Randomization and Restarts in Proof Planning

Andreas Meier[1] Carla P. Gomes[2] Erica Melis[1]

[1] Fachbereich Informatik
Universität des Saarlandes
66041 Saarbrücken, Germany
{ameier|melis}@ags.uni-sb.de

[2] Computer Science Department
Cornell University
Ithaca, NY 14853, USA
gomes@cs.cornell.edu

Abstract. Proof planning is an application of AI-planning in mathematical domains. One of its strengths comes from the usage of mathematical knowledge that heuristically restricts the search space.
We investigate problem classes for which little or no heuristic control knowledge is available and test the usage of randomization and restart techniques for "search control". This approach does not rely on domain-specific control knowledge but takes advantage of the surprising diversity of proofs in terms of the size and style.

1 Introduction

Proof planning considers mathematical theorem proving as a planning problem. Proof planning domains have many properties in common with deterministic real world planning domains which require high branching, HTN, constraint solving, complicated knowledge representation and acquisition, and forward and backward reasoning.

Proof planning has enabled the derivation of mathematical theorems that lay outside the scope of the current traditional logic-based theorem proving systems. One of its strengths comes from heuristic mathematical knowledge that restricts the search space and thereby facilitates the proving process for problems whose proofs belong to the restricted search space. But this may delays solutions and restricts the kinds of proofs that can be found for a given problem.

This paper describes an approach that combines knowledge-based proof planning (based on mathematically motivated heuristic control) with control knowledge learned from the analysis of randomized runs. It is based on investigations on so-called *heavy-tailed distributions* ([4, 3, 2]). Because of the non-standard nature of heavy-tailed cost distributions the controlled introduction of randomization into the search procedure and quick restarts of the randomized procedure can eliminate heavy-tailed behavior and can take advantage of short runs.

To apply these techniques to the complicated domains of proof planning, the first task was to find problem classes for which proof planning exhibits an unpredictable run time behavior, i.e., with heavy-tailed cost distributions. The second problem was to find appropriate randomization points, i.e., decisions in the proof planning process that are randomized. Thirdly, the experiments provided the basis for determining suitable *cutoff values*, i.e., the time interval after which a proof attempt is interrupted and a new attempt is started. Finally, we designed a new control strategy which dramatically boosts the performance of our proof planner for a class of problems for which proof planning exhibits heavy-tailed cost behavior.

2 Proof Planning

A proof planning problem is defined by an *initial state* specified by the proof assumptions, the *open goal* given by the theorem to be proved, and a set of *operators* [1]. A mathematical proof corresponds to a plan that leads from the initial state to the goal state.

Operators can encode general proof steps as well as proof steps particular to a mathematical domain. For a very basic example of an operator in proof planning consider the $=Subst$ operator. Its purpose is to replace occurrences of terms with respect to a given equation. $=Subst$ is applicable during the planning process if a current goal is a term $t[a]$ that contains an occurrence of a term a and there is an assumption that is an equation with a as one side and another term b as the other side. The application of $=Subst$ reduces then goal $t[a]$ to the new goal $t[b]$ which is the same term as $t[a]$ except for the occurrence of a.

As basic proof planner for the experiments described in this paper we used the proof planner of the ΩMEGA system [8, 7]. ΩMEGA's proof planner employs both backward and forward planning.

3 The Domain of Residue Classes

In this section, we describe the domain of residue classes over the integers. A detailed description of the domain can be found in [6].

