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

1 Introduction

Proof planning considers mathematical theorem proving as a planning problem. It has enabled the derivation of mathematical theorems that lay outside the scope of 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 in the restricted search space. But this may exclude solutions or restrict the kinds of proofs that can be found for a given problem.

We take a different perspective and investigate problem classes for which little or no heuristic control knowledge is available and test the usage of randomization and restart techniques. Our approach to control in those mathematical domains 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. Secondly, the experiments provided the basis for determining suitable *cutoff values*, i.e., the time interval after which a running 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.

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 given equations. $=Subst$ 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 $t[a]$ but the occurrence of a is replaced by an occurrence of b.

The proof planning approach developed in this paper is implemented in the ΩMEGA system [8, 7]. ΩMEGA employs backward chaining as its main planning strategy. That is, the planner continuously tries to reduce open goals by applying an operator that has an appropriate effect, which in turn might result in one or more new open goals and so on. Initially, the only open goal is the theorem. During this planning process there are several choice points such as which goal should be tackled or which operator should be applied in the next step.

3 The Domain of Residue Classes

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

The Residue Class Domain 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 $\mathbb{Z}_3, \mathbb{Z}_5, \mathbb{Z}_3 \setminus \{\bar{1}_3\}, \ldots$ 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 given residue class set and binary operation 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. Thereby, 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 $isomorphic$.

Two Proof Strategies 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 basic but reliable approach to prove a property of a residue class structure. Its essence is a proof by cases. It exhaustively checks 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 is to check for each possible function from the one residue class set into the other one that it is either not injective, not surjective, or not a homomorphism.

(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 thus, in particular, an injective homomorphism. It derives the 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. Note, that the proof is with respect to all possible homomorphisms h and we do not have to give 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 the example that $(\mathbb{Z}_5, \bar{x}\bar{*}\bar{y})$ is not isomorphic to $(\mathbb{Z}_5, \bar{x}\bar{+}\bar{y})$. The strategy first constructs the situation for the indirect argument. From the hypothesis that the two structures are isomorphic follow the two assumptions that there exists a function h that is injective and a homomorphism. By the first assumption a contradiction can be concluded when we are able to show that h is not injective.

The planner continues by applying a method to the second assumption, that introduces the homomorphism equation $h(x\bar{*}y) = h(x)\bar{+}h(y)$ instantiated for every element of the domain as new assumptions. In the above 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 $h(c_1) = h(c_2)$. In our example $\bar{0}_5$ and $\bar{1}_5$ are chosen for c_1 and c_2, respectively, which leads to $h(\bar{0}_5) = h(\bar{1}_5)$. This goal is transformed 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 equation system with the operator $=Subst$. 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 an application of the operator $Solve{-}Equation$ which calls the Computer Algebra System MAPLE to evaluate the equation. The final equation holds since $5\bar{*}h(\bar{0}_5)$ equals $\bar{0}_5$ modulo 5. The choice of the next instantiated homomorphism equation to be applied is guided by a heuristic described in [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 compute cluster. A detailed description of all experiments can be found in [5].

4.1 Randomization and Heavy-Tailed Behavior

First let us consider 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.

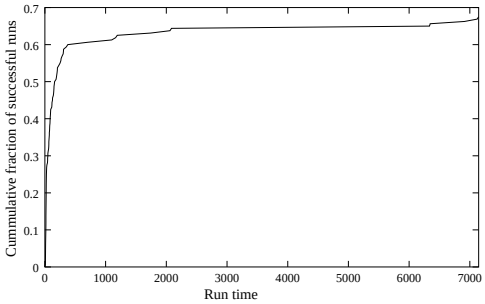

Fig. 1. Run time distribution over testbed without randomization.

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 give the deviation from the mean. Fig. 1 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. 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 the search process, combined with a restart strategy.

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.

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. When faced with such equally ranked equations, the planner applies them in a random order. This randomized version of the `NotInjNotIso` technique was run 225 times for the 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. 1 (see [5] for a detailed analysis). This indicates an inherent variance in the search process of the strategy.

Given this result, we can now use a restart strategy to improve the proof search performance. Fig. 1 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

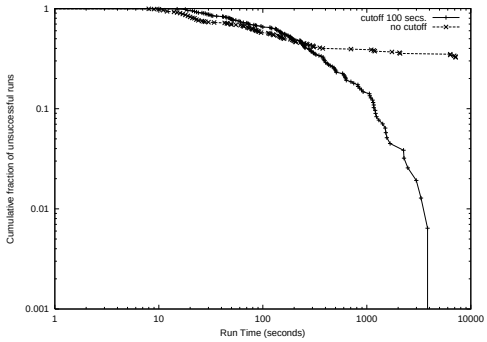

Fig. 2. Log-Log plots of run time distribution over testbed with and without randomization.

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 considered several cutoff values, using a binary search strategy. The cutoff value of 100 provided the best results. The planner found proof plans for 156 of the 160 problems (97.5%) in an average time of 473.4 seconds. For the four remaining unsolved problems MAPLE does not provide any substitution hint and, thus, the proof by contradiction strategy becomes quite ineffective.

Fig. 2 plots the run time distribution of the resulting restart strategy with cutoff 100 (log-log scale) on the problems of the testbed. The restart data is given by the curve that drops rapidly. The figure also shows the run time distribution of the deterministic strategy. The sharp drop of the run time distribution of the restart strategy clearly indicates that this strategy does not exhibit heavy tailed behavior.

In previous applications of randomization and restarts in combinatorial domains run time has been the key issue [2]. In the case of proof planning, an additional important issue is the length of the proof discovered by the system: shorter proofs are generally more elegant than long proofs. An interesting aspect of the application of randomization and restart strategies that is novel in our context is the fact that it 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. Such a degree of variance is unusual for proofs generated by proof planning.

Having a set of proof planning operators and a flexible control that includes randomization the planner can generate a variety of proofs. This greatly enhances the ability of the system to find proofs and increase the overall robustness of the theorem proving system.

4.2 Case Analysis Strategy

This strategy explores all possible mappings between the structures. Since the goal is to prove a non-isomorphism, the prover 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 still 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 strategy 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 case analysis 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 cut-off values for the restart. As a conclusion, we have introduced new kind of control knowledge into the proof planning process, a much larger fraction of problem instances became solvable (from 67.5% to 97.5%), and a variety of proofs can be generated for a problem. The application of randomization and restart techniques makes the search process more robust even when the size of the search spaces involved grows super-exponentially. We described in this paper 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 in general because they provide a stochastic approach to semi-automatically designing control knowledge and because this kind of control knowledge can augment the mathematically motivated control knowledge previously used in proof planning.

References

1. A. Bundy. The Use of Explicit Plans to Guide Inductive Proofs. In E. Lusk and R. Overbeek, editors, PROCEEDINGS *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 *Proceedings 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 *Proceedings 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)*, pages 644–659, 2000.
8. E. Melis and J. Siekmann. Knowledge-based proof planning. *Artificial Intelligence*, 1999.