1 and g_2 must have a proof of size* $\leq n/2$ (where division truncates downwards if n is odd.) Now in typical proofs:

- the number of possible proofs increases exponentially with the inference bound; and

[8] Although its close relative, Maslov's inverse method (Lifschitz 1986), is motivated by and presented as forward search for a cut-free sequent proof, it nevertheless permits variables in any facts deduced to be treated as universal. In a sense, it performs meta-level proof search.

- most expansions do not result in a successful proof, even locally, let alone globally.

This suggests the following algorithm. First, attempt to solve g_1 with inference bound $n/2$. If that succeeds, then solve g_2 with whatever is now left over from n. If this fails (or the solution of the remaining subgoals fails under all the resulting instantiations), reverse the roles of g_1 and g_2 and try it that way round. Now exploration of g_1 and g_2 to the full depth is often avoided where it is clearly unnecessary. Against that, pairs of solutions to *both* g_1 and g_2 with size $\leq n/2$ will be found twice. (If the other subgoals cannot be solved with that instantiation and backtracking occurs, which will almost always happen.) One would expect, on average, that this is a small price to pay; this is emphatically borne out in the results below, though there are a few exceptional cases. What is important is that the remaining subgoals are not solved twice, since then the duplication could amplify exponentially across the proof tree. We will see below how to make sure of this.

The above generalizes easily to more subgoals, $g_1, \ldots, g_m$. One alternative would be to start by trying each in turn with depth bound n/m. However this implicitly leads to $m!$ different reorderings. It's probably better to recursively divide the goals into two approximately equal parts, and treat them as above (with g_1 and g_2 now standing for sets of subgoals). In this way, instead of examining all $m!$ permutations we 'only' get 2^{m-1}. [9] Even with small branching factors this is an improvement, and in the (admittedly unlikely) event of very large branching factors, a significant one.

Using this optimization, the Łoś example is handled much more easily: it runs in 5.6 seconds, and requires 25,613 inferences. To be sure, it is still worse than best-first and depth-first search, but we will see below that the opposite is more often true.

5 Redundancy in the proof skeleton

Anticipating later results, we will see that optimized inference-bounded search does better on average than depth-first search. Nevertheless, the margin is small enough to seem quite surprising at first sight. After all, the blowup of the search space with depth is usually dramatically exponential, so there's no real hope of finding proofs with even one longish branch. The conclusion might be that a large number of the TPTP theorems have rather symmetric, and therefore shallow, proofs. This does not imply that the *only* proofs are highly symmetric. On the contrary, there are many situations in mathematics where there is a great deal of redundancy in the proof skeleton.

Consider proofs in equational logic; there are a certain number of axioms together with reflexivity, symmetry and transitivity of equality, and congruence properties for all the function symbols involved. Most reasonably large proofs can be rearranged in myriad ways without greatly changing the proof size. (To say nothing of pointless detours like $a = b = a = b = a = b$.) First of all, transitivity chains can be implemented by various different associations of the one-step transitivity rule. Symmetry rules can be intermixed with transitivity rules in various different ways, e.g.

$$\frac{\dfrac{c = a}{a = c}\ SYM \qquad c = b}{a = b}\ TRANS$$

versus

[9] The recursion equations $C_1 = 1$, $C_{2k} = 2C_k^2$ and $C_{2k+1} = 2C_kC_{k+1}$ must have a unique solution and $C_k = 2^{k-1}$ is a solution.

$$\dfrac{\dfrac{\dfrac{c=b}{b=c}\,SYM \qquad c=a}{b=a}\,TRANS}{a=b}\,SYM$$

Moreover, instances of congruence rules can be floated up the proof tree past symmetry rules, and, at the cost of some duplication, past transitivity rules. One could easily arrive at a normal form theorem for equational proofs. For example, consistently floating symmetry and congruence rules up past transitivity rules, then symmetry rules past congruence rules, would yield something like: each equational theorem has a proof which is either immediate reflexivity or else a right-associated transitivity chain, each equation in which is derived by applying congruence rules (possibly zero times) to either an axiom or the reversal of an axiom. By analogy with Gentzen's cut-elimination theorem and its application in tableaux, even though such proofs might be longer (e.g. congruence rules are duplicated by floating them up past transitivity rules) the search space is dramatically reduced, making automated proof search easier. Actually, tableaux enforce a canonical order beyond what is already present in cut-free proof systems, since rule applications are always done in a fixed order (be it based on clever heuristics or simply on a round-robin basis).

The above gives one reason why throwing in equality axioms is a poor way of extending a proof procedure to deal with equality, a fact universally acknowledged in the ATP community. The axioms allow enormous redundancy, blowing up the search space dramatically. Unfortunately many of the more mathematical TPTP problems do just throw in the equality axioms like that. Even when they don't, it seems likely that there are similar redundancies lurking. A reduction to simple primitives always seems liable to lead to many ways of proving the same theorem. By the way, the above remarks suggest that a better set of equational axioms may be possible by introducing several semantically equivalent equality symbols. For example, the transitivity rule could be:

$$x =_1 y \wedge y = z \Rightarrow x = z$$

to force right-association. Some kind of weak 'normal form' could be enforced by such methods; compare the way operator precedences and associativities are encoded in programming language grammars by the introduction of extra nonterminals. Additional redundancy control mechanisms would still be desirable, though.

In any case, that's another story. The above was mainly meant to emphasize that (i) it isn't so surprising that inference-bounded search can perform badly, since there are still many redundancies involved, and (ii) it may be that there are both highly symmetric and highly skewed proofs of the same fact, and focusing the search on *either* of them may be an equally defensible policy, and better than allowing both. Accordingly, we experimented with biasing the proof search in the optimized inference-bounded case.

First, instead of forcing one half of the subgoals or the other to be solved with size $\leq n/2$, we force one to be solved with size $\leq n/3$, or $\leq n/4$ or $\leq n/5$. This of course means that not all proofs of size n will be found with the nominal size limit of n, but all will be solved at some larger value of n, while skewed proofs are encouraged. We will refer to the above as 'a skew of 3' (or 4, 5, etc.) This is a very simple modification of the divide-and-conquer refinement. To favour symmetric proofs, we simply insist that each of m subgoals is solved with size limit n/m. If one takes less than this, the difference is not made available to the others; if it fails then no alternative orders need to be tried.

6 Implementation

All versions of the algorithm were coded in CAML Special Light, a compilable version of the CAML language (Weis and Leroy 1993). This was because our original interest was in integrating the MESON procedure into an LCF-style interactive theorem prover (Gordon, Milner, and Wadsworth 1979; Paulson 1987; Gordon and Melham 1993) which is written in CAML Light; compare the work of Tarver (1990). However the CAML language stands on its own merits: it's perfectly suited for this kind of work, providing garbage-collected recursive data structures, uniform support for higher order functions, and a range of imperative features if required. Actually the programs are mostly applicative. In particular, instantiation is done by maintaining instantiation lists, not by destructive assignment. Apart from the flags controlling options and the incrementing of the inference counter, the only uses of imperative features are the arrays used to implement priority queues in the best-first case, and the caching of continuations in the others (see below). It would probably be possible to achieve a significant improvement in performance by recoding in carefully optimized C, but at the cost of a much less clear and elegant implementation. It was not our objective to compete with leading-edge theorem proving systems.

First of all, the input clauses are reordered to put the smaller ones, i.e. those with fewer literals, first. Hence the rules which generate smaller numbers of subgoals will be tried first. This seems a reasonable policy in general, though it actually makes the runtimes longer for the Łoś example. Moreover, the literals within each rule are reordered to put those with fewest free variables first (this being equal, those with the fewest variables which are not free in the conclusion). The idea is that these subgoals permit fewer choices of rules with which to unify, and so will lead to fruitless exploration being abandoned more quickly. Observe however that our divide-and-conquer optimization tends to negate the effect of this second kind of reordering, since it often leads to the solution of the subgoals in different orders. Note that, in contrast to the original PTTP and many of its descendants, the rules are *not* compiled into code, but are interpreted at runtime.

The best-first version is somewhat different from the others. It maintains a heap-based priority queue of the current goal states, each goal state being a list of goal-ancestor pairs. At each iteration, the front of the queue is taken off; if it is empty then the goal is solved, otherwise its head goal is expanded using each possible rule; the possible subgoals generated are appended to the tail of the current subgoals, and each such new subgoal state is inserted into the queue, its priority having been computed.

The other versions are implemented using a continuation-passing style. Rather than retain a list of 'additional goals to solve', a continuation is passed which will attempt to solve any additional goals under the instantiation given it as one of its arguments (other arguments include the total size left for the solution). This gives the program a very simple control structure, since backtracking can be initiated when the continuation fails. It also allows greater flexibility. For example, to generate multiple possible solution instantiations for a goal containing variables, one could pass a toplevel continuation which stores the solution then fails; this would initiate backtracking. Such flexibility is convenient for the divide-and-conquer optimization, though of course it could be implemented in other ways.

Variables in rules which are free in some or all of the hypotheses but not in the conclusion do not become instantiated when the rule is applied, and must be replaced with fresh variables to be later candidates for instantiation. This is done implicitly during unification: variables are all given numbers, and those less than a certain value only appear in the rules. Consequently these can be bumped up by the currently active 'offset' value when they are encountered during unification.

This offset value is also passed along through the chain of continuations, so that it only gets incremented once per successful proof step.

Continuations also allow a simple form of caching. First, note that if we are solving two goals g_1 and g_2, and a solution of g_1 is found *which does not produce any new instantiations*, then if the solution of g_2 fails, no further solutions to g_1 need be tried, because these could only achieve *greater* instantiation of g_2 and so there is no chance of its succeeding. This gives a simple search optimization; however, as pointed out by Stickel, it can interact badly with inference-bounded iterative deepening: the point is that an alternative solution to g_1 might use fewer inferences, and hence g_2 could succeed even under the same or a more special instantiation because there are more inferences available. Stickel restricted this optimization to cases where the solution of g_1 took just one inference (i.e. was solved by a rule with no hypotheses or by the negation of an ancestor), though an option was experimented with to treat non-instantiating solutions as if they were of zero size.

