# Exploiting Competitive Planner Performance

Adele E. Howe, Eric Dahlman, Christopher Hansen, Michael Scheetz, and Anneliese von Mayrhauser

Computer Science Department, Colorado State University
Fort Collins, CO 80523 U.S.A.
e-mail: {howe, dahlman, hansenc, scheetz, avm}@cs.colostate.edu

**Abstract.** To date, no one planner has demonstrated clearly superior performance. Although researchers have hypothesized that this should be the case, no one has performed a large study to test its limits. In this research, we tested performance of a set of planners to determine which is best on what types of problems. The study included six planners and over 200 problems. We found that performance, as measured by number of problems solved and computation time, varied with no one planner solving all the problems or being consistently fastest. Analysis of the data also showed that most planners either fail or succeed quickly and that performance depends at least in part on some easily observable problem/domain features. Based on these results, we implemented a meta-planner that interleaves execution of six planners on a problem until one of them solves it. The control strategy for ordering the planners and allocating time is derived from the performance study data. We found that our meta-planner is able to solve more problems than any single planner, but at the expense of computation time.

## 1 Motivation

The implicit goal of planning research has been to create the best general purpose planner. Many approaches have been implemented (e.g., partial order, SAT based, HTN) for achieving this goal. However, evidence from the AIPS98 competition [9] and from previous research (e.g., [14]) suggests that none of the competing planners is clearly superior on even benchmark planning problems.

In this research, we empirically compared the performance of a set of planners to start to determine empirically which works best when. No such study can be comprehensive. To mitigate bias, we tested unmodified publically available planners on a variety of benchmark and new problems. As a basis, we included only problems in a representation that could be handled by all of the planners.

The AIPS98 planning competition facilitated comparison of planners through a common representation language (PDDL [10]). Consequently, we had access to a set of problems and planners that could accept them (Blackbox, IPP, SGP and STAN). To expand the study, we added a domain that we had been developing and two other planners with compatible representations (UCPOP and Prodigy).

Our study confirmed what we expected: no one planner excelled. Of the 176 problems solved by at least one of the planners, the most solved by a single planner was 110. Computation times varied as well, with each planner posting the best time for some problem.

The data and PDDL provided a further opportunity. If no single planner is best, then an alternative is to build a meta-planner that incorporates several planners. The meta-planner tries the planners in sequence until a solution is found or a computational threshold is exceeded. Fortunately, we have found that if they can find any solution, most of the planners solve problems relatively quickly. The idea is a crude interpretation of the plan synthesis advocated in Kambhampati's IJCAI challenge [5], of the flexible commitment strategy advocated by Veloso and Blythe [14] and the interleaving of refinement strategies of UCP [6]; it was inspired by the Blackbox planner's inclusion of multiple SAT solving strategies[7].

We implemented a prototype meta-planner, called *BUS* for its role of communication and control. BUS allocates time to each of the planners until one solves it. As the basis of BUS's control strategy, we developed models of planner performance from the comparative study data. We tested BUS on the original problems and new problems. We expected that we would sacrifice some computation time to increase the overall problem solving rate. Although BUS provides a simple mechanism for averaging planner performance, its primary contribution is as a testbed for comparing performance of different planners and testing models of what works best when.

## 2 Planners

Because the AIPS98 competition required planners to accept PDDL, the majority (four) of planners used in this study were competition entrants, or are later versions thereof. The common language facilitated comparison between the planners without having to address the effects of a translation step. The two exceptions were UCPOP and Prodigy; however, their representations are similar to PDDL and were translated automatically. The planners represent four different approaches to planning: plan graph analysis, planning as satisfiability, partial order planning and state space planning with learning.

*STAN* [11] extends the Graphplan algorithm[2] by adding a preprocessor to infer type information about the problem and domain. This information is then used within the planning algorithm to reduce the size of the search space that the Graphplan algorithm would search. STAN can only handle problems using the STRIPS subset of PDDL.

*IPP* [8] extends Graphplan by accepting a richer plan description language, a subset of ADL. The representational capabilities are achieved via a preprocessor that expands more expressive constructs like quantification and negated preconditions into STRIPS constructs. The expanded domain is processed to remove parts of it that are irrelevant to the problem at hand. We used the AIPS98 version of IPP because the newer version no longer accepts PDDL.

*SGP* [16] also extends Graphplan to a richer domain description language. As with IPP, some of this transformation is performed using expansion techniques to remove quantification. SGP also *directly* supports negated preconditions and conditional effects. SGP tends to be slower (it is implemented in Common Lisp instead of C) but more robust than the other Graphplan based planners.

