EXPERIMENTS WITH THE ALPOC THEOREM PROVER

by

B. Spencer, J.D. Horton and K. Francis

TR95-094, June 1995

Faculty of Computer Science
University of New Brunswick
Fredericton, N.B. E3B 5A3
Canada

June 1995

Phone: (506) 453-4566
Fax: (506) 453-3566

Experiments with the ALPOC Theorem Prover

Bruce Spencer, J. D. Horton and Kelsey Francis
email: bspencer@unb.ca, jdh@unb.ca
http://www.cs.unb.ca

Abstract

A system for selecting and preparing a batch of data files, and running a selected computer program with each data file is described. Another facility collects the experimental results from several different batches and summarizes the results in a tabular form.

A theorem prover, ALPOC, is implemented that combines Shostak's C–literal resolution steps with Stickel's PTTP compiler, and uses Spencer's Ordered Clause set restriction. A series of experiments is run that compares ALPOC with PTTP, using the batch system. The results are summarized and compared with Stickel's PTTP implemented in Prolog. The results show that ALPOC is slower than PTTP by a factor of at most 4, but frequently is much faster. On the problems where ALPOC is faster the number of extension steps in the ALPOC proof is less than the PTTP proof, which leads the iterative deepening search method to explore fewer levels.

1. Introduction

Clause trees provide a new methodology/data structure for resolution[Rob65]. They are developed extensively in [HS94,HS95]. In this paper we describe an implementation of the ALPOC procedure for building rooted clause trees. This is an adaptation of the procedure described by Shostak [Sho76], and the ordered clause set restriction [Spe91, Spe93] to the clause tree framework. The implementation is based on the technique proposed by Stickel [Sti92] in the Prolog Technology Theorem Prover, PTTP. In fact the code for PTTP formed the basis of the ALPOC program. Most of the problems on which the system was run were provided by the Thousands of Problems for Theorem Provers library, TPTP [SSY94]. We have included as many of the problems as we could that were reported by Stickel in [Sti88].

Some of the empirical results in this report differ from previously reported work [HS94]. There are two major differences between the two ALPOC programs. Some of the proofs that were rejected in the previous implementation contained unifiable tautology paths, but did not contain tautology paths. Therefore they should not have been rejected. This made the procedure incomplete.

The second major difference was to eliminate left merge paths to atom nodes that were proved by a unit input clause. This is accomplished by *not* putting these atom nodes in the list of atom nodes to the left of the current node that are visible from the current node.

The effect of this is to decrease the number of inferences, since both types of proof would be tried at the same level of the search, but only one is needed. In addition, keeping the visible list smaller speeds up the other operations that use the visible list.

2. Description of the Batch System

The batch system described here allows the user to conveniently perform three successive tasks on a given set of input files. These tasks are

1. Convert an input file into a data file with a given suffix.
2. Run a given program with a UNIX command line that takes its input from the data file and writes its output to an output file with another given suffix. This command line is given to the UNIX batch processor, so that the program can be run in the background.
3. Summarize the data from a number of different output files with the same suffix.
4. Compare the summaries of the output files with one suffix against those with another suffix.

The user specifies (in any order) the following information in a file to be read by Prolog:

1. each name of the data sets of interest	select_data_set/1
2. the command that generates the data	generate_data_file/1
3. the command that processes the data	process_command/1
4. the file suffix of the data file	data_file_suffix/1
5. other command-line parameters of item 3 specified one at a time also specifies the data set of interest	command_line_parameter/2
6. a cpu time limit on the batch file	cpu_time_limit/1
7. the directory from which the input for item 2 comes	path_data_file_directory/2
8. the suffix of the output files from item 3	output_file_suffix/1
9. the path to where the output file is written	output_file_path/1

The data sets of interest can be specified by the enumeration of individual problem names, and/or by accessing a list of problems asserted from a file as facts.

1. Individual problems can be specified using the problem name:
ex. select_data_set('COM001-1').
select_data_set('MSC007-1.003').

2. Collections of problems may be specified from a file, possibly based upon certain selection criteria.
ex. All of Stickel's problems are selected with the following:

```
select_data_set(DataSet):-
    ensure_loaded('/ail4/kfrancis/Process/stickels.qof'),
    stickels(DataSet).
```

3. Problems may be selected using properties as specified in the 'ProblemSynopsis' file accompanying tptp problems. In the following lines, only propositional problems with at least 20% non-Horn clauses and no more than 4 literals-per-clause (L_C) will be selected:

```
select_data_set(DataSet):-
    ensure_loaded('/ail4/kfrancis/Process/problem_synopsis.qof'),
    problem_synopsis(DataSet, Number_of_clauses, Percent_NonHorn,
                Percent_Unit, Percent_Equality, Prop_or_NP,
                Lit_per_clause),
    Prop_or_NP = 'Pr',
    Percent_NonHorn >= 20.0,
    Lit_per_clause =< 4.0.
```