We generalize this as follows: each continuation has a cache or 'memo' (Michie 1968), so that it remembers the arguments it has already seen and failed on. Now if the continuation is called with an instantiation θ and size n, and there was already a call with instantiation θ and size $m \geq n$, the call fails at once (the sizes are ignored if depth-bounded search is used). This seems to be quite a useful optimization in practice, as indicated by some results below. We could adopt the refinement of also failing with an instantiation θ' which is no more general than θ. That would require more costly checks, but these might get repaid by eliminating many more redundant searches. Such a facility has been incorporated into SETHEO; these remembered instantiations are there called 'anti-lemmata'. We prefer the more neutral term 'caching' because the name emphasizes that it isn't tied to theorem proving in particular, but is applicable to Prolog-style search generally.[10] However the term 'anti-lemma' usefully emphasizes that the cache is only used in a negative way: "this has already been tried, and it didn't work" rather than "this has already succeeded and here are the instantiations it gave rise to that time". This more general kind of caching, where moreover the cache is more persistent (ours exists only ephemerally with the continuation), has been explored by Astrachan and Stickel (1992).

Caching is enough to deal with the problem which we have already mentioned in the optimized version: if a given pair of solutions is repeated in different orders, it will generate the same instantiations, and caching will prevent the continuation's body from being called twice. However since setwise comparison of instantiation lists is expensive[11] we pre-empt this filtering out by an additional wrapper which makes the continuation fail immediately if the second goal is solved with size $\leq n/2$ and then the first one is too (the other way round is permitted).

It is never necessary to repeat a goal, so if a goal ever appears in its own ancestor list, search of that branch can be abandoned immediately. This optimization was already included by Stickel. Of course, it may only happen that a repetition occurs after additional instantiations are performed. For example, applying a transitivity rule to $P(a,b)$ gives $P(a,X)$ and $P(X,b)$ as subgoals, but if a later instantiation sets X to a or b, there is a repetition. For simplicity, we only check when adding a goal as an ancestor that it is not already in the list, under the currently pending instantiations. The best-first version does a complete check of the ancestor list each time; this is rather costly, but otherwise that version can get stuck on rather long dead ends. In the other versions, such additional checks reduce the number of inferences, but usually still increase runtimes.

[10] Though of course from one point of view, that is theorem proving too. Some related techniques for avoiding redundant search are common in Prolog implementations.

[11] To reduce consing, instantiation lists are augmented from the head, rather than being maintained in a canonical order.

In some situations, it is impossible for a small increment in the inference bound to result in any new proofs. For example, if all the rules either have no hypotheses or 2, then it is impossible to generate new proofs by an increment of 1. As noted by Stickel (1988), it is possible to record, each time a proof fails because it overflows the size limit, the amount by which it overspilled; the least such value can then be chosen as the next increment. We did not implement this optimization as Stickel did since there seemed to be rather few situations where it is useful.

7 Results

Thanks to the TPTP problem library (Suttner and Sutcliffe 1995), there are well over 2,000 problems in clausal form available as a test suite. We decided to use this, rather than some smaller collection, in the hope that it would exercise the different search strategies on a representatively wide range of problems. Version 1.2.0 of the TPTP library contains 2,755 fixed-size problems; there are 'generators' to produce different sizes of generic ones, but we just took the single instance which is also provided. Of these problems, our programs immediately detected that 4 were satisfiable, since there were no all-negative support clauses available to start the search (recall that the rules with falsity as conclusion need only be tried for all-negative clauses).[12] By the way, just about half the problems (1,378) have precisely one possible support clause; most of the rest have just a few. There are 20 problems which have more than 10, the record-holder being `PUZ028-3` with 432 supports.

Given such a large set of problems, it was necessary to set quite low limits on CPU time in order to complete the tests in a reasonable time. We split the tests over 20 Sparc-4 workstations, making some effort to compensate for small differences in hardware characteristics. Each problem was allocated a CPU time limit of 300 seconds (5 minutes) and a (virtual) store limit of 12 megabytes (the latter is only really relevant for the best-first implementation). Successful problems were rerun on a fixed machine to make the CPU times completely normalized. Of course it's possible that a few problems took just over 5 minutes on the first machine even though they would have taken less on the second. We believe that such cases are rare, as is the converse situation of final runtimes exceeding 5 minutes; however the attentive reader may notice below a few instances of the latter phenomenon.

7.1 Comparison of search strategies

First, we will give some statistics for the general success of each search strategy. We will just make explicit what the names stand for:

- best: best-first search with the heuristic given above.
- sym: inference-bounded search where each of the subgoals is given an equal share of the inference limit and leftovers are not made available to the others.
- deep: depth-bounded iterative deepening.
- unopt: unoptimized inference-bounded search.
- opt: optimized inference-bounded search.
- skew-3: optimized inference-bounded search with a skew of 3.
- skew-4: optimized inference-bounded search with a skew of 4.

[12] There are probably plenty of others which are in fact satisfiable; the TPTP library is intended to contain a few such, as well as some whose status is open, such as the "Robbins conjecture".

- skew-5: optimized inference-bounded search with a skew of 5.

We give the number of TPTP problems solved in the 5 minute time and 12M space limits, the number which *only* that strategy solved, and the number on which that strategy was at least equal fastest. With some strategy or other, 976 problems were solved; 495 problems were solved by every strategy. As already stated, this is from a total stock of 2,755 problems, 4 of which were flagged as satisfiable.

Strategy	Successes	Only	Fastest
best	593	22	219
sym	768	1	332
deep	816	43	397
unopt	717	0	383
opt	850	1	330
skew-3	873	2	435
skew-4	871	0	453
skew-5	869	7	480

From this table it is apparent that most modes have at least a few problems on which they shine, though it appears that the divide-and-conquer optimization with a skew of 3 is best on average. But the large number of different strategies, and the many variants of our optimized version, tends to obscure individual comparisons, so let's also look at some interesting pairs, on the same basis. First, it is evident that the optimization is a clear improvement.

Strategy	Successes	Only	Fastest
unopt	717	3	618
opt	850	136	827

It gets over a hundred more problems, while only 3 become impossible. Almost all runtimes are significantly reduced by the optimization, often by one or two orders of magnitude (see the large table of results below). We tried the three rogues without resource limits with the divide-and-conquer optimization to see just how much worse they became. Problem `LCL097-1` took 12:32.7 minutes 'optimized' against 2:59.3 minutes 'unoptimized'; and `LCL107-1` took 8:21.2 against 2:16.9. The most dramatic was `NUM283-1.005`, which took 46:54.8 instead of just 0:41.4 without the divide-and-conquer optimization. We examined this theorem, and discovered that it is rather atypical: it essentially calculates 5! in unary arithmetic based on recursive definitions of the arithmetic operations. As such the proof is (a) very long (150 inferences) and (b) completely deterministic. So it's not too surprising that the optimization performs badly, since it explores alternative ways of making choices even where no choice exists. Most of the few instances where the optimization slows things down are also concentrated in the `LCL100` area. These problems involve discovering proofs from complicated single axioms in equivalential calculus. The only rules are the axioms and the rule of 'condensed detachment' (from $\vdash a \equiv b$ and $\vdash a$, deduce $\vdash b$). Once instantiation has made a goal sufficiently specialized that it cannot be an instance of an axiom, failure is quick. Though not completely deterministic as `NUM283-1.005` is, these are clearly not good candidates for the optimization.

Our experience echoes that of SETHEO researchers (Letz, Schumann, Bayerl, and Bibel 1992), in that depth-bounded search seems to do better on average than *unoptimized* inference-bounded search, though where inference bounded search succeeds it is often faster:

Strategy	Successes	Only	Fastest
unopt	717	54	680
deep	816	153	634

However our optimized version works out significantly better than depth-bounded search:

Strategy	Successes	Only	Fastest
deep	816	70	571
opt	850	104	709

Slightly surprisingly, notwithstanding the justification above, our optimization in conjunction with a slight skew (3) seems even better:

Strategy	Successes	Only	Fastest
deep	816	60	510
skew-3	873	117	749

We will now give a fairly comprehensive list of runtimes for different strategies on different problems. To save space and focus on the more interesting cases, we include only those where at least one strategy took over a minute (which includes exceeding the 5 minute limit), but at least one succeeded. That is, we exclude problems which are either very easy or too difficult.

