

# Strategy Parallelism in Automated Theorem Proving

Andreas Wolf & Reinhold Letz

Institut für Informatik der Technischen Universität München

D-80290 München

Germany

{wolfa,letz}@informatik.tu-muenchen.de

## Abstract

Automated theorem provers use search strategies. Unfortunately, there is no unique strategy which is uniformly successful on all problems. A combination of more than one strategy increases the chances of success. Limitations of resources as time or processors enforce efficient use of these resources by partitioning the resources adequately among the involved strategies. This leads to an optimization problem. We describe this problem for general search problems and discuss in more detail an application in automated theorem proving.

## Introduction

A search problem is typically solved by applying a *uniform* search procedure. In automated deduction, different search strategies may have a strongly different behavior on a given problem. Unfortunately, in general, it cannot be decided in advance which strategy is the best for a given problem. This motivates the *competitive* use of different strategies. In order to be successful with such an approach, the strategies must satisfy the following two intuitively given conditions.

1. For a given set of problems, the function  $f(t) = \frac{|s(t)|}{t}$ , where  $s(t)$  is the set of problems solved within time  $t$  is sub-linear, i.e., with each new time interval less new problems are solved.
2. The strategies must be *complementary* wrt. a given problem set and a given time limit, i.e., they have to solve different sets of problems, or, at least, the sets of solved problems (wrt. a given problem set) must differ "significantly".

If both conditions are satisfied, then a competitive use of different strategies can be more successful than the best single strategy. A convincing illustration of this fact is the unexpected success of the theorem prover Gandalf at the theorem prover competition (CASC-14 Homepage 1997) held in summer 1997 at CADE-14, which applies this method by time slicing different strategies. The first condition is typically satisfied in automated theorem proving whereas the sec-

ond condition has not been in the focus of automated deduction research so far.

In order to study the possibilities for increasing the performance of automated theorem provers, we implemented p-SETHEO, an experimental framework for the control of the parallel execution of different prover strategies; and we tested this framework using the automated theorem prover SETHEO (Moser *et al.* 1997). This paper, and especially the success of p-SETHEO, shows that it is worthwhile to develop methods for achieving complementary strategies, which is left to future research.

In this paper we concentrate on the problem of determining an optimal selection of strategies from a fixed set of given strategies. Such a method could be useful in practice, as follows. A non specialist user of a theorem proving system could adapt the system to his specific domain of problems by selecting a representative *training* set of problems and computing an optimal set of competitive strategies for the training set. If the training set was representative for the domain, then the computed strategies will also perform well on the whole domain. Concrete applications we envisage are interactive proof systems like ILF (Dahn *et al.* 1997), Isabelle (Paulson 1994) or KIV (Reif 1992), which use or intend to use automated theorem provers as subroutines.

## Strategy Parallelism and Related Approaches

The selection of more than one search strategy in combination with techniques to partition the available resources (time and processors) with respect to the actual task defines a parallelization method, which we call *strategy parallelism*. (Distributed) competitive agents traverse the same search space, but in a different order. Whenever an agent finds a solution, all other agents are stopped. With this method it is intended that the strategies traverse the search space in such a manner that, in practice, the repeated consideration of the same parts can be avoided. In the pure form of strategy parallelism which is discussed in this paper, there is no interaction between the competitive agents. This enables that even completely different

---

Copyright ©1998, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

search paradigms (for example, resolution and model elimination) can be combined.

This method differs significantly from a *partitioning* of the search space (Suttner & Schumann 1994) which is done, for instance, in SPTHEO (Suttner 1997) or PARTHEO (Schumann & Letz 1990). Partitioning *guarantees* that no part of the search space is explored more than once. However, partitioning has the main disadvantage, that completeness can be guaranteed only if all agents are reliable. In contrast, strategy parallelism retains completeness as long as one agent is reliable.

Another combination of different strategies is used, e.g., within the TeamWork concept (Denzinger 1995) of DISCOUNT (Denzinger, Kronenburg, & Schulz 1997) (a combination of several completion strategies for unit equality problems which periodically exchange intermediate results) or with the Clause Diffusion concept of AQUARIUS (Bonacina & Hsiang 1993) (a resolution based prover with cooperating agents on splitted databases).

### The Complexity of Strategy Allocation

Recalling the application mentioned in the Introduction, we are faced with the following *strategy allocation problem*. Given a set of (training) problems, a set of admissible strategies, a time limit, and a number of available processors, we want to determine an optimal distribution of resources to each strategy, i.e., a combination of strategies which can solve a maximal number of problems from the training set. In order to recognize the worst-case complexity of this problem, we formulate it more precisely.