Here is a sample of a complete command file:

```
select_data_set('RNG001-2').
path_data_file_directory('/ail4/bspencer/tptp/TPTP-v1.1.1/Problems').
generate_data_file('prolog +1 /ail4/kfrancis/batch/convert').
processor_command('prolog +1 /ail4/kfrancis/pttp/pttp_alpoc.qof').
command_line_parameter(_, count_inferences).
command_line_parameter(_, print_proof).
command_line_parameter(_, prove_this).
command_line_parameter(_, halt).
cpu_time_limit(30000).  %seconds
data_file_suffix('.pttp').
output_file_suffix('.alpocV1.1').
output_file_path('/ail4/kfrancis/tptp/TPTP-v1.1.1/Infer/').
```

2.1 A Tree Drawing Package

In addition to the batch system, a separate system has been developed by David Sharpe for displaying proofs generated by PTTP, and related systems like ALPOC. It displays the proof in ASCII text as a rooted clause tree, and spaces the nodes on the page for ease of reading. Merge paths are indicated by labels next to the closed leaves.

3. Experimental Conditions and Results

All programs were written in Prolog using Quintus Prolog 3.1.1, and run on a Sun SPARCStation 2. A total of 112 problems were run using ALPOC. A complete set of measurements is found in Appendix B. Of these, 7 did not run on PTTP because they ran over the time limit of 20,000 seconds. Every problem that PTTP ran, was run by ALPOC. Twenty-four problems took ALPOC more than one second, and 7 more took PTTP more than one second. The results on these 31 problems are listed in the table below. Time is measured in seconds. These times are not very accurate; a second run was done for some, and the times often varied by about 10% either way. Therefore the time listed in the table have been truncated to two significant digits, and the second of these is suspect.

NAME	Time in seconds (Ratio)				Inferences		Proof Size	
	ALPOC		PTTP		ALPOC	PTTP	A	P
EX6-T2	9.1		50.	(5.5)	31868	556894	10	14
GRP001-1	32.	(2.7)	12.		100839	98881	9	
GRP008-1	640		12000	(18.)	1260198	NA	10	12
GRP009-1	3.6	(2.5)	1.4		13204	13008	8	
GRP012-2	570	(3.0)	190		1504907	1421095	11	
GRP013-1	2.9	(2.4)	1.2		9300	9191	7	
GRP029-1	.27		13.	(49.)	1134	138528	6	9
GRP030-1	.08		1.9	(23.)	402	20332	6	9
GRP037-3	7.3	(2.4)	3.0		23898	23549	7	
MSC001-1	35.		61.	(1.8)	104203	328492	12	13
MSC004-1	.48		77.	(160)	1032	260936	12	23
MSC007-1(5)	.23		NA	NA	674	NA	prop'l	
MSC007-1(6)	2.1		NA	NA	4975	NA	prop'l	
MSC007-1(7)	20.		NA	NA	40527	NA	prop'l	
MSC007-1(8)	260		NA	NA	365330	NA	prop'l	
NUM024-1	14.	(2.2)	6.4		27869	27555	7	
NUM027-1	25.	(2.2)	11.		47128	48418	8	
PRV001-1	19.		8100	(420)	47723	NA	13	19
PUZ023-1	2.6		21.	(8.1)	5419	109233	9	12
PUZ025-1	63.		4400	(70.)	125247	21363017	13	17
RNG001-2	3300		NA	NA	NA	NA	14	NA
RNG001-3	1.9		14.	(7.5)	6230	131224	14	18
RNG001-5	2200	(3.8)	570		5529364	4777282	11	
RNG040-2	70.	(2.3)	30.		162762	165983	7	
RNG041-1	3.6	(2.5)	1.5		8791	8828	6	
SET009-1	8.8	(2.2)	3.4		16628	23409	11	
SYN001-1(5)	0.07		1100	(17000)	465	26790196	prop'l	
SYN038-1	24000		NA	NA	NA	NA	21	NA
SYN082-1	0.05		25	(500)	79	131071	5	13
TOP005-2	75.		NA	NA	90740	NA	16	NA
stark035	0.18		7.6	(41.)	399	50089	8	12

Table 1. Comparison of ALPOC and PTTP on selected problems

The ratio of the times is listed in parentheses beside the larger value in Table 1. PTTP was faster for 11 problems, and ALPOC for 20. The largest ratio by which ALPOC was worse than PTTP was 3.8 for RNG005-1. All of the other ratios were between 2.2 and 3.0. The ratios by which ALPOC was better than PTTP varied much more. Only one of these

ratios was less than 4; the other 19 were greater. The largest known ratio was 17,000 for SYN001-1.005; the ratios for the pigeon-hole problems would be even greater (if we could have measured PTTP on these).