PROBLEM	Best	Sym	Deep	Unopt	Opt	Skew-3	Skew-4	Skew-5
BOO003-1 [B2 part 1]				2:20.0	5.5	4.4	3.6	3.6
BOO003-2 [prob2_part1.ver2.in]								3:52.2
BOO003-4 [TA]			4:38.6		2:25.7	1:15.6	57.1	39.8
BOO004-1 [B2 part 2]			3.2	2:24.5	5.4	4.5	3.6	3.8
BOO004-2 [prob2_part2.ver2.in]								5:04.6
BOO004-4 [TA]					2:09.0	1:04.5	45.0	33.2
BOO005-1 [B3 part 1]			5.9	2:36.3	9.9	8.1	6.6	7.0
BOO006-1 [B3 part 2]				2:32.6	9.3	7.4	6.1	6.6
BOO012-1 [B8]		2.2	3.8		1:25.2	1:03.2	50.2	44.7
BOO012-3 [B8]		46.6	1:01.5					
BOO013-1 [B9]							3:58.4	3:03.6
CAT001-3 [C1]		12.7	22.5	32.0	7.1	5.2	4.7	3.5
CAT001-4 [C1]		6.3	8.6	7.3	4.9	3.7	3.4	2.5
CAT002-3 [C2]			34.4	5.8	3.1	2.4	2.2	1.7
CAT002-4 [C2]		3:06.0	15.6	2.4	2.3	1.8	1.6	1.3
CAT003-3 [C3]		9.4	9.9	0.7	0.5	0.4	0.4	0.4
CAT003-4 [C3]		4.1	3.6	0.3	0.4	0.3	0.3	0.3
CAT004-3 [C4]					39.5	26.9	22.7	1:33.7
CAT004-4 [C4]				1:13.8	27.5	18.5	14.1	53.5
CAT005-1 [C5]		1.9	9.5		6.1	1:11.7		
CAT006-1 [C6]		1.8	6.4		6.1	1:09.7		
CAT008-1 [C8]					4:20.5			
CAT017-4	18.7	0.3		0.3	0.2	0.2	0.2	0.2
CAT018-1 [p18.ver1.in]	11.5				16.8	10.2	17.4	16.6
CAT019-3 [p15.ver3.no2.in]			4.1					
COL001-1 [C1]			11.2			4:34.7	2:44.1	2:21.0
COL001-2 [C1]		2.2	0.5	1.6	1.3	1.0	1.0	1.0
COL002-1 [C1.1]		1.1	0.1	0.2	0.2	0.2	0.2	0.2
COL002-2 [C1]		0.2	0.4	0.2	0.2	0.2	0.2	0.1
COL002-3 [C1]		0.2	0.0	0.0	0.1	0.1	0.1	0.1
COL008-1 [Question 13]		0.0	0.0	0.0	0.0	0.0	0.0	0.0
COL009-1		0.8	0.1	0.6	0.5	0.5	0.4	0.5
COL010-1		0.0	0.0	0.0	0.0	0.0	0.0	0.0
COL011-1								4:52.5
COL015-1		0.0	0.0	0.0	0.0	0.0	0.0	0.0
COL017-1		0.0	0.0	0.0	0.0	0.0	0.0	0.0
COL019-1		0.9	0.0	0.2	0.1	0.2	0.2	0.1
COL020-1		0.8	0.1	0.2	0.1	0.1	0.2	0.2
COL021-1		0.0	0.0	0.0	0.0	0.0	0.0	0.0
COL022-1		0.0	0.0	0.0	0.0	0.0	0.0	0.0
COL023-1		36.9	1.2	0.6	0.5	0.4	0.4	0.3
COL024-1		0.0	0.0	0.0	0.0	0.0	0.0	0.0
COL025-1 [stage1.in & stage2.in]		0.0	0.0	0.0	0.0	0.0	0.0	0.0
COL026-1		36.6	1.1	0.5	0.4	0.4	0.4	0.3
COL027-1		38.8	1.2	0.6	0.5	0.4	0.4	0.4
COL028-1		39.1	1.2	0.6	0.5	0.4	0.4	0.4
COL030-1		0.4	6.8	0.8	0.3	0.3	0.2	0.2
COL032-1		14.6	4.7	48.8	2.8	1.6	1.1	1.0
COL033-1					1:48.4	50.6	27.3	21.1
COL035-1		2.7	4.0	56.3	5.8	3.1	2.6	2.5

PROBLEM	Best	Sym	Deep	Unopt	Opt	Skew-3	Skew-4	Skew-5
COL039-1		2.2	1.9	0.6	0.4	0.3	0.3	0.3
COL040-1 [Question 5]			4:45.2		3:07.2	1:08.5	41.7	27.4
COL041-1						3:26.0	2:09.0	1:16.7
COL044-1 [CL3]			4:58.6		3:07.2	1:09.8	39.2	27.7
COL045-1		0.0	0.0	0.0	0.0	0.0	0.0	0.0
COL048-1		0.0	0.0	0.0	0.0	0.0	0.0	0.0
COL050-1 [bird1.ver1.in]		0.0	0.0	0.0	0.0	0.0	0.0	0.0
COL051-1 [bird2.ver1.in]		0.4	0.1	0.1	0.1	0.0	0.1	0.1
COL052-1 [bird4.ver1.in]		32.5	2.1		5.4	1.7	1.1	0.8
COL052-2 [bird4.ver2.in]		2:28.6	3.0		2.6	1.0	0.7	0.6
COL054-1 [bird6.ver1.in]		0.3	0.1	0.4	0.3	0.2	0.2	0.3
COL056-1 [bird8.ver1.in]		2:20.9	0.6	0.9	0.7	0.4	0.4	0.4
COL057-1 [CL5]								5:00.9
COL058-2		2:24.0	36.6			1:52.5	47.9	24.1
COL060-1 [CL-1]						3:56.6	1:44.8	1:10.1
COL060-2 [CL-1]	0.5	1:31.5				2:45.0	1:34.3	1:09.8
COL060-3 [CL-1]	0.5	1:23.6	4:38.8		1:56.6	1:02.0	35.9	25.2
COL061-1 [CL-2]								3:28.0
COL061-2 [CL-2]	0.5						5:03.5	2:55.0
COL061-3 [CL-2]	0.5		4:32.4			3:06.9	1:40.5	1:12.1
COL062-2 [CL-3]	0.8							
COL062-3 [CL-3]	0.7							
COL063-2 [CL-4]	0.7							
COL063-3 [CL-4]	0.7							
COL063-4 [CL-4]	0.9							
COL063-5 [CL-4]	0.9							
COL063-6 [CL-4]	0.7							
COL064-10 [CL-5]	1.3							
COL064-11 [CL-5]	1.0							
COL064-2 [CL-5]	1.0							
COL064-3 [CL-5]	0.9							
COL064-4 [CL-5]	1.2							
COL064-5 [CL-5]	1.2							
COL064-6 [CL-5]	0.9							
COL064-7 [CL-5]	1.0							
COL064-8 [CL-5]	1.0							
COL064-9 [CL-5]	1.4							
COL066-2 [CL-7]							4:50.3	3:19.7
COL066-3 [CL-7]							4:43.9	3:20.5
COL070-1 [Question 11]		38.2	0.7	0.5	0.4	0.3	0.3	0.3
COL075-1			2:30.4					
COL075-2		10.2	10.5		23.1	12.3	8.3	18.7
COM004-1			0.2		20.1	6.0	2.8	1.0
GEO001-1 [T1]		1:40.0						
GEO003-1 [T3]	1.3	0.5		0.4	0.4	0.4	0.4	0.3
GEO003-2 [T3]	1.1	0.4	3:10.7	0.4	0.3	0.3	0.3	0.3
GEO011-2 [T11]		0.3	0.4	0.3	0.3	0.3	0.3	0.3
GEO011-3 [T11]		1.4	1.1	1.4	1.4	1.4	1.5	1.5
GEO011-4 [T11]		0.3	0.3	0.3	0.3	0.3	0.3	0.3
GEO017-2 [D4.1]		1:13.6		32.6	0.9	0.8	0.8	0.8
GEO018-2 [D4.2]		0.5	9.0	0.4	0.4	0.3	0.3	0.4
GEO020-2 [D4.4]	1.6	1:16.0		32.4	0.9	0.7	0.7	0.7
GEO022-2 [D5]		1:16.3		32.6	1.0	0.8	0.8	0.8
GEO022-3 [D5]		1.7	17.2	0.5	0.6	0.6	0.6	0.6
GEO024-2 [D7]	1.1	0.5	3:14.5	0.4	0.4	0.3	0.4	0.3
GEO026-2 [D9]			59.6					
GEO027-3 [D10.1]		8.1		2:18.8	14.4	11.1	10.8	10.8
GEO039-2 [B1]						29.6	20.4	13.1
GEO039-3 [B1]				2:13.7	2:11.3	2:11.7	2:07.6	2:10.4
GEO040-2 [B2]						42.7		
GEO041-3 [B3]		22.2		29.3	30.6			
GEO047-3 [B9]		1:34.7		2:33.6	2:33.6			
GEO055-3 [R2.2]		2.1	5.2	0.7	0.6	0.7	0.7	0.7
GEO056-2 [R3.1]	1.2	1.8			2.1	1.4	1.4	1.5
GEO058-2 [R4]		1.6			8.4	6.0	3.0	2.8
GEO058-3 [R4]	1:08.1	0.8	-	1.1	1.0	0.9	1.0	1.0
GEO059-3 [R5]		4.7	1:06.4	4.2	4.2	4.1	3.9	4.1
GEO064-3 [C2.1]	1.6	1.3		1.3	1.3	1.3	1.3	1.3
GEO065-3 [C2.2]	1.6	1.3		1.2	1.2	1.3	1.3	1.3
GEO066-3 [C2.3]	1.5	1.3		1.2	1.3	1.2	1.2	1.2
GRP001-1 [wos10]					9.2	5.7	5.1	4.9
GRP003-2		0.7	1.6	2.6	0.8	0.6	0.5	0.5
GRP004-2		0.5	0.9	0.1	0.1	0.1	0.0	0.1
GRP006-1 [EX6]		0.0	0.0	0.0	0.0	0.0	0.0	0.0
GRP008-1 [wos4]	17.1	2.4	4:01.3		5.1	16.4	2:17.6	
GRP009-1 [wos6]		0.6	1.4	8.2	2.7	2.3	2.2	2.2
GRP010-1 [wos7]		1.5	0.7	0.4	0.3	0.2	0.2	0.2
GRP012-1 [wos9]		8.1	18.4	0.7	0.6	0.5	0.5	0.5
GRP012-2 [ls36]					2:11.5	1:16.4	1:07.2	1:00.1
GRP012-3					2:25.7	1:30.5	1:15.6	1:08.7
GRP013-1 [wos11]		51.1	1:13.4	15.5	10.3	8.9	8.2	7.0
GRP022-1 [wos8]		0.2	0.1	0.1	0.1	0.1	0.1	0.1
GRP022-2 [Established lemma]			49.8		43.8	24.5	17.3	11.7
GRP025-1 [G8]		2.0	0.6		20.5	2.6	6.0	7.2
GRP025-2 [G8]		9.6	1.5	3.6	2.1	2.1	2.0	2.1
GRP025-3 [G8]		1.3	0.6		17.9	2.4	5.8	6.7
GRP025-4 [G8]		26.4	1.8	7.9	4.6	4.6	4.7	4.7
GRP026-1 [G9]		3.1	0.7		33.8	4.0	8.3	10.5
GRP026-2 [G9]		35.3	3.6	14.5	3.9	4.1	3.8	3.7
GRP026-3 [G9]		1.9	0.6		29.4	4.0	8.5	9.9
GRP026-4 [G9]		1:39.6	4.2	36.3	9.0	8.5	8.5	8.9