#### GIVEN

1. a set  $F = \{f_1, \dots, f_n\}$  of objects ("formulae" or "problems"),
2. a set  $S = \{s_1, \dots, s_m\}$  of functions ("strategies")  $s_i : F \rightarrow \mathbb{N}^+ \cup \{\infty\}$  ( $1 \leq i \leq m$ ) ( $\infty$  is added because the strategy  $s_i$  may be incomplete, or the problem cannot be solved in the given time),
3. nonnegative integers  $t$  (time resource),  $p$  (processors), and  $k$  (limit for the decision variant of the problem).

#### FIND

ordered pairs  $(t_1, p_1), \dots, (t_m, p_m)$  (strategy  $i$  will be scheduled for time  $t_i$  on processor  $p_i$ ) of nonnegative integers such that

1.  $\sum_{\{i:p_i=j\}} t_i \leq t$  for all  $j = 1, \dots, p$ , and
2.  $|\bigcup_{i=1}^m \{f : s_i(f) \leq t_i\}|$  is maximal (optimization variant) or  $\geq k$  (decision variant).

Let us briefly explain the abstract problem formulation. For every training formula  $f$  and every strategy  $s$ :  $s(f)$  is the time  $s$  needs to solve  $f$ . It is important

to note that the time limit is used for solving each formula separately, i.e., the time is not divided among the formulae.

In order to assess the difficulty of the problem, we have given two formulations, the decision variant (with number  $k$ ) and the optimization variant. It is evident that the (easier) decision variant (guess a resource allocation  $(t_1, p_1), \dots, (t_m, p_m)$  such that the number of solved problems is greater than or equal to a given number  $k$ ) of the problem is in NP: a given satisfying allocation  $(t_1, p_1), \dots, (t_m, p_m)$  can be verified in polynomial time.

Unfortunately, the decision variant of the problem for only a single processor is already *strongly NP-complete*. Note that a problem is strongly NP-complete if it is NP-complete even if the numbers occurring in the input are written in unary notation. For example, the *knapsack problem* (Garey & Johnson 1979) which is obviously related with our problem is not strongly NP-complete unless  $P=NP$ ; for the knapsack problem, there exists a *pseudo-polynomial* algorithm, i.e., an algorithm which is polynomial for a unary representation of the input numbers. Interestingly, if the admissible strategies are totally complementary, then the single processor variant of our problem reduces to the knapsack problem, and a standard dynamic programming algorithm with pseudo-polynomial complexity can be used, which also solves the optimization problem in pseudo-polynomial time. This motivates to strive for strategies that are as complementary as possible.

For the general single processor problem containing arbitrary strategies, however, no pseudo-polynomial algorithms are known. The strong NP-completeness of this problem can be recognized, for example, by providing a polynomial reduction of the strongly NP-complete *minimum cover problem* (Garey & Johnson 1979) to our problem.

The full problem of strategy allocation includes an additional placement of strategies on different processors as a subproblem, which is exactly the *multiprocessor scheduling problem* (Garey & Johnson 1979). The latter is also known to be strongly NP-complete. Therefore, in practice, one would give up finding optimal solutions for the full problem. One reasonable possibility is to use a gradient procedure, as described below.

### Implementation: p-SETHEO

In order to investigate the potential of strategy parallelism in practice, we have evaluated the method on different strategies of the sequential model elimination prover SETHEO (Letz *et al.* 1992; Goller *et al.* 1994; Letz, Mayr, & Goller 1994; Moser *et al.* 1997). We implemented an environment for the strategy parallel execution of theorem provers, the system p-SETHEO which is based on PVM (Geist *et al.* 1994). p-SETHEO can be configured very easily by an ASCII file containing information about the usable hosts, the

configuration	solutions	(%)	$\Sigma$ time	time/solution	$\Sigma$ work	work/solution
-dr	151	66	106128	702.8	106128	702.8
-dr with -foldup	159	69	91561	575.8	91561	575.8
-wdr	122	53	128587	1054.0	128587	1054.0
-wdr with -foldup	161	70	87321	542.4	87321	542.4
p-SETHEO	230	100	28168	122.5	112627	489.7

Table 1: Comparison of strategy competition with single strategies.

maximum load allowed (the degree of time-slicing on a processor), and the strategy allocation for the competitive strategies. Currently, all contained theorem provers are variants of SETHEO obtained by modifying the parameter settings. Because of the generic layout of the p-SETHEO controlling mechanism, new strategies and even new theorem provers can be integrated very easily. The implementation uses PERL, PVM, C, and shell tools in approximately 1000 lines of code (without SETHEO). The parallelization model of p-SETHEO works as follows.