The table also contains the number of inferences required by each procedure to search for a proof, and the size of the proof (number of atom nodes minus one and not counting the query node). One can check that PTTP does two to three times more inferences per second than ALPOC. The number of inferences per second varied more among the problems than it did between the theorem provers. For PTTP the inference rate varied from about 3,000 for MSC004-1 to 24,000 for SYN001-1.005; most fell in the range 4,000-10,000. For ALPOC the inference rate for most problems ranged from somewhat below 2,000 to somewhat above 3,000 inferences per second.

The size of the proof obtained seems to be a determining factor of performance. If the size of the proofs for ALPOC and PTTP are the same, the number of inferences are about the same, so ALPOC runs two to three times slower. But if the ALPOC proof is smaller (and it can never be larger) then ALPOC takes many fewer inferences, and much less time than PTTP, often by a big factor.

The size of the proof was not discovered by the system if the problem was purely propositional. The size of the proof was discovered by counting the number of levels the search took using iterative deepening. But the propositional problems did not use iterative deepening.

4. Conclusions

The batch system is a general purpose program for preparing, running and summarizing experiments. It has been used to compare ALPOC and PTTP in this paper.

The ALPOC procedure runs a small set of these problems dramatically faster than PTTP. Usually this happens because the ALPOC proof is smaller than the PTTP proof, because of left merge paths (resolutions with C–literals). In cases where the proof sizes are the same, PTTP runs between 2 and 4 times faster than ALPOC. The number of inferences that ALPOC uses to search for a proof is never greater than PTTP by 20%, but is frequently less by significant factors, up to 50,000 in the table.

References

[HS94] J. D. Horton and Bruce Spencer, Clause Trees and Factor Paths, TR94-088, Faculty of Computer Science, University of New Brunswick, 1994.

[HS95] J. D. Horton and Bruce Spencer, Clause Trees: A Tool for Understanding and Implementing Resolution in Automated Reasoning, TR95-095, Faculty of Computer Science, University of New Brunswick, 1995, submitted for publication.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle *J. ACM,* 12: 23–41, 1965.

[Sho76] R. E. Shostak, Refutation Graphs, *Artificial Intelligence,* 7 (1) 51-64, 1976.

[Spe91] B. Spencer, Linear resolution with ordered clauses. In J. Lobo, D. Loveland and A. Rajasekar, ed., *Proceedings ILPS Workshop – Disjunctive Logic Programming* (1991).

[Spe93] B. Spencer, The ordered clause restriction of model elimination and SLI resolution. In Dale Miller, editor, *Proceedings of the International Symposium of Logic Programming,* Vancouver, Canada (MIT Press, 1993) 678.

[Sti88] M. Stickel, A prolog technology theorem prover: implemented by an extended prolog compiler, *J. of Automated Reasoning,* 1 (4) 353-380, 1988.

[Sti92] M. Stickel, A Prolog technology theorem prover: implementation by an extended Prolog compiler, *Theoretical Computer Science,* 104, 109-128, 1992.

[SSY94] G. Sutcliffe, C. Suttner and T. Yemenis, The TPTP problem library. In *Proceedings of the International Conference on Automated Deduction*, 1994.

Appendix A

Automated Theorem Proving (ATP) Batch System

File Name	Directory
batch.pl	ai14/kfrancis/batch
command_file	ai14/kfrancis/batch
convert.pl	ai14/kfrancis/batch
do_arg_list.pl	ai14/kfrancis/batch
Makefile	ai14/kfrancis/batch
summarize.pl	ai14/kfrancis/batch
hour_min_sec.pl	ai14/kfrancis/pttp
process_id.c	ai14/kfrancis/pttp
pttp_alpoc.pl	ai14/kfrancis/pttp
pttp.pl	ai14/kfrancis/pttp
treedraw	ai14/sharpe/treedraw
extract_problem.pl	ai14/kfrancis/Process
problem_synopsis.pl	ai14/kfrancis/Process
stickels.pl	ai14/kfrancis/Process

How to use the ATP Batch System

System Set-Up

Makefile:

The 'Makefile' updates all parts of the batch system (ie. recompiles or creates files, as necessary). It can be invoked from within the directory in which it is located (specified above), by typing the word 'make' (without the qoutes).

Command file:

Each job or task is presented to the ATP batch system as an individual command file. The command file specifies system information (such as the name of the processor) in the form of prolog facts.

Only one command file can be sent to the system. Multiple command files can be created, but the system must be invoked separately for each one. The actual filename of a command file can be anything.

The command file specified in the file list above can serve as a guide. Creating a new command file can be made easier by copying the file '/ai14/kfrancis/batch/command_file' to a directory with appropriate read-write file permissions, and modifying the copy to suit individual requirements.