PROBLEM	Best	Sym	Deep	Unopt	Opt	Skew-3	Skew-4	Skew-5
GRP027-1		0.2	0.2	0.2	0.2	0.2	0.2	0.2
GRP027-2 [cyclic.ver3.in]		0.1	0.2	0.2	0.1	0.2	0.1	0.2
GRP029-1 [wos1]		1:46.3	8.3		6.8	2.3	1.7	1.7
GRP029-2 [G5]		23.5	7.9		5.5	2.2	1.7	1.7
GRP030-1 [wos2]		1.4	2.9	1:15.9	2.5	1.4	1.1	1.1
GRP031-1 [wos5]		21.3	6.3	1.8	0.3	0.3	0.3	0.3
GRP031-2 [ls23]		0.1	0.2	0.1	0.1	0.1	0.1	0.1
GRP034-3 [wos14]		0.2	0.2	0.1	0.1	0.1	0.1	0.0
GRP036-3 [wos16]	5.4	3.2		0.7	0.8	0.5	0.5	0.5
GRP037-3 [wos17]		35.3		12.3	6.7	5.3	5.0	5.3
GRP046-2		1.8	0.7	0.1	0.1	0.1	0.1	0.1
GRP047-2	0.3	2.4		4.2	2.3	1.4	1.4	1.3
GRP123-1.003 [Bennett QG1]		1:28.8	3.6		1:06.1	51.5	1:03.7	1:11.1
GRP123-2.003		1:36.2	4.0		1:09.1	55.0	1:06.6	1:13.0
GRP123-3.003		1:41.3	5.6		1:16.3	57.8	1:05.3	1:15.8
GRP123-4.003		2:24.7	7.0		1:28.2	58.6	1:07.2	1:19.1
GRP123-6.003 [QG1a]		2:10.3	14.5		2:30.6	1:46.1	2:07.0	2:25.1
GRP123-7.003		2:19.3	15.0		2:31.0	1:52.2	2:09.7	2:29.1
GRP123-8.003		2:23.7	16.8		2:33.1	1:50.9	2:10.7	2:31.6
GRP123-9.003		2:19.2	14.2		2:30.0	1:48.9	2:08.2	2:19.7
GRP124-1.003 [Bennett QG2]		1:34.1	3.6		1:06.1	53.6	1:02.1	1:06.6
GRP124-2.003		1:38.0	4.0		1:11.0	56.4	1:05.8	1:14.4
GRP124-3.003		1:42.9	5.5		1:13.8	59.0	1:07.8	1:12.8
GRP124-4.003		2:26.2	6.9		1:22.6	59.1	1:08.7	1:13.3
GRP124-6.003 [QG2a]		2:18.5	14.5		2:28.2	1:48.4	2:05.9	2:23.9
GRP124-7.003		2:19.0	13.8		2:28.3	1:46.6	2:04.2	2:28.6
GRP124-8.003		2:20.7	16.0		2:31.6	1:53.2	2:10.9	2:19.8
GRP125-1.003 [Bennett QG3]		1:35.8	7.6		2:20.9	1:50.6	1:55.7	2:01.8
GRP125-2.003		1:35.8	9.6		2:36.9	1:56.0	2:04.6	2:13.0
GRP125-3.003		1:50.7	13.6		2:39.5	1:59.7	2:08.7	2:18.5
GRP125-4.003			40.4				3:39.2	
GRP127-1.003 [Bennett QG5]		1:32.8	8.9		2:13.8	1:38.5	1:48.6	2:11.9
GRP127-2.003		1:41.0	12.1		2:27.1	1:45.0	1:57.5	2:12.2
GRP127-3.003		1:50.2	18.3		2:30.8	1:47.5	2:01.2	2:17.1
GRP127-4.003			41.4					
GRP129-1.002 [Bennett QG7]		1:08.7	5.5		4:24.8	2:58.3	2:38.9	2:45.3
GRP129-2.002		1:15.8	6.9			3:20.6	2:58.6	3:14.4
GRP129-3.002		1:33.3	10.0			4:38.1	4:21.0	4:47.0
GRP129-4.002	10.6		41.1		44.8	7.4	6.5	8.8
GRP130-1.002 [Bennett QG8]	10.3	1:22.4	5.6		37.9	29.5	31.2	36.7
GRP130-2.002	10.0	1:33.4	7.4		44.0	33.6	35.7	42.4
GRP130-3.002	16.1	1:52.7	10.5		59.0	50.8	54.0	1:05.9
GRP130-4.002	7.5	13.5	3.6		53.9	5.8	7.3	8.7
GRP131-1.002 [QG1-ni]			46.4					
GRP131-2.002			1:06.1					
GRP132-1.002 [QG2-ni]			43.3					
GRP132-2.002			1:02.5					
GRP133-1.002 [QG3-ni]			9.2		4:07.9	2:53.8	2:33.1	2:40.6
GRP133-2.002			17.6		4:33.6	3:08.8	2:45.1	2:56.9
GRP134-1.002 [QG4-ni]		1:14.0	2.9			3:47.4	3:10.0	3:18.5
GRP134-2.002		1:29.8	4.9			4:13.0	3:44.5	3:42.1
GRP135-1.002 [QG5-ni]	14.6	2:09.3	4.8		38.6	32.9	34.3	40.3
GRP135-2.002	13.1	2:20.4	6.9		43.5	37.5	38.7	46.4
GRP139-1 [ax_glb1b]		3.6	0.5	0.9	1.1	1.0	0.8	1.5
GRP140-1 [ax_glb1c]		2:22.0			4:54.9	2:32.2	1:53.9	1:26.2
GRP143-1 [ax_glb2b]		0.3	0.1	0.2	0.4	0.3	0.2	0.3
GRP145-1 [ax_glb3b]		0.1	0.1	0.1	0.2	0.2	0.2	0.2
GRP146-1 [ax_lub1a]		3.5	0.5	1.0	1.1	0.9	0.8	1.6
GRP148-1 [ax_lub1c]		2:09.4				2:43.4	2:04.4	1:34.7
GRP150-1 [ax_lub2a]		0.1	0.1	0.1	0.1	0.1	0.1	0.1
GRP152-1 [ax_lub3a]		0.4	0.3	1.0	1.2	2.5	2.2	1.9
GRP156-1 [ax_mono1c]	0.4	8.1		1:31.4	11.6	7.6	6.3	4.9
GRP159-1 [ax_mono2c]	4.9	4:47.5						
GRP162-1 [ax_transa]		39.2			1:48.5	55.2	1:19.0	1:05.1
GRP163-1 [ax_transb]		32.3			3:21.4	1:48.0	1:21.7	1:04.5
GRP165-1 [lat1a]		14.9	25.1		2:22.0	1:12.2	51.3	38.5
GRP166-3 [lat3a]		15.5	24.7		2:24.4	1:15.6	50.1	38.3
GRP167-5		0.1	0.2	0.2	0.2	0.2	0.2	0.3
GRP168-1 [p01a]	0.2	33.8		3:23.2	35.2	21.9	18.3	16.4
GRP168-2 [p01b]	0.2	35.1		3:22.6	34.6	22.1	18.5	16.8
GRP186-4 [p23x]	2.1	2:31.1					3:57.5	3:06.6
GRP188-1		0.4	0.3	1.0	1.3	2.5	2.2	1.9
GRP188-2 [p38a]		0.5	0.3	1.2	1.5	3.0	2.7	2.2
HEN003-1 [H3]						2:10.1	1:55.5	2:09.1
HEN003-2 [H3]						4:38.7	2:34.1	1:44.9
HEN003-3 [HP3]					10.3	4.9	3.6	3.0
HEN003-4 [H3]				5:01.4	19.1	8.6	5.9	4.9
HEN003-5 [H3]					20.5	9.5	6.8	5.6
HEN004-2 [H4]								4:18.0
HEN004-4 [H4]					41.7	18.8	12.7	9.7
HEN004-5 [H4]							4:33.2	3:01.2
HEN005-2 [H5]					1:44.9	2:14.3	3:47.7	
HEN005-3 [HP5]						4:23.5	2:21.4	1:40.3
HEN005-6 [H5]							4:40.3	3:12.9
HEN006-4 [H6]				40.8	5.6	4.0	3.2	3.0
HEN007-2 [H7]				3:51.7	26.7	19.3	15.7	15.3
HEN007-4 [H7]		0.2	2.8	0.2	0.2	0.2	0.2	0.2
HEN007-6 [H7]			53.0	3:21.6	20.6	14.7	11.2	11.6
HEN008-1 [H8]			1:09.1		18.7	17.0	20.6	28.9
HEN008-2 [H8]					36.0	25.2	20.8	19.1
HEN008-3 [HP8]				52.5	4.7	2.8	2.2	1.7