1. Select a set of triples of strategies, computation times, and assigned processors according to the problem to be solved and the usable resources in terms of processors, processor load and time.
2. Perform the preprocessing steps needed for all selected strategies (reordering, equality treatment, lemma generation etc.) using the available parallel resources (usable processors, maximal load per host).
3. Start all prover strategies using the available parallel resources, the first prover finding a proof stops the others.

## Experiments

The practical applicability of the strategy parallelism concept has been tested on a number of examples introduced next. These tests have been performed within our research group (the combination of default strategies and the tests on the TPTP library) as well as by external cooperation partners (the examples taken from the MIZAR library). For the experiments at our location we used SUN Ultra1 computers.

### A Simple Combination of Default Strategies

The good performance of the strategy parallel approach is shown by the combination of four fixed strategies on four processors. For details on the parameters see (Letz, Mayr, & Goller 1994). We used the following parameter selections of the sequential prover: iterative deepening on tableau depth (-dr), with and without folding-up (-foldup), and the iterative deepening with the weighted depth bound (-wdr), also with and without folding-up (-foldup). The experiments were performed on the 230 problems in the TPTP v2.0.0 that

are non-trivial and solvable by one of the strategies in 1000 seconds. As non-trivial we considered all problems which cannot be solved by at least three configurations within 20 seconds. The results are shown in Table 1.

Even the best single strategy solves only 70% of the problems p-SETHEO solves using 4 processors without a task specific strategy selection; and the single strategy needs 310% of p-SETHEO's time. If we consider parallel work instead of time, p-SETHEO solves significantly more problems with nearly the same costs.

### The TPTP Problem Library

We have tested the performance of p-SETHEO and compared it to the sequential prover SETHEO. As training set  $F$  we have taken the 420 problems of the TPTP problem library (Sutcliffe, Suttner, & Yemnis 1994) that have been used by the theorem prover competition (CASC-14 Homepage 1997) at CADE-14. For a given set of 20 strategies  $S$ , the solution times  $s(f)$ , for any  $f \in F, s \in S$  were computed, with a time limit  $T$  of 300 seconds per problem. The strategies are parameter settings of SETHEO resulting from experiments within another project, and we expected that they would perform well here, too)

To select the appropriate set of strategies, a resource allocation algorithm applying a gradient method (as described next) was used. Within this experiment, we considered the case of only a single processor; that is, the strategies were time-sliced.

In order to solve the sequential strategy allocation problem, we have used the following approximation algorithm. Depending on a given fraction  $t = \frac{T}{T}$  of the time limit  $T$ , the following procedure is iteratively applied for computing the strategy allocation vector  $v = t_1, \dots, t_m$ .

1. Start with  $v := t_1, \dots, t_m$  with  $t_i = 0$ , for  $1 \leq i \leq m$ , and set  $\Delta T := t$ .
2. Let  $k$  be the number of problems solved with  $v$ . As long as  $t_1 + \dots + t_m < T$ , let  $v_1, \dots, v_m$  be the  $m$  allocation vectors obtained from  $v$  by adding  $\Delta T$  to one of its members  $t_i, 1 \leq i \leq m$ . Select one, say  $v_i$ , which solves a maximal number of problems from  $F$ . If  $v_i$  solves more problems than  $v$ , set  $v := v_i$  and  $\Delta T := t$ ; otherwise increment  $\Delta T := \Delta T + t$ . Then go to 2.