The prolog facts that must be present in the command file are listed below (the lines beginning with '%' are comments for explanation purposes and do not have to be present in command files used in the system).

```
select_data_set('PRV001-1').
% These lines specify the name of problems to be processed
% by the system.  One or more problems can be specified in
% a command file; there must be a fact such as above for each
% problem.
% Collections of problems may be specified from certain files.
% All of stickels problems are selected with the following:
% select_data_set(DataSet):-
%        ensure_loaded('/ai14/kfrancis/Process/stickels.qof'),
%        stickels(DataSet).
%
% Problems may be selected using properties as specified in
% the 'ProblemSynopsis' file accompanying tptp problems.  In
% the following example, only propositional problems with at
% least 20% non-Horn clauses and no more than 4 literals-per-
% clause (L_C) will be selected:
% select_data_set(DataSet):-
%   ensure_loaded('/ai14/kfrancis/Process/problem_synopsis.qof'),
%   problem_synopsis(DataSet, Number_of_clauses, Percent_NonHorn,
%                    Percent_Unit, Percent_Equality, Prop_or_NP,
%                    Lit_per_clause),
%   Prop_or_NP = 'Pr',
%   Percent_NonHorn >= 20.0,
%   Lit_per_clause =< 4.0.

path_data_file_directory(
      '/ai14/bspencer/tptp/TPTP-v1.1.1/Problems').
% This specifies the name of the directory containing the
% problem files, in their original form.  The directory
% given as an example is the location of the tptp problem
% files, version 1.1.1.

generate_data_file('prolog +1 /ai14/kfrancis/batch/convert').
% The original problem files may have to be converted into
% a particular form acceptable to the processor.  For example,
% tptp problem files must be converted into pttp format before
% they can be processed by pttp or alpoc.  The command given
% in this prolog fact causes a convert program to run; this
% program takes the original problem file and creates a new
% file.  This new file is based upon the original problem
% file, but is in pttp format.

processor_command('prolog +1 /ai14/kfrancis/pttp/pttp_alpoc.qof').
% This prolog fact must contain the command used to invoke the
% system processor.  There are currently two processors in use
% within the ATP batch system: pttp and alpoc, theorem provers.
% To process the problem using the alpoc theorem prover, use
% the command exactly as given.  To process the problem using
% pttp, replace 'pttp_alpoc' with 'pttp' in the command above.

command_line_parameter(_, count_inferences).
command_line_parameter(_, dont_print_proof).
% The information given in the 'command_line_parameter' facts
% is used by the system when the processor is invoked.  This
% information controls certain aspects of how the processor
% functions.  The underscore in each command_line_parameter
% fact must be present (it was originally intended that
% certain information would be given in place of the
% underscore, but the system has not yet been set up to
% handle this).
```

```
% These first two parameters determine whether or not
% inferences will be counted during processing, and whether
% or not a proof will be printed.  These two parameters MUST
% precede all other parameters given; but it doesn't matter
% whether the inference count parameter precedes the print
% proof parameter or vice versa.
% To cause the processor to print the proof, use 'print_proof'
% (without the goutes) in the fact.  Using 'dont_print_proof'
% (without the quotes) will direct the processor NOT to print
% the proof.  Similarily, to count inferences, use
% 'count_inferences' (without quotes), or use
% 'dont_count_inferences' (without quotes) to prevent inference
% counting.
% There should be only one prolog fact for inference count, and
% one for printing the proof.

command_line_parameter(_, prove_this).
command_line_parameter(_, halt).
% These two facts MUST always be in this order ('prove_this'
% must be first, followed by 'halt'), and must be the last
% command_line_parameter facts given (print proof and count
% inference information must come before these facts).
% The first parameter is the command which causes processor
% execution to begin.  In the case of the theorem provers
% alpoc and pttp, 'prove_this' is the command which begins
% execution.
% The second parameter is the command to stop the execution
% of the processor after the desired output has been produced.
% For the theorem provers alpoc and pttp (both written in
% prolog), 'halt' is the command which stops the execution
% of the processor ('halt' is the command to leave the prolog
% environment).

cpu_time_limit(20000).  %seconds
% Any batch job may be given a time limit; if the job
% is not complete when the time limit has been reached,
% processing will halt and a mail message will be sent
% by the system, saying that the time limit was reached.
% This prolog fact specifies a time limit, in seconds,
% for the processing of each individual problem.

data_file_suffix('.pttp').
% This fact specifies the file suffix to be used for
% the problem files created by the conversion program.
% Recall from above that the conversion program converts
% the original problem file into a form compatible with
% the processor being used.  If theorem provers pttp and
% alpoc are used, the problem is converted into pttp
% format (pttp format is used by both alpoc and pttp theorem
% provers); thus the suffix '.pttp' was chosen.  The
% original problem name is concatenated to this suffix
% to give the name of the new problem file used by the
% processor.
% Any file suffix can be used here, but care should be
% taken to choose a file suffix which will prevent naming
% conflicts with other files in the chosen output directory
% (see below).

output_file_suffix('.alpocV1.1').
% The ATP batch system will concatenate this file suffix
```

```
% to the name of the problem being processed to get a
% filename.  A file with this filename will be created
% and used for system output.
% Any file suffix can be chosen, but care should be
% taken to choose a suffix which, when used with the
% problem name, will give a filename which is not already
% in use in the output directory (see below).

output_file_path('/ai14/kfrancis/tptp/TPTP-v1.1.1/Infer/').
% This specifies the directory in which the system will place
% output files.  Any temporary files which are created, used,
% and then deleted by the system will also be placed in this
% this directory.  (For example, the pttp version of the problem
% file (produced by conversion) is placed in this directory.)
% WARNING: The system user should ensure that he/she has
% read-write access in the directory specified as the output
% directory.
% NOTE: There must be a forward slash at the end of the output
% file path.  For illustration:
%       output_file_path('/ai14/kfrancis').     % WRONG!
%       output_file_path('/ai14/kfrancis/').    % OK.
```