*Blackbox* [7] converts planning problems into boolean satisfiability problems, which are then solved using a variety of different techniques. In constructing the satisfiability problem, Blackbox uses the planning graph constructed as in Graphplan. We used version 2.5 of Blackbox because the newer versions were not able to parse some of the problems and in the parseable problems, we did not find a significant difference in performance.

*UCPOP* [1] is a Partial Order Causal Link planner. The decision to include UCPOP was based on several factors. First, it does not expand quantifiers and negated preconditions; for some domains, Graphplan-like expansion can be so great as to make the problem insolvable. Second, we had used UCPOP in developing an application which provides the third category of problems.

*Prodigy* [15] combines state-space planning with backward chaining from the goal state. A plan under construction consists of a head-plan of totally ordered actions starting from the initial state and a tail-plan of partially ordered actions related to the goal state. Informal results presented at the AIPS98 competition suggested that Prodigy performed well in comparison to the entrants.

## 3 Test Problems

Three criteria directed the compilation of the study test set; the problems should be: comprehensive, challenging and available in accepted representations. We included a wide set of problems, most of which were available to or had been used by the planner developers. We favored domains that were challenging, meaning that not all of the planners could solve the problems within a few minutes.

The common representation is the Planning Domain Definition Language (PDDL)[10]. PDDL is designed to support a superset of the features available in a variety of planners. At the minimum, the planners that are the subjects of our study all accept STRIPS representation. Although restricting problems to STRIPS reduced the test set significantly, we were concerned that our modifying the test problems could bias the results.

We included problems from three test suites: the AIPS98 competition set, the UCPOP benchmarks and a Software Testing domain. The first two sets are publically available. The third was developed over the past three years as an application of planning to software engineering; it has proven difficult for planners and had features not present in the other domains.

*Competition Domains* The largest compendium of planning problems is probably the AIPS 98 competition collection [9]. Creators of competition planners knew their planners would be run on these problems and so had the opportunity to design to them. For UCPOP and Prodigy[1], the PDDL STRIPS representation was syntactically

[1] We thank Eugene Fink for translation code from PDDL to Prodigy.

modified to match their requirements. The suite contained 155 problems in 6 domains.

*UCPOP Benchmarks* The UCPOP distribution [13], includes 85 problems in 35 domains. The problems exploit a variety of representational capabilities. As a consequence, many of these problems could not be accepted by the chosen planners. From the distribution, we identified 18 problems from 7 domains that could be accepted by all of the planners with only minor syntactic modification.

*Our Addition: Software Testing* Generating test cases for software user interfaces can be viewed as a planning problem. The commands to the interface can be represented as operators in the domain theory. The problems then describe changes that a user might wish to have happen to the underlying system as a consequence of executing a sequence of commands. The planner automates the process of generating test cases by constructing plans to solve these problems.

We developed a prototype system, based on the UCPOP planner, for a specific application: Storage Technology's Robot Tape Library interface [4]. The application involves moving tapes around, into and out of a large silo, reading them in a tape drive and monitoring the status of the system. We selected UCPOP initially (in 1995) because it was publically available and easy for software engineers to use. However, we were having trouble generating non-trivial test cases.

The basic domain theory contains 11 actions and 25 predicates[2]. The tape library can be configured as connected silos with a tape drive in each and positions for tapes designated by panels, rows and columns. The configuration is described in the initial conditions of problems along with identifiers and initial positions for the tapes.

We created three variants on the basic domain theory to recode some problematic aspects of the original (i.e., conditional effects and disjunctive preconditions) and six core problems whose goals required use of different actions on two tapes. We then extended the set of problems by varying the size of the library configuration; the configurations always included an equal number of panels, rows and columns of values 4, 8, 12, 16 and 20. These positions tested the

[2] We will be making the basic domain theory available on a web site.

vulnerability of planners to extraneous objects. These combinations produced 90 different problems.

## 4 Empirical Comparison of Planners

We hypothesized that the best planner for a given domain would vary across the domains. To support our design of the meta-planner, we hypothesized that planners would exhibit significantly different times between success and failure (would either fail or succeed quickly) and that easily measurable features of problems and domains would be somewhat predictive of a planner's performance. The purpose of the empirical comparison is to test these three hypotheses.

We ran the six planners on 263 problems from the three sets. For each problem/domain combination, we recorded five easily observed features. For the domain, we counted the number of actions (Act) and the number of predicates (Pred). For the problem, we counted the number of objects (Obj), the number of predicates in the initial conditions (Init) and the number of goals (Goal).