PROBLEM	Best	Sym	Deep	Unopt	Opt	Skew-3	Skew-4	Skew-5
HEN008-4 [H8]		2:06.1	4.6	15.2	9.4	6.2	5.2	4.4
HEN008-5 [H8]				53.7	18.7	9.7	7.3	5.8
HEN008-6 [H8]				34.3	6.6	4.1	3.1	2.7
HEN009-2 [H9]					1:25.6	1:15.5		
HEN009-4 [H9]		0.4	7.3	2.7	0.7	0.6	1.8	7.2
HEN009-5 [H9]		7.1			12.0	20.9		
HEN010-4 [H10]			2:22.2	1:38.8	16.1	10.0	8.5	6.3
HEN010-6 [H10]			1:02.3	1:21.3	9.6	5.7	4.6	3.5
HEN011-4 [H11]		1:32.5	1:41.9			4:42.4		
HEN012-1						2:53.1	2:22.8	2:40.9
HEN012-3 [new.ver2.in]					12.4	6.1	4.5	3.4
LCL006-1 [EC-69]		3:57.3	3:18.7					
LCL008-1 [EC-71]		2:57.5	6.7	0.5	0.6	0.4	0.3	0.3
LCL009-1 [EC-72]			5:02.0					
LCL010-1 [EC-73]		3:00.5	6.9	20.6	13.5	30.1	49.2	1:56.7
LCL022-1 [ec.in part 1]		4:41.2	1:19.8					
LCL023-1 [ec.in part 2]			4:50.7					
LCL033-1 [C0-45]	5.6			4.2	4.6	2.4	1.5	0.8
LCL035-1 [C0-47]		1:12.5	2.8	1.1	0.9	0.5	0.7	1.2
LCL045-1 [CN-6]		51.0	49.8		2:43.3	4:05.6	2:46.9	
LCL064-2 [morgan.six.ver1.in]				3:11.5	2:22.3	1:25.3	1:06.9	50.3
LCL066-1 [CN-27]								3:51.4
LCL076-3 [morgan.four.ver1.in]				1:43.7	48.3	36.4	26.9	28.4
LCL077-2 [morgan.two.ver1.in]		1:36.3	2:06.7	3.7	2.8	2.2	2.0	2.1
LCL081-1 [ls1]				2:33.4	1:20.5	1:50.1	56.7	35.3
LCL082-1 [ls2]				3.5	2.7	1.5	0.8	0.5
LCL096-1 [LG-89]		18.1	15.5	19.3	47.3	44.3		
LCL097-1 [CD-90]		4:11.8	3:24.3	2:59.3				
LCL098-1 [LG-91]	2.3	13.4	8.1	11.8	41.7	3:17.5	3:46.5	3:29.3
LCL102-1 [LG-95]	43.4							
LCL106-1 [LG-99]	1.0	1:22.9	56.0	1.0	1.2	1.0	0.9	0.9
LCL107-1 [LG-100]		1:04.3	8.4	2:16.9				
LCL111-1 [CADE-11 Competition 6]		16.3	17.3	1:05.3	1:02.6	54.5	43.4	
LCL118-1 [R-86]			3:35.4	23.0	28.3	18.8	13.5	30.8
LCL120-1 [R-88]				6.2	15.4	11.9	29.9	1:32.8
LCL130-1 [RG-111]		32.0	3.3	13.6	1:19.0			
LCL132-1 [Lemma 1]			22.4		2:55.9	1:10.2	35.9	1:07.3
LCL143-1 [Lattice structure theorem 2]	17.6	28.5		1:49.2	1.2	0.8	0.8	0.7
LCL174-1 [Problem 2.06]		0.0	0.0	0.0	0.0	0.0	0.0	0.0
LCL178-1 [Problem 2.12]		0.2	0.1	0.1	0.0	0.0	0.0	0.0
LCL182-1 [Problem 2.16]					21.8	15.8	13.2	13.1
LCL187-1 [Problem 2.24]		0.2	0.1	0.1	0.1	0.0	0.1	0.0
LCL189-1 [Problem 2.26]		0.2	0.1	0.1	0.1	0.1	0.1	0.0
LCL193-1 [Problem 2.36]		0.4	0.4	0.3	0.1	0.1	0.1	0.1
LCL194-1 [Problem 2.37]		5.1	2.4	2.4	0.8	0.6	0.5	0.5
LCL195-1 [Problem 2.38]					30.2	21.3	18.3	17.6
LCL196-1 [Problem 2.4]						2:37.3	2:13.5	2:11.2
LCL199-1 [Problem 2.45]					7.0	4.6	3.8	3.4
LCL200-1 [Problem 2.46]		1:48.9	44.7	51.2	1.4	0.9	0.8	0.7
LCL201-1 [Problem 2.47]					11.7	7.8	6.5	6.0
LCL202-1 [Problem 2.48]					47.8	26.2	21.2	20.6
LCL203-1 [Problem 2.49]					12.6	8.5	6.9	6.7
LCL204-1 [Problem 2.5]					12.9	8.5	7.3	6.8
LCL205-1 [Problem 2.51]					1:04.7	38.9	33.9	30.3
LCL206-1 [Problem 2.52]					13.3	9.0	8.0	7.5
LCL207-1 [Problem 2.521]		1:51.2	44.9	52.0	2.2	1.5	1.3	1.3
LCL208-1 [Problem 2.53]					1:57.6	1:21.5	1:10.2	1:07.9
LCL210-1 [Problem 2.55]						4:05.1	3:32.0	3:24.8
LCL211-1 [Problem 2.56]		2:38.2	1:36.3	1:47.1	3.3	2.7	2.1	2.1
LCL213-1 [Problem 2.61]		3:04.7	1:41.8	1:53.9	7.1	5.8	5.2	4.9
LCL214-1 [Problem 2.61]		2:51.4	1:41.8	1:53.6	5.6	4.5	4.0	3.8
LCL215-1 [Problem 2.62]					26.3	17.9	16.4	14.7
LCL216-1 [Problem 2.64]		2:18.1	1:02.9	1:12.9	3.5	2.6	2.2	2.1
LCL217-1 [Problem 2.65]		2:31.3	1:03.8	1:14.3	5.4	4.2	3.8	3.6
LCL218-1 [Problem 2.67]					22.2	15.8	13.5	12.5
LCL226-1 [Problem 2.8]		0.0	0.0	0.0	0.0	0.0	0.0	0.0
LCL230-1 [Problem 2.85]					3:41.1	2:39.8	2:12.4	2:07.7
LCL231-1 [Problem 2.86]						4:01.0	3:24.6	3:19.7
LDA003-1 [Problem 3]					19.6	7.0	3.7	2.4
LDA007-3 [Problem 8]	55.4							
MSC002-1 [DBABHP]			0.7	4.2	1.3	1.1	0.7	0.8
MSC002-2			0.5	3.7	1.2	0.9	0.7	0.7
MSC006-1 [nonob.lop]	1.0	5.0	1.2		5.6	6.2	8.2	9.9
MSC007-2.002 [Pelletier 73 (Size 4)]	11.6	9.3	0.6	1:32.5	0.8	0.8	0.7	0.9
MSC008-1.002			1:08.8					
MSC008-2.002			51.6					
NUM002-1 [ls29]		0.2	0.3	1.5	0.7	0.7	0.6	0.6
NUM003-1 [Chang-Lee-10c]		4.2	1.3	0.4	0.3	0.3	0.3	0.3
NUM004-1 [Chang-Lee-10d]		0.2	0.3	1.2	0.6	0.5	0.5	0.5
NUM020-1 [ls55]		0.1	0.0	0.0	0.0	0.0	0.0	0.0
NUM021-1 [ls65]			2.4	8.2	3.8	2.9	2.6	2.1
NUM024-1 [ls75]		9.5	9.0	13.5	14.6	13.3	12.0	22.0
NUM027-1 [ls87]			53.0	1:22.8	46.5	30.7	25.1	23.2
NUM180-1 [LIM2.1]		32.5			27.0	6.2	5.8	5.7
NUM283-1.005 [fac2.lop (Size 2)]	2.2		6.0	41.4				
NUM284-1.010 [fib3.lop (Size 3)]	6.9		53.6					
PLA001-1			6.7	25.0	44.2	31.2	29.2	29.3
PLA002-1 [Problem 5.7]		46.4	0.1	3.7	3.9	3.7	8.8	20.2
PLA004-1			20.5					
PLA004-2			5.0					
PLA005-1			0.6					