A separate batch job is created for each problem specified in the command file. The information presented by the other prolog facts in the command file applies to each and every problem.

Although multiple problems can be specified in a single command file, only one processor can be used with each command file. Therefore, to run both theorem provers (pttp and alpoc) on a given problem, the system would have to be used on two separate occasions; once with a command file specifying alpoc as processor, and once with a command file specifying pttp as processor.

****

Execution

Execution of the ATP batch system begins when a command file and the file 'batch.pl' are loaded with prolog by the user, with the following command:

```
prolog +l command_file_name +l batch.pl
```

The system can be executed from any directory. The only two files loaded directly by the user are the command file and the file 'batch.pl'. These files may be located in the directory from which the system is invoked, or they may be in different directories. If the files are not located in the same directory, directory paths must be used.

ex. prolog +l /ai14/com_file +l /ai14/kfrancis/batch/batch.pl

The command file must be loaded first, because the file 'batch.pl' uses information contained in the command file.

If the system is successful in setting up and sending the batch job, several lines beginning with '%' will appear on the standard output. The last line should give a job number, and

the date. The other lines simply indicate files loaded and compiled in prolog as part of the system.

IF THERE IS AN ERROR...

If there is an error in the command file or in the command line as typed, the system will stop in the prolog environment. To return to the system prompt, type 'halt.' (without the quotes, but including the period).

A mail message is sent by the system when the batch job is complete. The mail message will list files loaded and compiled by prolog as part of the system, and temporary files created and used by the system. If there were no errors, and the problem could be processed within the time limit specified, the output files should be in the output directory as specified in the command file (in prolog fact called 'output_file_path').

If the problem could not be processed within the time limit specified in the command file, the mail message will finish with the following:

```
[ Fault: CPU time limit exceeded ]
! Execution aborted
```

There will still be an output file in the specified output directory, but the file will not contain a time or inference count for the problem.

WARNING:

Problems may arise when several command files specify the same 'input problem - output suffix - output directory' sets. If such command files were processed at nearly the the same time, the resulting batch jobs could be executed nearly concurrently. Several different batch jobs could be writing to a single output file.

****

Summary

A table summary of inference counts, timings, or level numbers may be prepared based upon selected groups of files, using the prolog program 'summarize.pl'.

The program must be executed within the directory where the files providing the data for the table are located. The summary is created and sent to standard output by this command:

```
prolog +l summarize +z Time .suffix1 .suffix2
```

'Time' may be replaced by 'Inference' or by 'Level'. Only one kind of information appears in each table (ie. timings and inference counts are not compared in a single table). '.suffix1' and '.suffix2' correspond to the suffixes of the data files to be compared, ie. indicate the groups of files being summarized.

ex. Files are: COM001-1.alpoc COM001-1.pttp

```
                        SET001-1.alpoc    SET001-1.pttp   etc.

            To compare times for a given problem, '.alpoc' group
            versus '.pttp' group, type:
             prolog +1 summarize +z Time .out_alpoc .out_pttp

        ex.  To create a table of inference counts for files ending
             in '.outa' and '.outb', and to store this table in a
             new file called 'MyTable', one would type the following:
             prolog +1 summarize +z Inference .outa .outb > MyTable
```

****

Tree drawing

The C++ program 'treedraw' draws a tree diagram based upon an input proof. Input is a text file containing a (Stickel like) proof. The program is invoked by:

```
/ai14/sharpe/treedraw/treedraw input_file_name options
```

Options allow the user to modify the following defaults:

```
T : Top edge labelled (default value).
B : Bottem edge labelled.
P : page width, followed immeadiatly by number
    (default is 80 columns).
W : width per node, followed immeadiatly by number
    (default depends on input tree data).
L : lines per predicate node, followed immeadiatly by number.
    (0 means no predicate displayed, default depends on input tree).
```

Note that options must occur contiguosly.
For example, valid options are : BW5L3.