A residue class set RS_n over the integers is the set of all congruence classes modulo an integer n, i.e., $\mathbb{Z}_n$ or an arbitrary subset of $\mathbb{Z}_n$. Concretely, we can deal with sets of the form $\mathbb{Z}_3, \mathbb{Z}_5, \mathbb{Z}_3 \backslash \{\bar{1}_3\}, \ldots$ where $\bar{1}_3$ denotes the congruence class 1 modulo 3. Binary operations $\circ$ on a residue class set are either $\bar{+}, \bar{-}, \bar{*}$ which are the addition, subtraction, and multiplication on residue classes or functions composed from these connectives, e.g., $(\bar{x} \bar{*} \bar{y}) \bar{+} (\bar{y} \bar{+} \bar{x})$. For a given residue class set RS_n and a binary operation $\circ$ we can examine their basic algebraic properties (Is the set RS_n closed with respect to the binary operation $\circ$? Is it associative? Does it have a unit element? etc.) and classify them in terms of groups, monoids, etc. Moreover, we are interested in classifying structures into equivalence classes of isomorphic structures. During this classification process, we have to prove proof obligations stating that two structures $(RS^1_{n_1}, \circ_1)$ and $(RS^2_{n_2}, \circ_2)$ are isomorphic or not. Two structures $(RS^1_{n_1}, \circ_1)$ and $(RS^2_{n_2}, \circ_2)$ are isomorphic, if there exists a total function $h : RS_{n_1} \rightarrow RS_{n_2}$ such that h is injective, surjective, and is a homomorphism with respect to $\circ_1$ and $\circ_2$. A function h is a homomorphism, if $h(x \circ_1 y) = h(x) \circ_2 h(y)$ holds for all $x, y \in RS_{n_1}$. A *non-isomorphism problem* is formalized as $\neg iso(RS^1_{n_1}, \circ_1, RS^2_{n_2}, \circ_2)$, where iso abbreviates $isomorphic$.

We developed several proof techniques to tackle these non-isomorphism problems in ΩMEGA. We will focus here on two of those techniques, (1) proof by case analysis and (2) proof by contradiction.

(1) The case analysis strategy is a simple but reliable approach to prove a property of a residue class structure. Its essence is a proof by cases. It exhaustively checks all cases, i.e., all instances of a conjecture. Since residue class sets are finite, only finitely many

instances have to be considered. For non-isomorphism problems the top-most case split considers every possible function h from the one residue class set into the other one and for each of these h the proof planner tries to establish that it is either not injective, not surjective, or not a homomorphism. In the remainder of the paper we call this proof technique the `TryAndError` strategy.

(2) An alternative proof strategy creates a proof by contradiction. It assumes that there exists a function $h{:}RS^1_{n_1} \rightarrow RS^2_{n_2}$ which is an isomorphism and hence an injective homomorphism. It derives a contradiction by proving that there are two elements $c_1, c_2 \in RS^1_{n_1}$ with $c_1 \neq c_2$ but $h(c_1) = h(c_2)$ which contradicts the assumption of injectivity of h.

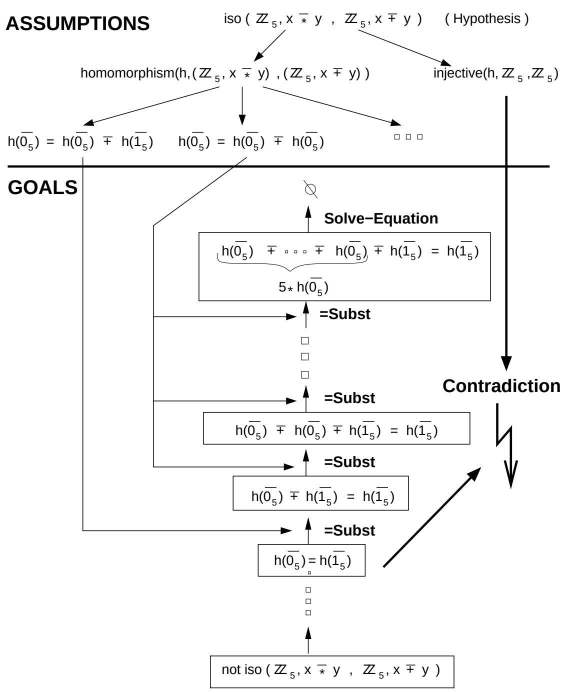

Fig. 1. Proof by ΩMEGA

Note, that the proof is with respect to all possible homomorphisms h rather than for a particular mapping. In the remainder of the paper we call the described proof technique to tackle non-isomorphism proofs the `NotInjNotIso` technique. We briefly explain the `NotInjNotIso` strategy for an example.