PROBLEM	Best	Sym	Deep	Unopt	Opt	Skew-3	Skew-4	Skew-5
PLA005-2			1.9					
PLA006-1		0.4	0.1	0.2	0.2	0.1	0.1	0.1
PLA007-1			3.6					
PLA008-1			43.3					
PLA009-1			2.3				4:42.8	
PLA009-2	10.3	21.1	5.8				3:25.8	3:56.7
PLA010-1			2:45.1					
PLA011-1			1.9					
PLA011-2			0.7					
PLA012-1			1:25.7					
PLA013-1			3.6					
PLA014-1			24.8					
PLA014-2			6.4					
PLA015-1			2:52.6					
PLA016-1			1.7					
PLA017-1		1.1	0.5	4.6	4.6	1:50.3		
PLA018-1			46.2					
PLA019-1			1.7					
PLA021-1			3.0					
PLA022-1	0.9	12.1	0.3		3.1	1.6	1.4	1.1
PLA022-2		1.8	0.5		3.5	1.8	1.5	1.2
PLA023-1			1:24.5					
PRV001-1 [PV1]		12.4	6.5		3.3	1.3	1.5	1.7
PRV005-1 [E4]		0.1	0.1	0.1	0.1	0.1	0.1	0.1
PRV006-1 [E5]		1.2	1.8	0.4	0.4	0.4	0.4	0.4
PUZ006-1 [mars_venus.in]					3:49.5	33.8	15.8	9.9
PUZ007-1 [mars_venus2.in]	6.9					2:32.7	2:29.9	3:36.1
PUZ008-3	5.2		2.2	10.7	3:13.6	54.0	34.3	29.4
PUZ014-1 [The School Boys]			0.6		57.5	9.7		
PUZ016-2.003		0.3	0.6	1.9	0.6	0.3	0.3	0.3
PUZ023-1 [Problem 27]	0.9	0.7	6.9	2:22.8	1.2	1.4	2.2	3.9
PUZ024-1 [Problem 31]		0.4	0.1	0.3	0.2	0.2	0.2	0.2
PUZ025-1 [Problem 35]		54.8	31.5		5.5	4.5	5.8	7.0
PUZ027-1 [Problem 42]							2:59.5	3:22.1
PUZ031-1 [Pelletier 47]	12.3		3.0					
PUZ032-1 [Problem 26]		21.4	1.8		23.0	1:04.3	3:25.2	
PUZ033-1 [winds.ver1.in]		0.7	0.0	0.3	0.5	0.3	0.2	0.2
RNG001-1 [R1]					1:15.5	49.3	44.6	36.4
RNG001-3 [EX6-T]			15.1	59.6	15.9	9.4	43.5	3:20.7
RNG001-4 [R1]				2:19.0	5.2	4.5	3.9	3.9
RNG001-5 [wos21]					1:14.0	49.5	42.9	37.9
RNG002-1 [Established lemma]		0.5	0.9	20.8	3.8	2.9	2.7	2.7
RNG003-1 [Established lemma]		0.7	1.8	22.4	4.5	3.5	3.3	3.3
RNG005-2 [wos23]		0.1	0.1	0.1	0.1	0.1	0.1	0.2
RNG006-1 [Problem 25]		0.4	38.0	0.3	0.2	0.2	0.2	0.2
RNG006-2 [wos25]		0.5	1:49.8	0.3	0.2	0.2	0.2	0.2
RNG023-6		1.5	0.2	0.2	0.2	0.2	0.2	0.2
RNG023-7		1.9	0.3	0.2	0.3	0.2	0.2	0.2
RNG024-6		1.6	0.2	0.1	0.2	0.1	0.1	0.2
RNG024-7		1.9	0.3	0.2	0.3	0.2	0.2	0.2
RNG037-2 [wos24]		0.1	0.1	0.2	0.1	0.1	0.2	0.1
RNG038-1 [Problem 27]					3:55.7	2:48.4	2:45.2	2:31.4
RNG038-2 [wos27]		0.2		0.1	0.2	0.2	0.1	0.2
RNG040-1 [Problem 29]		0.2	0.2	0.2	0.2	0.2	0.2	0.2
RNG040-2 [wos29]		1.1	1:14.4	19.8	6.0	21.8	1:19.3	
RNG041-1 [wos30]		17.9		3.2	3.3	3.3	3.2	3.3
ROB010-1 [Lemma 3.3]	9.3	45.4			46.5	23.6	16.6	10.9
ROB013-1 [Lemma 3.5]	1.0	14.2		16.8	4.6	2.6	5.1	11.0
ROB016-1 [Corollary 3.7]	0.3			5.3	3.1	2.2	1.8	1.6
ROB021-1	4.9	4:58.0	13.9	3:02.7	44.7	29.3	1:18.4	
SET005-1 [ls108]					2:13.0	28.2		
SET008-1 [ls115]		0.4	0.2	0.4	0.2	0.2	0.2	0.2
SET009-1 [ls116]		31.4	3:16.4	19.9	1.6	0.7	0.5	0.4
SET011-1 [ls121]					2:21.0	18.4	1:10.0	
SET014-2 [EST-S4]						3:43.5		
SET016-7 [OP4]		2.4	3.2	2.3	2.4	2.4	2.4	2.4
SET018-7 [OP5]		2.5	3.2	2.5	2.4	2.5	2.5	2.5
SET024-3 [Lemma 9]			1:51.8					
SET024-4 [Lemma 9]			1:53.4					
SET024-7 [SS2]			4:07.9	19.1	8.1	8.1	8.3	8.4
SET025-3 [Lemma 10]			1:48.8	18.0	15.7	15.6	15.6	15.5
SET025-4 [Lemma 10]			1:46.1	17.1	14.9	15.1	15.0	15.3
SET025-7 [OP1]		2.1	2.8	2.1	2.1	2.1	2.1	2.1
SET027-6	1.9				9.8	8.5	8.2	6.6
SET027-7 [PO3]	2.0				10.8	9.1	9.0	6.9
SET041-3 [Lemma 26]			2:39.8					
SET047-5 [p43.in]	0.4	0.1	0.1		0.4	1.0	3.7	10.0
SET050-6	1.4	3.4	1:55.4	1.5	1.5	1.4	1.4	1.4
SET051-6	1.4	3.4	1:50.5	1.5	1.5	1.4	1.5	1.4
SET055-6	1.3	1.3	1:33.2		6.8			
SET059-7 [EQ2.4]	1.6	1.1	1.5	1.3	1.3	2.3	9.6	1:04.3
SET061-7 [SP2]		9.6		25.5	9.9	10.3	10.3	10.0
SET065-7 [SP5]	7.1		3:12.5	17.7	7.6	7.7	7.8	7.9
SET073-7 [UP6.1]	7.3		2:21.7	20.0	7.7	7.7	8.0	8.1
SET074-7 [UP6.2]	7.2		2:15.8	19.8	7.8	7.7	8.1	8.2
SET075-7 [UP6 cor.]	7.8		2:21.2	19.7	7.9	7.7	8.0	7.8
SET077-7 [SS1]		1.6	2.0	1.6	1.6	1.6	1.6	1.5
SET079-7 [SS2 cor.1]	7.0		3.5	20.0	7.7	7.7	7.9	8.0
SET080-6	1.7				1:18.6	1:12.3	1:11.3	1:09.6
SET081-6	1.8		17.8	15.9	2.2	2.0	1.9	1.9
SET081-7 [SS3]			4:14.3					

PROBLEM	Best	Sym	Deep	Unopt	Opt	Skew-3	Skew-4	Skew-5
SET082-7 [SS4]			4:27.4					
SET083-6	2.3							
SET090-7 [SS7]			4:58.0	20.9	9.4	9.5	9.4	9.3
SET093-6		1:11.6	2.4	2.4	2.3	2.3	2.2	2.2
SET094-6 [SS10]	4.6							
SET098-7 [SS13 cor.1]		9.5		22.3	9.9	9.9	10.3	10.0
SET101-6	1.7		23.2	1:28.6	11.3	10.2	10.4	9.9
SET101-7 [OP2.1]				23.5	9.7	9.7	9.8	10.3
SET102-7 [OP2.2]				22.9	9.4	9.3	9.8	9.8
SET108-6		2:05.0		3.6	3.2	3.2	3.2	3.2
SET117-6		1:10.1	2.4	2.5	2.2	2.1	2.2	2.3
SET152-6 [C2.2]	2.3		6.3	1:51.6	1:35.0	1:36.4	1:38.4	1:33.1
SET153-6 [C3.1]	2.3	1:48.2	6.1	1:59.7	1:44.4	1:40.0	1:44.7	1:40.9
SET184-6 [SU2]	2.3	14.0		22.3	4.4	3.9	3.7	3.8
SET187-6 [SU5]	2.8							
SET196-6 [LA1.3]	1.9	1.5	2:19.8	1.4	1.4	1.4	1.4	1.3
SET197-6 [LA1.4]	2.0	1.4	2:20.5	1.4	1.4	1.4	1.4	1.4
SET203-6 [CP1 cor.]		2:01.3	2.5		2:26.6			
SET204-6 [CP2]	2.5	3.1	2.1	3.7	3.4	16.1	1:50.8	
SET231-6 [CP14.1]		1.4	1.7	1.4	1.4	1.4	1.4	1.4
SET234-6 [CP14.4]				2:05.2	1:55.2	1:54.8	1:48.0	1:54.9
SET236-6 [CP15.2]					3:48.7	3:27.9	3:29.3	3:18.5
SET239-6 [RS1]		9.7	21.0	2.7	2.5	2.6	2.5	2.5
SET240-6 [RS2]		22.2		16.6	5.0	5.0	4.8	4.8
SET241-6 [RS3]		36.3		49.2	7.0	6.6	6.2	6.5
SET242-6 [RS4]	2.1	1:26.2	1.9	3.3	3.0	2.9	3.0	2.9
SET252-6 [RS10.1]	2.9	13.5		22.4	4.7	4.2	4.0	4.0
SET253-6 [RS10.2]		14.2		23.2	4.9	4.4	4.2	4.2
SET411-6 [CO15]						4:12.5	3:50.2	3:50.2
SET451-6 [SR1]		13.9		53.5	5.0	4.6	4.4	4.3
SET479-6 [RP2.1]		7.6	32.6	2.3	2.4	2.3	2.3	2.3
SET553-6 [CA1]		14.1		30.1	5.2	4.7	4.6	4.6
SYN002-1.007:008 [eder7-8.lop]		1:06.3	56.7		1:57.3	3:51.4		
SYN004-1.007 [Problem 5.3]	0.1	0.3	0.2	5.9	1.7	16.3		
SYN010-1.005:005 [Example 5.1]	5.2	5.6	3.5					
SYN012-1 [Example]			33.2					
SYN038-1 [EX4-T]			31.1					
SYN058-1 [Pelletier 28]		0.0	0.0	0.0	0.0	0.0	0.0	0.0
SYN067-2	16.5				3:58.7			
SYN070-1 [p46.in]		29.8	0.4		1:04.1	25.3	27.5	29.8
SYN071-1 [Pelletier 48]	1.4	5.3	4.1		2.2	2.5	3.0	3.4
SYN072-1 [Pelletier 49]	9.9				51.2	9.9	11.6	12.9
SYN074-1 [Pelletier 51]					2:23.8	5.7	5.5	6.3
SYN075-1 [Pelletier 52]						7.8	7.6	9.3
SYN081-1 [Pelletier 59]		0.0	0.0	0.0	0.0	0.0	0.0	0.0
SYN090-1.008 [T3n]			9.9					
SYN094-1.005 [U(T3n)]			51.2					
SYN096-1.008 [M(T3n)]			12.7					
SYN098-1.002 [Sym(U(T3n))]		1.2	0.3	2:00.6	1.4	1.5	1.7	2.1
SYN099-1.003 [Sym(M(T2n))]		0.5	0.5	0.5	0.5	0.5	0.5	0.5
SYN100-1.005 [Sym(M(T3n))]			28.5		3:46.1	1:32.6	50.5	31.8
SYN102-1.007:007 [N(T3n))]			25.0					
SYN137-1	5.0	58.8	5.0	4:03.3	7.2	4.8	4.6	4.7
SYN139-1	35.5	1:17.1	4.4		2:34.7	7.3	5.9	5.5
SYN140-1	35.6	1:18.0	4.3		4:41.9	13.3	8.4	7.4
SYN142-1		2:42.4	4.5					1:40.0
SYN143-1		2:36.8	4.5					1:38.3
SYN155-1			14.6		14.1	4.8	4.8	5.0
SYN156-1			14.8		20.5	4.8	9.1	1:14.5
SYN159-1			13.9			14.3	12.5	13.4
SYN163-1			17.0		4:04.4	13.2	12.0	12.9
SYN171-1	6.1	9.8	4.5			2:50.1	56.9	35.3
SYN178-1		9.8	5.7	7.4	4.3	4.1	4.2	4.2
SYN179-1		1:33.7	49.9		22.1	7.7	7.8	8.9
SYN180-1			5.3		23.4	5.7	5.2	5.1
SYN190-1		11.9	6.3		12.8	6.3	5.6	5.7
SYN202-1		7.6	5.6	6.9	4.3	4.2	4.2	4.3
SYN204-1	29.1	40.8	4.4	1:19.4	15.8	5.2	4.7	4.9
SYN205-1	29.2	40.5	4.3	1:19.5	15.3	5.2	4.7	4.7
SYN213-1	9.6	2:17.0	4.5		5.0	4.3	4.3	4.2
SYN214-1	5.5	1:56.1	4.5		4.7	4.3	4.2	4.2
SYN215-1	5.7	1:52.2	4.5		5.0	4.4	4.3	4.4
SYN252-1	35.1	1:18.6	4.3		3:10.1	11.0	7.1	6.4
SYN253-1	34.5	1:17.1	4.3			19.5	11.3	10.0
SYN254-1	34.0	1:15.4	4.4		1:28.4	9.1	6.3	5.4
SYN269-1	6.0	10.1	4.5			2:09.5	43.2	31.1
SYN271-1	6.0	10.3	4.6			2:57.5	59.2	37.0
SYN311-1 [H2]	0.1	4:06.0		3:27.7	3:58.2	3:41.2	3:41.9	3:42.4
SYN328-1 [Ch12N3]		38.5	2.2		1:16.4	22.6	51.5	1:10.4
SYN334-1 [Ch14N6]			29.1					
SYN347-1 [Ch17N3]	9.3	6.5	1.9	2:23.8	6.6	4.7	4:01.2	3:20.5
SYN349-1 [Ch17N5]	11.0	1.3	0.4		3.1	6.0	12.6	23.1
TOP001-2 [Lemma 1a]		11.8	1.6		10.1	8.0	5.7	5.0
TOP005-2 [Lemma 1e]			7.2		1:05.9	14.8	23.7	3:10.9