Appendix B Empirical Results

All of the experimental results are collected in the following sections.

B.1 Timing Results

Problem	.alpocV1.1		.pttpV1.1	
COM001-1	0.000		0.016	
COM002-2	0.300	(4.48)	0.067	
EX6-T1	0.000		0.000	
EX6-T2	9.100		50.266	(5.52)
GRA001-1	0.017		0.000	
GRP001-1	31.950	(2.69)	11.883	
GRP003-1	0.017	(1.06)	0.016	
GRP004-1	0.017	(1.06)	0.016	
GRP005-1	0.000		0.000	
GRP006-1	0.017		0.017	
GRP007-1	0.000		0.000	
GRP008-1	639.433		11522.000	(18.02)
GRP009-1	3.600	(2.54)	1.417	
GRP010-1	0.083	(1.66)	0.050	
GRP012-1	0.217	(1.85)	0.117	
GRP012-2	573.033	(3.01)	190.684	
GRP013-1	2.883	(2.37)	1.217	
GRP022-1	0.034	(2.00)	0.017	
GRP028-1	0.000		0.016	
GRP029-1	0.267		13.134	(49.19)
GRP030-1	0.083		1.917	(23.10)
GRP031-1	0.167	(2.49)	0.067	
GRP031-2	0.033	(1.94)	0.017	
GRP032-3	0.000		0.017	
GRP033-3	0.000		0.017	
GRP034-3	0.033	(1.94)	0.017	
GRP034-4	0.016		0.017	(1.06)
GRP036-3	0.383	(2.09)	0.183	
GRP037-3	7.300	(2.39)	3.050	
GRP038-3	0.000		0.000	
LCL181-2	0.000		0.000	
LCL230-2	0.000		0.000	
MSC001-1	34.450		60.883	(1.77)
MSC002-1	0.800	(4.79)	0.167	
MSC003-1	0.050		0.050	
MSC004-1	0.483		76.484	(158.35)
MSC007-1.002	0.000		0.000	
MSC007-1.003	0.017		0.000	
MSC007-1.004	0.017		0.033	(1.94)
MSC007-1.005	0.233		NA	(too large)
MSC007-1.006	2.083		NA	(too large)
MSC007-1.007	19.667		NA	(too large)
MSC007-1.008	257.083		NA	(too large)
NUM001-1	0.900	(1.86)	0.483	
NUM002-1	0.333	(1.82)	0.183	
NUM014-1	0.000		0.000	
NUM015-1	0.917	(1.28)	0.717	
NUM016-1	0.083	(2.52)	0.033	
NUM016-2	0.134	(4.06)	0.033	
NUM019-1	0.016		0.000	
NUM023-1	0.000		0.000	
NUM024-1	14.017	(2.18)	6.417	
NUM025-1	0.000		0.000	

NUM027-1	25.117	(2.21)	11.384	
PRV001-1	19.166		8133.300	(424.36)
PUZ009-1	0.000		0.000	
PUZ013-1	0.000		0.000	
PUZ014-1	0.017	(1.06)	0.016	
PUZ023-1	2.550		20.733	(8.13)
PUZ024-1	0.183	(2.20)	0.083	
PUZ025-1	63.250		4448.900	(70.34)
PUZ029-1	0.800	(1.92)	0.417	
PUZ033-1	0.017		0.000	
RNG001-2	3306.450		NA	(OVER 30000)
RNG001-3	1.900		14.284	(7.52)
RNG001-5	2186.350	(3.82)	572.617	
RNG005-2	0.033		0.000	
RNG006-2	0.083	(4.88)	0.017	
RNG037-2	0.083	(2.52)	0.033	
RNG038-2	0.067	(3.94)	0.017	
RNG040-2	69.567	(2.29)	30.383	
RNG041-1	3.550	(2.45)	1.450	
SET001-1	0.000		0.000	
SET002-1	0.750	(2.25)	0.333	
SET003-1	0.017		0.000	
SET004-1	0.016		0.000	
SET006-1	0.017		0.000	
SET008-1	0.100	(2.00)	0.050	
SET009-1	8.766	(2.20)	3.983	
SET043-5	0.000		0.016	
SET044-5	0.016		0.017	(1.06)
SET046-5	0.000		0.334	
SYN001-1.002	0.000		0.000	
SYN001-1.003	0.000		0.000	
SYN001-1.004	0.033		0.117	(3.55)
SYN001-1.005	0.066		1119.250	(16958.33)
SYN008-1	0.000		0.000	
SYN011-1	0.017		0.000	
SYN014-2	0.166	(2.52)	0.066	
SYN028-1	0.017		0.000	
SYN029-1	0.000		0.000	
SYN030-1	0.000		0.016	
SYN031-1	0.017		0.083	(4.88)
SYN032-1	0.016		0.016	
SYN033-1	0.000		0.000	
SYN034-1	0.016		0.650	(40.62)
SYN035-1	0.000		0.000	
SYN038-1	24272.200		NA	(OVER 30000)
SYN044-1	0.000		0.000	
SYN045-1	0.000		0.000	
SYN047-1	0.000		0.000	
SYN052-1	0.000		0.000	
SYN055-1	0.033	(1.94)	0.017	
SYN060-1	0.017		0.000	
SYN061-1	0.000		0.000	
SYN066-1	0.000		0.000	
SYN081-1	0.033	(1.94)	0.017	
SYN082-1	0.050		25.267	(505.34)
TOP005-2	75.233		NA	(OVER 20000)
s03PRIME	0.867	(1.58)	0.550	
s06ances2	0.017		0.000	
s07NUM1	0.017		0.017	
stark035	0.184		7.600	(41.30)