For proving $(\mathbb{Z}_5, \bar{x}\bar{*}\bar{y})$ is not isomorphic to $(\mathbb{Z}_5, \bar{x}\bar{+}\bar{y})$ (see Fig. 1) first the strategy constructs the indirect argument: From the hypothesis that the two structures are isomorphic it deduces the two assumptions that there exists a function h that is injective and a homomorphism. From *injective(h,*$\mathbb{Z}_5$*,*$\mathbb{Z}_5$*)* a contradiction can be concluded, if the negation is provable as well. The strategy continues by applying an operator to the second assumption *homomorphism(h,*$(\mathbb{Z}_5, \bar{x}\bar{*}\bar{y})$*,*$(\mathbb{Z}_5, \bar{x}\bar{+}\bar{y})$*)*. It introduces all instances of the homomorphism equation $h(x\bar{*}y) = h(x)\bar{+}h(y)$ as new assumptions instantiated for every element of the domain. In the example 25 equations like

$h(\bar{0}_5) = h(\bar{0}_5)\bar{+}h(\bar{1}_5)$ for $x = \bar{0}_5, y = \bar{1}_5$ (a)

$h(\bar{0}_5) = h(\bar{0}_5)\bar{+}h(\bar{0}_5)$ for $x = \bar{0}_5, y = \bar{0}_5$ (b)

are introduced. From this set of instantiated homomorphism equations the `NotInjNotIso` strategy tries to derive that h is not injective. To prove this, it has to find two witnesses c_1 and c_2 such that $c_1 \neq c_2$ and $h(c_1) = h(c_2)$ holds. In the example, $\bar{0}_5$ and $\bar{1}_5$ are chosen for c_1 and c_2 which leads to $h(\bar{0}_5) = h(\bar{1}_5)$. The operator $=Subst$ transforms this goal into the equation $h(\bar{0}_5)\bar{+}h(\bar{0}_5)\bar{+}h(\bar{0}_5)\bar{+}h(\bar{0}_5)\bar{+}h(\bar{0}_5)\bar{+}h(\bar{1}_5) = h(\bar{1}_5)$ by successively applying equations from the assumptions. The choice of the next instantiated homomorphism equation to be applied is guided by a heuristic described in [5]. First, equation (a) is applied to the left hand side of the equation which results in $h(\bar{0}_5)\bar{+}h(\bar{1}_5) = h(\bar{1}_5)$. Then equation (b) is applied four times to occurrences of $h(\bar{0}_5)$ on the left hand side. The final goal is closed by the operator $Solve{-}Equation$ which calls the Computer Algebra System MAPLE to evaluate the equation. The goal holds since $5\bar{*}h(\bar{0}_5)$ equals $\bar{0}_5$ modulo 5.

4 Experimental Results

The experiments were conducted with 160 non-isomorphism problems for the residue class set $\mathbb{Z}_5$. We decided for the residue class set $\mathbb{Z}_5$ because its cardinality is small enough to obtain solution statistics in a reasonable time. Problems from this class are:
1. $\neg iso(\mathbb{Z}_5, x\bar{*}y, \mathbb{Z}_5, x\bar{+}y)$,
2. $\neg iso(\mathbb{Z}_5, x\bar{-}y, \mathbb{Z}_5, (x\bar{-}y)\bar{+}(x\bar{-}y))$.
The overall experimental effort was around one month of cpu time on a 32 node computer cluster. A detailed description of all experiments can be found in [5].

4.1 Randomization and Heavy-Tailed Behavior

First let us report the results for the `NotInjNotIso` strategy because this strategy leads to the most interesting proof planning behavior in the residue class domain. The application of the `NotInjNotIso` strategy to all problems of the testbed solved 108 of the 160 instances (67.5%) (2 hour time limit per proof attempt). The runs revealed a surprisingly high variance in the performance of this strategy on the different problems of the testbed. On some of the problems it succeeded very fast and produced short proof plans consisting only of a few applications of $=Subst$, whereas on other problems the planning process took much longer and resulted in proof plans with many applications of $=Subst$. Furthermore, for over 30% of the instances no proof was found in 2 hours.

Table 1 displays the performance extrema for this deterministic proof by contradiction strategy on the testbed as well as the mean values over all successful runs. The values in brackets indicate the deviation from the mean. Fig. 2 shows the underlying distribution of the run time for these experiments. In fact, the distribution exhibits *heavy-tailed* behavior [2] which is manifested in the long tail of the distribution stretching for several orders of magnitude.