7.2 The value of caching

On the basis of these examples, caching is unmistakably worthwhile. Almost every problem is solved more quickly and with fewer inferences if caching is used (of course it is inevitable that caching cannot increase the number of inferences, but the time performing cache lookup may swamp any improvements). The difference is not usually spectacular, but significant. Caching permitted the solution of 40 problems not solved without it, whereas only 3 were solved without and not with. Here are the situations where the runtimes are at least a minute (or not within 5 minutes) in one or both cases. Note that even if the number of inferences is the same in both cases, a longer runtime in the no-caching case need not be put down to experimental error: caching may provide a cheaper way of discovering that no rule applications at all are possible.

	Run times		Inferences	
PROBLEM	with	without	with	without
BOO003-4	2:25.7	3:47.0	159,185	259,953
BOO004-4	2:09.0	3:25.9	137,484	227,705
BOO012-1	1:25.2	2:35.8	150,869	261,774
CAT008-1	4:20.5		450,863	
COL033-1	1:48.4	2:08.1	42,044	49,283
COL040-1	3:07.2	3:41.4	62,727	76,417
COL044-1	3:07.2	3:46.5	62,727	76,417
COL060-3	1:56.6	3:39.5	82,422	149,162
GEO039-2		2:37.5		154,897
GEO039-3	2:11.3	2:13.7	189,818	193,121
GEO040-2		3:49.1		233,073
GEO047-3	2:33.6	2:38.9	228,579	235,054
GRP012-2	2:11.5	2:36.0	192,809	230,074
GRP012-3	2:25.7	3:07.9	192,119	238,679
GRP022-2	43.8	1:03.8	56,719	86,825
GRP123-1.003	1:06.1	3:02.4	225,810	591,981
GRP123-2.003	1:09.1	3:18.2	236,541	631,936
GRP123-3.003	1:16.3	3:10.7	245,418	622,364
GRP123-4.003	1:28.2	3:26.8	274,002	645,597
GRP123-6.003	2:30.6		521,931	
GRP123-7.003	2:31.0		522,064	
GRP123-8.003	2:33.1		530,070	
GRP123-9.003	2:30.0		521,931	
GRP124-1.003	1:06.1	3:01.0	225,810	591,990
GRP124-2.003	1:11.0	3:18.4	236,541	631,945
GRP124-3.003	1:13.8	3:12.6	245,418	622,373
GRP124-4.003	1:22.6	3:22.0	274,002	645,606
GRP124-6.003	2:28.2		521,580	
GRP124-7.003	2:28.3		521,713	
GRP124-8.003	2:31.6		529,719	
GRP125-1.003	2:20.9		452,451	
GRP125-2.003	2:36.9		469,749	
GRP125-3.003	2:39.5		482,600	
GRP127-1.003	2:13.8		411,861	
GRP127-2.003	2:27.1		430,332	
GRP127-3.003	2:30.8		439,273	
GRP129-1.002	4:24.8		914,465	
GRP129-4.002	44.8	1:12.1	140,807	240,859
GRP130-2.002	44.0	1:05.8	157,509	236,495
GRP130-3.002	59.0	1:22.1	209,928	295,491
GRP130-4.002	53.9	1:19.2	176,369	252,369
GRP133-1.002	4:07.9		872,714	
GRP133-2.002	4:33.6		972,901	
GRP135-2.002	43.5	1:01.9	164,029	228,835
GRP140-1	4:54.9		160,911	
GRP162-1	1:48.5	1:55.6	79,665	93,281
GRP163-1	3:21.4	3:42.2	139,252	166,843
GRP165-1	2:22.0	3:06.9	81,756	117,876
GRP166-3	2:24.4	3:07.7	81,813	117,933
HEN004-4	41.7	1:05.3	68,435	117,368
HEN005-2	1:44.9	2:52.4	206,307	382,583
HEN008-2	36.0	1:11.4	73,288	157,874
HEN009-2	1:25.6	2:10.5	171,004	264,807
LCL045-1	2:43.3	2:44.0	26,693	26,693
LCL064-2	2:22.3	2:24.3	22,504	23,771
LCL081-1	1:20.5	1:24.0	12,748	12,748
LCL111-1	1:02.6	1:08.0	16,883	16,883
LCL130-1	1:19.0	1:26.1	8,251	8,251
LCL132-1	2:55.9	4:41.4	146,118	239,709
LCL196-1		4:28.4		148,815
LCL205-1	1:04.7	58.6	37,867	37,867
LCL208-1	1:57.6	1:51.8	71,728	71,728
LCL230-1	3:41.1	3:35.2	105,707	105,707
PUZ006-1	3:49.5		972,547	
PUZ008-3	3:13.6		313,948	
PUZ014-1	57.5		509,410	
RNG001-1	1:15.5	2:11.7	123,792	217,499
RNG001-5	1:14.0	2:04.2	120,279	211,509
RNG038-1	3:55.7		386,374	
ROB010-1	46.5	1:05.6	30,225	42,479
SET005-1	2:13.0		377,191	
SET011-1	2:21.0		385,106	
SET080-6	1:18.6	1:23.9	112,094	112,149
SET152-6	1:35.0	1:38.3	182,665	182,777
SET153-6	1:44.4	1:44.9	182,195	182,307
SET203-6	2:26.6	2:29.9	168,151	168,151
SET234-6	1:55.2	1:47.1	182,364	182,366
SET236-6	3:48.7	3:50.5	333,608	334,497
SYN002-1.007:008	1:57.3	1:58.9	55,437	55,437
SYN067-2	3:58.7		799,398	
SYN070-1	1:04.1		385,685	

PROBLEM	Run times		Inferences	
	with	without	with	without
SYN072-1	51.2	2:12.8	191,493	500,951
SYN074-1	2:23.8		314,333	
SYN100-1.005	3:46.1		1,211,849	
SYN139-1	2:34.7		438,113	
SYN140-1	4:41.9		778,650	
SYN155-1	14.1		34,340	
SYN156-1	20.5		57,983	
SYN163-1	4:04.4		767,717	
SYN179-1	22.1	1:21.8	55,139	258,502
SYN180-1	23.4		65,199	
SYN190-1	12.8		27,993	
SYN204-1	15.8		31,569	
SYN205-1	15.3		30,576	
SYN252-1	3:10.1		531,046	
SYN254-1	1:28.4		227,636	
SYN311-1	3:58.2	3:41.1	278,299	278,299
SYN328-1	1:16.4		302,604	
TOP005-2	1:05.9	2:02.5	155,369	288,657

We have already emphasized that caching can be regarded as a general Prolog implementation optimization. For example, we have tried introducing caching of continuations into a simple tableau prover roughly based on leanT^AP (Beckert and Posegga 1994), also coded in CAML Light, though running *interpreted* (so these runtimes are inflated). On some problems (generally, the harder ones), caching makes a significant improvement; in other cases it introduces only a modest loss. Here are a few results on the first 46 Pelletier problems:[13]

Problem	Without caching	With caching
1	0.00	0.01
11	0.00	0.00
21	0.01	0.01
33	0.00	0.01
34	118.56	68.08
35	0.01	0.01
36	0.01	0.08
37	0.13	0.13
38	0.78	0.83
43	11.51	0.61
44	0.01	0.03
45	0.88	0.88
46	0.15	0.11

7.3 The value of the positive refinement

Throughout the above tests, we used Plaisted's positive refinement. At least we used a partial version: no ancestor solutions were tried for positive goals, but these were always checked for repetition. Perhaps this is wasteful, but these equality tests are quite cheap since the ancestor lists are seldom long. As Plaisted points out, there is an advantage in not even storing certain ancestors: global caching schemes are more likely to be useful because with short ancestor chains the probability of a goal's arising in the same ancestor context is greatly increased.