B.2 Inference Counts

26-Aug-94

Problem	.alpocV1.1		.pttpV1.1	
COM001-1	35		35	
COM002-2	690		690	
EX6-T1	4		4	
EX6-T2	31868		556894	(17.48)
GRA001-1	40		49	(1.23)
GRP001-1	100839	(1.02)	98881	
GRP003-1	24		105	(4.38)
GRP004-1	24		24	
GRP005-1	4		4	
GRP006-1	17	(1.06)	16	
GRP007-1	12		12	
GRP008-1	1260198		NA	(time 11522 sec.)
GRP009-1	13204	(1.02)	13008	
GRP010-1	378		378	
GRP012-1	856		856	
GRP012-2	1504907	(1.06)	1421095	
GRP013-1	9300	(1.01)	9191	
GRP022-1	199		199	
GRP028-1	5		5	
GRP029-1	1134		138528	(122.16)
GRP030-1	402		20332	(50.58)
GRP031-1	736	(1.00)	733	
GRP031-2	187		187	
GRP032-3	6		6	
GRP033-3	26		26	
GRP034-3	113	(1.01)	112	
GRP034-4	26	(1.08)	24	
GRP036-3	1297	(1.00)	1293	
GRP037-3	23898	(1.01)	23549	
GRP038-3	12		12	
LCL181-2	6		6	
LCL230-2	3		3	
MSC001-1	104203		328492	(3.15)
MSC002-1	2059	(1.19)	1729	
MSC003-1	94		242	(2.57)
MSC004-1	1032		260936	(252.84)
MSC007-1.002	2		2	
MSC007-1.003	15		16	(1.07)
MSC007-1.004	99		491	(4.96)
MSC007-1.005	674		NA	(too large)
MSC007-1.006	4975		NA	(too large)
MSC007-1.007	40527		NA	(too large)
MSC007-1.008	365330		NA	(too large)
NUM001-1	1924	(1.00)	1923	
NUM002-1	719		719	
NUM014-1	24		24	
NUM015-1	1623		3830	(2.36)
NUM016-1	226		231	(1.02)
NUM016-2	187		191	(1.02)
NUM019-1	9		9	
NUM023-1	2		2	
NUM024-1	27869	(1.01)	27555	
NUM025-1	4		4	

NUM027-1	47128		48418	(1.03)
PRV001-1	47723		NA	(too large)
PUZ009-1	18		45	(2.50)
PUZ013-1	26		27	(1.04)
PUZ014-1	45		127	(2.82)
PUZ023-1	5419		109233	(20.16)
PUZ024-1	458		482	(1.05)
PUZ025-1	125247		21363017	(170.57)
PUZ029-1	1654		4699	(2.84)
PUZ033-1	19		28	(1.47)
RNG001-3	6230		131224	(21.06)
RNG001-5	5529364	(1.16)	4777282	
RNG005-2	94		94	
RNG006-2	253	(1.02)	249	
RNG037-2	212		212	
RNG038-2	154		154	
RNG040-2	162762		165983	(1.02)
RNG041-1	8791		8828	(1.00)
SET001-1	8		8	
SET002-1	1794		2249	(1.25)
SET003-1	35		35	
SET004-1	34		34	
SET006-1	35		35	
SET008-1	247		254	(1.03)
SET009-1	16628		23409	(1.41)
SET043-5	3		8	(2.67)
SET044-5	26		60	(2.31)
SET046-5	37		2464	(66.59)
SYN001-1.002	6		8	(1.33)
SYN001-1.003	27		87	(3.22)
SYN001-1.004	113		2892	(25.59)
SYN001-1.005	465		26790196	(57613.32)
SYN008-1	5		5	
SYN011-1	10		14	(1.40)
SYN014-2	407		422	(1.04)
SYN028-1	7		7	
SYN029-1	7		7	
SYN030-1	13		36	(2.77)
SYN031-1	53		550	(10.38)
SYN032-1	11		27	(2.45)
SYN033-1	5		5	
SYN034-1	44		4510	(102.50)
SYN035-1	14		14	
SYN044-1	11		13	(1.18)
SYN045-1	4		4	
SYN047-1	6		6	
SYN052-1	7		21	(3.00)
SYN055-1	81		235	(2.90)
SYN060-1	12		12	
SYN061-1	7		7	
SYN066-1	8		8	
SYN081-1	27		84	(3.11)
SYN082-1	79		131071	(1659.13)
TOP005-2	90740		NA	(too large)
s03PRIME	1941		3953	(2.04)
s06ances2	11		19	(1.73)
s07NUM1	22		22	
stark035	399		50089	(125.54)