Costs	Mean	Min.	Max.
Proof length	55	45 (18.2%)	83 (50.9%)
Run Time	483	8(98%)	7145(1380%)

Table 1. Statistics for successful runs (108 out of 160) on testbed using deterministic strategy.

Gomes *et al.* have shown that one can take advantage of the large variations in run time of such heavy-tailed distributions by introducing an element of randomness into

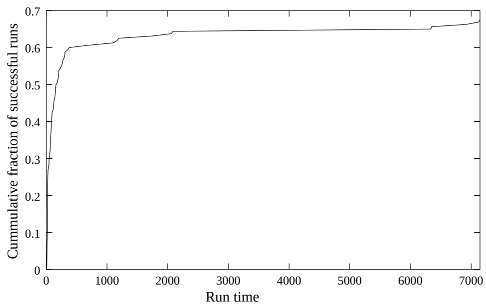

Fig. 2. Run time distribution over testbed without randomization.

the search process, combined with a restart strategy. A key criterion for the success of such a randomization and restart approach is a large variance in different randomized runs with the same instance.

To explore this issue, we considered multiple runs on a single instance by introducing a stochastic element into the planning process. Typically, the heuristic for choosing the next instantiated homomorphism equation to be applied ranks several equations equally good. For the randomized version the planner applies them in a random order when faced with such equally ranked equations. This randomized version of the `NotInjNotIso` technique was applied 225 times to one problem instance of the testbed:

$$\neg iso(\mathbb{Z}_5, (\bar{x} \bar{+} \bar{y}) \bar{+} \bar{2}_5, \mathbb{Z}_5, (\bar{2}_5 \bar{*} (\bar{x} \bar{+} \bar{y})) \bar{+} \bar{2}_5)$$

(in the remainder of this section we refer to this problem as the *standard problem*).

Interestingly, the run time distribution of the randomized proof search by contradiction on the single instance also exhibits heavy-tailed behavior similar to Fig. 2 (see [5] for a detailed analysis). This indicates an inherent variance in the search process of the strategy. Fig. 2 shows that the ascend of the cumulative cost distribution function is very steep at the beginning but becomes very flat beyond approximately 300 seconds. This steep ascend at the beginning indicates that there is a large fraction of short and successful runs, whereas the flat ascend after 300 seconds provides evidence that the probability of finding a proof plan decreases considerably. Hence, it is advantageous to perform a sequence of restarts on a single instance (with a predefined cutoff) until reaching a successful run or the total time limit, instead of performing a single long run.

Based on an analysis of the underlying distributions of the experiments for the full testbed and for the standard problem, we experimented with several cutoff values. The cutoff value of 80 seconds provided the best results. The planner found proof plans for 156 of the 160 problems (97.5%) in an average time of 291.4 seconds. Fig. 3 plots the run time distribution of the resulting restart strategy with cutoff 80 (log-log scale) on the problems of the testbed. The restart curve is the one that drops rapidly. The sharp drop of the run time distribution of the restart strategy clearly indicates that this strategy does not exhibit heavy tailed behavior. For comparison, the figure also shows the run time distribution of the deterministic strategy. This curve is approximately linear.

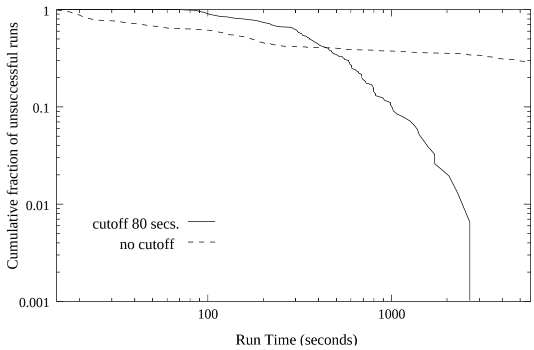

Fig. 3. Log-Log plots of run time distribution over testbed with and without randomization.