The process of running this test suite seemed a good opportunity to assess its usefulness; in the original article Plaisted (1990) gave an extensive theoretical analysis but no practical results. Accordingly, we also ran the 'opt' version without the positive refinement but with no other change. The results are not clear-cut. Both versions solved exactly 850 problems within the time limit, but not the same ones: each solved 15 problems which the other didn't. The following problems were solved only with the positive refinement:

[13]Some of these, like Andrews's challenge (34) are not directly solvable in a reasonable time by MESON, which might lead one to suppose that tableaux are better than MESON. In some cases this is true; however if a preprocessing pass is added to split the problem into subproblems (e.g. to refute $(p \vee q) \wedge r$, refute $p \wedge r$ and $q \wedge r$ separately), which happens implicitly in tableaux anyway, then all the problems are trivial for MESON. For example Andrews's challenge splits into 32 independent subproblems (if bi-implications are expanded appropriately during translation to clausal form), each of which is easy.

PROBLEM	Run time	Inferences	Size
NUM228-1	2.6	35	3
PLA001-1	44.2	24,166	12
PLA002-1	3.9	4,798	11
PLA003-1	0.4	369	7
PLA006-1	0.2	157	6
PLA017-1	4.6	4,548	9
PLA020-1	0.1	22	4
PLA022-1	3.1	3,364	15
PLA022-2	3.5	3,790	15
PRV001-1	3.3	15,499	21
PRV003-1	0.0	52	3
PRV005-1	0.1	203	4
PRV006-1	0.4	977	5
SYN067-2	3:58.7	799,398	52
SYN140-1	4:41.9	778,650	29

while the following were solved only without it:

PROBLEM	Run time	Inferences	Size
CAT019-3	3:40.7	452,699	11
GEO039-2	3:49.6	147,932	10
GRP129-2.002	29.4	106,842	27
GRP129-3.002	41.3	145,537	27
GRP131-1.002	3:14.0	704,975	37
GRP131-2.002	3:36.9	770,958	37
GRP132-1.002	2:58.6	648,976	37
GRP132-2.002	3:21.6	710,351	37
GRP134-1.002	31.9	114,314	27
GRP134-2.002	35.4	126,958	27
PUZ007-1	3:53.2	846,428	28
PUZ027-1	4:14.0	627,260	26
PUZ031-1	4:34.8	1,204,039	53
SET232-6	3:55.2	382,317	7
SET233-6	3:57.0	382,318	7

Otherwise, most of the results tend to be rather similar, with the positive refinement slightly better on average. It's worth noting that in completely Horn problems (and nearly half the TPTP problems *are* completely Horn), negative goals never arise, so there is no substantial difference to be expected (the positive refinement is slightly quicker because it just takes one sign check instead of selecting an empty set of possible ancestors to unify with). Here are a few where the difference seems particularly clear-cut.

	Run times		Inferences		Size	
PROBLEM	with	without	with	without	with	without
GRP135-1.002	38.6	10.0	148,122	38,662	29	23
GRP135-2.002	43.5	11.5	164,029	42,695	29	23
MSC006-1	5.6	17.6	25,613	67,402	20	20
PUZ006-1	3:49.5	51.7	972,547	213,088	25	21
PUZ014-1	57.5	5.8	509,410	49,129	71	47
PUZ015-2.003	3.7	2.0	44,963	22,056	24	20
PUZ025-1	5.5	3.1	19,244	9,836	18	16
PUZ032-1	23.0	2.4	65,838	6,496	15	11
SET152-6	1:35.0	3.3	182,665	3,679	6	5
SET153-6	1:44.4	3.3	182,195	3,611	6	5
SYN002-1.007:008	1:57.3	2:46.5	55,437	55,437	32	32
SYN070-1	1:04.1	11.5	385,685	62,193	28	20
SYN072-1	51.2	1:13.9	191,493	260,458	26	26
TOP005-2	1:05.9	4.6	155,369	11,482	32	19

7.4 Full results

Because of space limitations, we cannot give a comprehensive listing of each search mode, complete with runtimes, numbers of inferences and other statistics. However this is available as a text file on the Web from:
`http://www.cl.cam.ac.uk/users/jrh/papers/me-results.txt`.

8 Conclusions

It is clear that our optimization is a large advance on Stickel's original scheme for iterative deepening. Moreover this and caching, the other implementation optimization we have discussed, are rather general. The divide-and-conquer optimization is applicable to any situation where search is bounded by a cumulative size measure. And caching can be regarded as a general Prolog optimization. So our work is a nice complement to the research that is being done on extending the model elimination calculus. There are lots of areas worthy of future investigation.

1. Best-first search could be further explored. The present heuristics used are extremely simplistic, and it would be interesting to experiment with alternatives. More sophisticated search methods could be tried. In the above tests, the rather low space limit may have crippled it unfairly. Given a priority queue with reasonable locality properties, the queue size could be extended well beyond the physical store limit of the machine.

2. The optimized form of inference-bounded search can be tried with other model elimination calculi. For example, restart model elimination as developed by Baumgartner and Furbach (1993) is a variant which only uses natural contrapositives. The restart steps tend to introduce asymmetry into the proofs, so it may be that optimized inference-bounded search will perform still better here relative to depth-bounded search.

3. It seems there should be scope for integrating the divide-and-conquer optimization more tightly with the iterative increase in the bound. For example, increasing any even number n by 1 doesn't change $n/2$, which leads to even greater duplication of results than is usually the case with iterative deepening. With some ingenuity, it ought to be possible to reduce this by combining iterative deepening with descent down the subgoal tree.

4. Combinations of inference and depth bounds could be tried. For example, inference bounded search in conjunction with a conservative but reasonable depth bound might turn out to give the best of both worlds, avoiding excursions into extremely skewed proofs. But some care must be taken to avoid pointless reruns by the optimization if a search failed purely because of a depth overspill.

Acknowledgements

Thanks to Larry Paulson for spurring my interest in the MESON procedure, and to Ralph Back for letting me work on these topics. My work has been generously funded by the European Commission under the Human Capital and Mobility programme. Comments on a draft of this paper from Richard Boulton and Jim Grundy were extremely helpful.

References

Astrachan, O. L. and Stickel, M. E. (1992) Caching and lemmaizing in model elimination theorem provers. In Kapur, D. (ed.), *11th International Conference on Automated Deduction*, Volume 607 of *Lecture Notes in Computer Science*, Saratoga, NY, pp. 224–238. Springer-Verlag.

Baumgartner, P. and Furbach, U. (1993) Model elimination without contrapositives and its application to PTTP. Research report 12-93, Institute for Computer Science, University of Koblenz, Rheinau 1, 56075 Koblenz, Germany.

Beckert, B. and Posegga, J. (1994) leanT^AP: lean, tableau-based theorem proving. In Bundy, A. (ed.), *12th International Conference on Automated Deduction*, Volume 814 of *Lecture Notes in Computer Science*, Nancy, France, pp. 793–797. Springer-Verlag. Extended version available on the Web from `http://i12www.ira.uka.de/~posegga/LeanTaP.ps.Z`.

Gordon, M. J. C. and Melham, T. F. (1993) *Introduction to HOL: a theorem proving environment for higher order logic.* Cambridge University Press.

Gordon, M. J. C., Milner, R., and Wadsworth, C. P. (1979) *Edinburgh LCF: A Mechanised Logic of Computation*, Volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag.

Letz, R., Mayr, K., and Goller, C. (1994) Controlled integrations of the cut rule into connection tableau calculi. Technical Report AR-94-01, Technische Universität München, Arcisstrasse 21, 80290 München, Germany.

Letz, R., Schumann, J., Bayerl, S., and Bibel, W. (1992) SETHEO: A high-performance theorem prover. *Journal of Automated Reasoning*, **8**, 183–212.

Lifschitz, V. (1986) *Mechanical Theorem Proving in the USSR: the Leningrad School*. Monograph Series on Soviet Union. Delphic Associates, 7700 Leesburg Pike, #250, Falls Church, VA 22043. Phone: (703) 556-0278. See also 'What is the inverse method?' in the Journal of Automated Reasoning, vol. 5, pp. 1–23, 1989.

Loveland, D. W. (1968) Mechanical theorem-proving by model elimination. *Journal of the ACM*, **15**, 236–251.

Michie, D. (1968) "Memo" functions and machine learning. *Nature*, **218**, 19–22.

Paulson, L. C. (1987) *Logic and computation: interactive proof with Cambridge LCF*. Number 2 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

Paulson, L. C. (1994) *Isabelle: a generic theorem prover*, Volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag. With contributions by Tobias Nipkow.

Plaisted, D. A. (1990) A sequent-style model elimination strategy and a positive refinement. *Journal of Automated Reasoning*, **6**, 389–402.

Rudnicki, P. (1987) Obvious inferences. *Journal of Automated Reasoning*, **3**, 383–393.

Stickel, M. E. (1988) A Prolog Technology Theorem Prover: Implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, **4**, 353–380.

Suttner, C. B. and Sutcliffe, G. (1995) The TPTP problem library. Technical Report AR-95-03, Institut für Infomatik, TU München, Germany. Also available as TR 95/6 from Dept. Computer Science, James Cook University, Australia, and on the Web.

Tarver, M. (1990) An examination of the Prolog Technology Theorem-Prover. In Stickel, M. E. (ed.), *10th International Conference on Automated Deduction*, Volume 449 of *Lecture Notes in Computer Science*, Kaiserslautern, Federal Republic of Germany, pp. 322–335. Springer-Verlag.

Weis, P. and Leroy, X. (1993) *Le langage Caml*. InterEditions. See also the CAML Web page: `http://pauillac.inria.fr/caml/`.