B.3 Search Level Results

26-Aug-94

Problem	.alpocV1.1	.pttpV1.1	
COM001-1	5	5	
COM002-2	11	11	
EX6-T1	0	0	
EX6-T2	10	14	(1.40)
GRA001-1	0	0	
GRP001-1	9	9	
GRP003-1	6	9	(1.50)
GRP004-1	6	6	
GRP005-1	3	3	
GRP006-1	6	6	
GRP007-1	2	2	
GRP008-1	10	12	(1.20)
GRP009-1	8	8	
GRP010-1	6	6	
GRP012-1	6	6	
GRP012-2	11	11	
GRP013-1	7	7	
GRP022-1	5	5	
GRP028-1	3	3	
GRP029-1	6	9	(1.50)
GRP030-1	6	9	(1.50)
GRP031-1	6	6	
GRP031-2	6	6	
GRP032-3	3	3	
GRP033-3	3	3	
GRP034-3	6	6	
GRP034-4	6	6	
GRP036-3	6	6	
GRP037-3	7	7	
GRP038-3	3	3	
LCL181-2	0	0	
LCL230-2	0	0	
MSC001-1	12	13	(1.08)
MSC002-1	10	10	
MSC003-1	7	9	(1.29)
MSC004-1	12	23	(1.92)
MSC007-1.002	0	0	
MSC007-1.003	0	0	
MSC007-1.004	0	0	
MSC007-1.005	0	NA	
MSC007-1.006	0	NA	
MSC007-1.007	0	NA	
MSC007-1.008	0	NA	
NUM001-1	6	6	
NUM002-1	6	6	
NUM014-1	5	5	
NUM015-1	10	11	(1.10)
NUM016-1	6	6	
NUM016-2	7	7	
NUM019-1	2	2	
NUM023-1	1	1	
NUM024-1	7	7	
NUM025-1	2	2	
NUM027-1	8	8	

PRV001-1	13	19	(1.46)
PUZ009-1	0	0	
PUZ013-1	0	0	
PUZ014-1	0	0	
PUZ023-1	9	12	(1.33)
PUZ024-1	6	6	
PUZ025-1	13	17	(1.31)
PUZ029-1	20	24	(1.20)
PUZ033-1	0	0	
RNG001-2	14	NA	
RNG001-3	14	18	(1.29)
RNG001-5	11	11	
RNG005-2	4	4	
RNG006-2	5	5	
RNG037-2	5	5	
RNG038-2	5	5	
RNG040-2	7	7	
RNG041-1	6	6	
SET001-1	3	3	
SET002-1	11	11	
SET003-1	4	4	
SET004-1	4	4	
SET006-1	4	4	
SET008-1	6	6	
SET009-1	11	11	
SET043-5	1	2	(2.00)
SET044-5	3	4	(1.33)
SET046-5	4	9	(2.25)
SYN001-1.002	0	0	
SYN001-1.003	0	0	
SYN001-1.004	0	0	
SYN001-1.005	0	0	
SYN008-1	0	0	
SYN011-1	0	0	
SYN014-2	5	5	
SYN028-1	0	0	
SYN029-1	0	0	
SYN030-1	0	0	
SYN031-1	3	4	(1.33)
SYN032-1	0	0	
SYN033-1	3	3	
SYN034-1	4	9	(2.25)
SYN035-1	4	4	
SYN038-1	21	NA	
SYN044-1	0	0	
SYN045-1	0	0	
SYN047-1	0	0	
SYN052-1	1	2	(2.00)
SYN055-1	7	11	(1.57)
SYN060-1	3	3	
SYN061-1	2	2	
SYN066-1	3	3	
SYN081-1	3	4	(1.33)
SYN082-1	5	13	(2.60)
TOP005-2	16	NA	
s03PRIME	10	11	(1.10)
s06ances2	0	0	
s07NUM1	5	5	
stark035	8	12	(1.50)