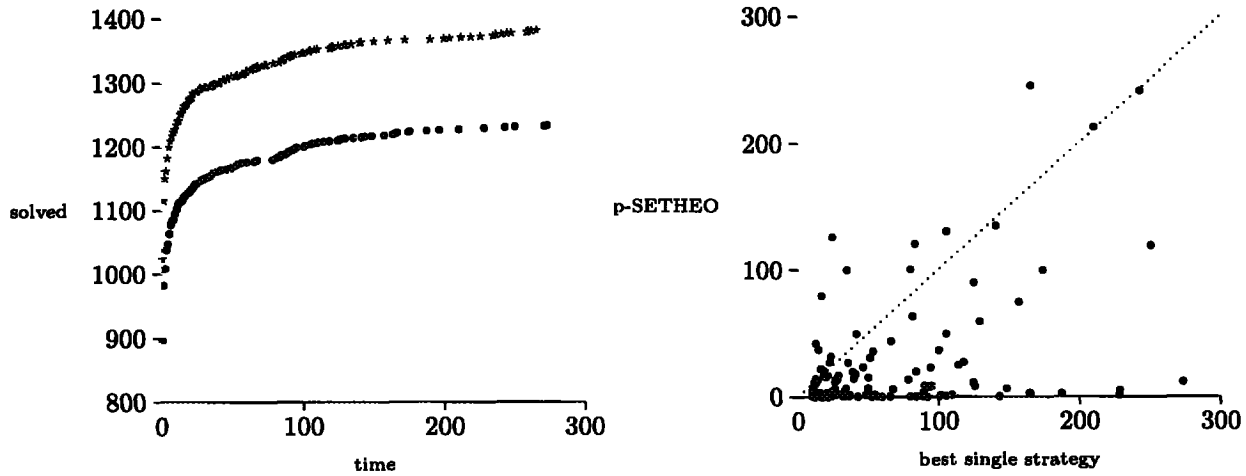


Figure 1: Performance of p-SETHEO compared to the best single strategy

Evidently, the run time of the procedure is polynomially bounded by the number of steps  $l$  and the size of the input (with numbers represented short). On the other hand, the procedure does obviously not always find the optimal strategy configuration. In order to improve the result, we have run the procedure several times with different  $l$ . Note also that not even the admissible strategy set was selected with respect to achieving high complementarity.

The strategy set determined that way was tested on the part of the TPTP v.2.0.0 in clausal logic with 300 seconds CPU time limit on a single processor and compared with the best single strategy. The result is shown in the Figure 1. In the left picture the ● marks the number of problems solved by the best single strategy within a certain time, whereas the ★ shows the problems solved by p-SETHEO. In the right picture the run-times of p-SETHEO and the best strategy are compared on all non-trivial (but solvable by p-SETHEO) problems. Each ● represents a problem; within the area below the dotted line p-SETHEO is faster than the single strategy. Within both diagrams, we compared only the pure proof times excluding pre-processing and compile times and the spawn times of the PVM. Table 2 compares proof times (in seconds) of p-SETHEO with the best single strategy on some selected problems. Note that the single strategy can solve 1233 problems within a maximum of 300 seconds per problem, whereas p-SETHEO reaches this number within only 11 seconds. So p-SETHEO is particularly suited if short response times are required.

### The ILF Interactive Proof Environment

Within the ILF system (Dahn *et al.* 1997) a simplified version of p-SETHEO was compared with other automated theorem provers on 97 problems extracted from the MIZAR library (Dahn & Wernhard 1997).

The time limit for each proof attempt was 15 sec-

Problem	single	p-SETHEO	speed-up
BOO004-4	11.86	0.05	237.2
BOO005-4	84.42	0.13	649.4
BOO006-4	56.78	0.10	567.8
BOO010-2	126.10	7.86	16.0
CAT002-3	143.61	0.15	957.4
CAT003-3	31.72	0.12	264.3
CAT005-3	124.78	10.95	11.4
CAT010-4	32.87	0.26	126.4
CAT019-3	42.90	0.06	715.0
GRP010-4	92.13	0.04	2303.2

Table 2: Speed-ups for selected problems from the TPTP.

Gandalf	47	SETHEO	53
OTTER	60	SPASS	52
p-SETHEO	76	all together	94

Table 3: Results on some MIZAR problems.

onds CPU time. Table 3 shows the number of solved problems. Some of these numbers have been improved later by direct support of the prover authors. Here we give the results reached using only the automatic configuration capabilities of the involved theorem provers.

### Assessment and Future Work

We have investigated the problems and perspectives of strategy parallelism. Our experiments demonstrate that one can significantly increase the performance of theorem provers, even with a very simple strategy allocation algorithm and a non-optimized set of admissible strategies. While, in theorem proving, often the system developer or advanced user can tune (the parameters

of) his system to a given problem by using his experience, this is not possible if the theorem prover is integrated into a larger proof environment like ILF (Dahn *et al.* 1997). In this case the configuration *must* be done automatically, and strategy parallelism is a good solution of this problem.

In this paper, we did not discuss a number of issues of high importance for strategy parallelism, which describe future research topics.