In previous applications of randomization and restarts in combinatorial domains run time has been the key issue [2]. In the case of proof planning, a new issue is the length of the solution discovered by the system because shorter proofs are generally more elegant than long proofs. In our experiments randomization and restarts leads to a variety of proof lengths for the same problem instance. For instance, for our standard problem instance, we found a range of proofs from proofs consisting of 47 to 78 nodes. That is, the planner can generate a variety of proofs for one problem if it employs a flexible control that includes randomization and restarts. This greatly enhances the ability of the system to find proofs and increases the overall robustness of the theorem proving system.

4.2 `TryAndError` Strategy

The `TryAndError` strategy explores all possible mappings between the structures. Since the goal is to prove a non-isomorphism, the proof planner needs to establish that no mapping is an isomorphism. Obviously, this strategy is computationally very expensive and, as our experiments show, it is practically infeasible for structures of cardinality larger than four.

There still is the question as to whether randomization may be of use in this context. Table 2 shows that there is some variation in run time and proof length (100 randomized runs on a single problem instance from $\mathbb{Z}_3$) due to different search pruning effects but the variations are small compared to those encountered for the proof by contradiction strategy. Further analysis (see [5]) shows that the underlying distribution is not heavy-tailed and therefore a restart approach would not boost the performance significantly.

Costs	Mean	Min.	Max.
Proof Length	598	540 (9.7%)	684 (14.4%)
Run Time	2456	1110 (54.8%)	4442 (80.9%)

Table 2. Randomized version of the `TryAndError` strategy.

5 Conclusions

The analysis of the cost distributions of proof planning attempts for a class of theorems and on the detection of *heavy-tailed* behavior gave rise to an application of randomization and restarts techniques. The experimental part of the investigations includes a study of two different planning strategies and the determination of cutoff values for the restart. The one planning strategy, `TryAndError`, implements a complete case analysis for proving a property. The strategy suffers from combinatorial explosion as soon as large sets are involved. Introduced into this strategy, randomization does not lead to substantially different proofs in length and run time. On the other hand, we showed that the planning strategy `NotInjNotIso` that constructs an proof by contradiction exhibits a remarkable high variance in performance in the presence of randomization. Our experiments evidence that the cost distribution underlying this strategy is heavy-tailed.

Further experiments provided a new kind of control knowledge in the form of appropriately randomized decision points and suitable cutoff values for restarts. The introduction of this knowledge into the proof planning process makes proof planning more robust, i.e., in our experiments a much larger fraction – from 67.5% to 97.5% – of problem instances became solvable. Moreover, a variety of proofs can be generated for one problem. In this paper, we described experiments with non-isomorphism problems of the residue class set $\mathbb{Z}_5$. We obtained analogous results on non-isomorphism problems of the residue class sets $\mathbb{Z}_2$, $\mathbb{Z}_3$, $\mathbb{Z}_4$, and $\mathbb{Z}_6$ (see [5]).

Proof planning can benefit from these investigations because they provide a stochastic approach to semi-automatically designing control knowledge. This kind of control knowledge can complement the mathematically motivated control knowledge previously used in proof planning. This is particularly important for problem classes that exhibit high branching for which little or no control knowledge is available.

References

1. A. Bundy. The Use of Explicit Plans to Guide Inductive Proofs. In *Proc. of CADE–9*, volume 310 of *LNCS*. Springer, Germany, 1988.
2. C. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasonong*, 24:67–100, 2000.
3. C. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proc. of AAAI-98*, pages 431–437, 1998.
4. C. Gomes, B. Selman, K. McAloon, and C. Tretkoff. Randomization in backtrack search: Exploiting heavy-tailed profiles for solving hard scheduling problems. In *Proc. of AIPS'98*, pages 208–213, 1998.
5. A. Meier. Randomization and heavy-tailed behavior in proof planning. SEKI-Report SR-00-03 (SFB), Universität des Saarlandes, Saarbrücken, Germany, 2000.
6. A. Meier, M. Pollet, and V. Sorge. Exploring the domain of residue classes. SEKI-Report SR-00-04 (SFB), Universität des Saarlandes, Saarbrücken, Germany, 2000.
7. E. Melis and A. Meier. Proof planning with multiple strategies. In *Proc. of the First International Conference on Computational Logic (CL2000)*, volume 1861 of *LNAI*. Springer, Germany, 2000.
8. E. Melis and J. Siekmann. Knowledge-based proof planning. *Artificial Intelligence*, 1999.