1. How do we obtain a set of strategies which solve many problems and differ as much as possible?
2. Often strategies are successful for a certain class of problems. For example, unit equality problems need different treatment than problems without equality. If such features can be identified, the selection of strategies can be made more specific and hence more successful.
3. The number of sensible strategies which are successful and differ as much as possible seems to be bounded. This restricts the scalability of strategy parallelism to large platforms of parallel processors. Can this problem be overcome, i.e., can we find a systematic method for producing as many successful and differing strategies as we want? Probably such a method needs to contain randomized parts.
4. The success of the selected strategies depends on the given training set. How do we obtain a training set which is representative for the considered domain of problems?
5. Furthermore, possible improvements of the strategy allocation algorithm should be investigated.
6. Up to now, the different strategies competing for the solution of a given problem do not communicate. It is imaginable and desirable that the strategies periodically report on their proof attempt as, e.g., inference rates, actual performance of their hosting processor, or expected time for the completion of a certain search level. These informations are usable for on-line evaluation of strategies. That way actually bad performing strategies may be replaced, and strategies on badly performing processors are able to migrate. So strategy parallelism can become a kind of cooperative theorem proving.

### Acknowledgments

This work is supported by the Deutsche Forschungsgemeinschaft within the Sonderforschungsbereich 342, subproject A5 (PARIS). Thanks to Ortrun Ibens for her support in performing the experiments.

### References

Bonacina, M. P., and Hsiang, J. 1993. Distributed Deduction by Clause Diffusion: The Aquarius Prover. In *Proc. DISCO-93*, volume 722 of *LNCS*, 272–287. Springer.

CASC-14 Homepage. 1997. <http://www.cs.jcu.edu.au/~tptp/casc-14/>.

Dahn, B. I., and Wernhard, C. 1997. First Order Proof Problems Extracted from an Article in the MIZAR Mathematical Library. In *Proc. FTP-97*, 58–62. RISC Linz, Austria.

Dahn, B. I.; Gehne, J.; Honigmann, T.; and Wolf, A. 1997. Integration of Automated and Interactive Theorem Proving in ILF. In *Proc. CADE-14*, volume 1249 of *LNAI*, 57–60. Springer.

Denzinger, J.; Kronenburg, M.; and Schulz, S. 1997. DISCOUNT. A Distributed and Learning Equational Prover. *JAR* 18(2):189–198.

Denzinger, J. 1995. Knowledge-Based Distributed Search Using Teamwork. In *Proc. ICMAS-95*, 81–88. AAAI-Press.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman.

Geist, A.; Beguelin, A.; Dongarra, J.; Jiang, W.; Manchek, R.; and Sunderam, V. 1994. *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press.

Goller, C.; Letz, R.; Mayr, K.; and Schumann, J. 1994. SETHEO V3.2: Recent Developments. In *Proc. CADE-12*, volume 814 of *LNAI*, 778–782. Springer.

Letz, R.; Schumann, J.; Bayerl, S.; and Bibel, W. 1992. SETHEO: A High-Performance Theorem Prover. *JAR* 8(2):183–212.

Letz, R.; Mayr, K.; and Goller, C. 1994. Controlled Integration of the Cut Rule into Connection Tableau Calculi. *JAR* 13(3):297–338.

Moser, M.; Ibens, O.; Letz, R.; Steinbach, J.; Goller, C.; Schumann, J.; and Mayr, K. 1997. SETHEO and E-SETHEO. The CADE-13 Systems. *JAR* 18(2):237–246.

Paulson, L. C. 1994. *Isabelle: A Generic Theorem Prover*, volume 828 of *LNCS*. Springer.

Reif, W. 1992. The KIV System: Systematic Construction of Verified Software. In *Proc. CADE-11*, volume 607 of *LNAI*, 753–757. Springer.

Schumann, J., and Letz, R. 1990. PARTHEO: a High Performance Parallel Theorem Prover. In *Proc. CADE-10*, volume 449 of *LNAI*, 44–56. Springer.

Sutcliffe, G.; Suttner, C.; and Yemenis, T. 1994. The TPTP Problem Library. In *Proc. CADE-12*, volume 814 of *LNAI*, 252–266. Springer.

Suttner, C., and Schumann, J. 1994. Parallel Automated Theorem Proving. In *Parallel Processing for Artificial Intelligence*, 209–257. Elsevier.

Suttner, C. 1997. SPTHEO. A Parallel Theorem Prover. *JAR* 18(2):253–258.