We measured performance by whether the planner successfully found a solution and how much computation time was required to fail or succeed. We counted time-outs, core dumps and planner flagged failures as unsuccessful runs. We allocated up to 15 hours for most of the planners. UCPOP was given a search limit of 100000. All runs were conducted on the same platform: Sun Ultra 1 workstations with 128M of memory. For analysis, we filtered the results to include only those problems that were solved by some planner: 176 problems remained of the 263 in the tested set.

| | # Fastest | | # Solved | | Computation Time Comparison | | | |
|---|---|---|---|---|---|---|---|---|
| Planner | Suite | Software | Suite | Software | $\mu$ Success | $\mu$ Fail | T | $P <$ |
| Blackbox | 15 | 2 | 89 | 11 | 23.06 | 210.81 | -4.80 | 0.0001 |
| IPP | 4 | 1 | 71 | 2 | 156.28 | 821.08 | -1.36 | 0.178 |
| SGP | 13 | 0 | 91 | 0 | 1724.58 | – | – | – |
| STAN | 59 | 0 | 79 | 0 | 67.89 | 12.29 | 0.33 | 0.741 |
| UCPOP | 14 | 69 | 41 | 69 | 20.68 | 387.45 | -15.81 | 0.001 |
| Prodigy | 3 | 0 | 48 | 12 | 52.27 | 2828.43 | -15.12 | 0.0001 |

**Table 1.** Summary of Planners' Performance: by superior performance on problems and by comparing computation times for success and failure

The first hypothesis is that the best planner varies across problems. For each planner, Table 1 lists the number of problems on which that planner posted the fastest time and the total number of problems solved. These results are subdivided according to the problem collections: "suite" includes the competition and UCPOP problems, and "software" is our software testing domain. No planner solved all of the problems or even more quickly solved all of its problems. STAN was fastest in general. IPP generally lagged the others, but did solve a few problems that the others did not. The competition planners solved more of the problems from the benchmark suite. We had hoped that some of the other planners would excel on the software testing domain, but UCPOP dominated for that domain.

The second hypothesis was that the computation time would depend on success: the time to succeed would differ significantly from the time to fail. To test this, we partitioned the data based on planner and success and performed two tailed, two sample T-tests for each planner (see Table 1). All of SGP's failures were time-outs with identical times. Blackbox, UCPOP and Prodigy show significant differences: succeeding quickly and failing slowly. Success and failure times for STAN and IPP were not significantly different.

Finally, we hypothesize that the performance of planners depend on observable problem and domain features. To test this, we partitioned the data according to each of the five features and six planners. We then tested the relationship between the features and time by running a set of one way ANOVAs with computation time as the dependent variable and the feature as the independent variable. We tested the dependence between feature and success using Chi-Squared tests with counts for successful and total runs. Because cells were sparse, we coded each feature into five bins because the lowest number of values for a feature was 10. Some Prodigy features were missing too many cells to be analyzed with five bins.

Table 2 shows that performance of each planner depends on some of the features (statistically significant at $P < .05$). The number of predicates is significant in almost every case. Each planner showed significant results on from one to all five features on both performance metrics. This suggests that a subset of the features can be used to predict success and required computation time; which of them will vary for each planner.

| | | Time | | Success | | | | Time | | Success | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Planner | Feature | F | P | Chi | P | Planner | Feature | F | P | Chi | P |
| Blackbox | Init | 2.94 | .023 | 7.62 | .054 | IPP | Init | 8.11 | .001 | 15.98 | .003 |
| | Obj | 0.64 | .632 | 7.92 | .048 | | Obj | 3.54 | .017 | 8.48 | .075 |
| | Goal | 52.34 | .001 | 32.17 | .001 | | Goal | 0.74 | .567 | 27.25 | .001 |
| | Act | 3.55 | .010 | 38.60 | .001 | | Act | 1.20 | .317 | 9.26 | .055 |
| | Pred | 17.75 | .001 | 36.76 | .001 | | Pred | 0.17 | .917 | 31.85 | .001 |
| SGP | Init | 4.54 | .005 | 2.45 | .656 | STAN | Init | 1.34 | .268 | 13.59 | .009 |
| | Obj | 1.06 | .380 | 0.71 | .950 | | Obj | 0.33 | .802 | 4.26 | .372 |
| | Goal | 3.26 | .016 | 3.03 | .552 | | Goal | 24.46 | .001 | 4.27 | .370 |
| | Act | 2.25 | .070 | 1.99 | .738 | | Act | 1.06 | .372 | 19.57 | .001 |
| | Pred | 2.51 | .048 | 14.49 | .006 | | Pred | 7.48 | .001 | 8.02 | .046 |
| UCPOP | Init | 5.79 | .001 | 16.39 | .002 | Prodigy | Init | - | - | 9.04 | .06 |
| | Obj | 53.63 | .001 | 44.12 | .001 | | Obj | - | - | 22.79 | .0001 |
| | Goal | 49.66 | .001 | 41.24 | .001 | | Goal | 6.747 | .0001 | 37.86 | .0001 |
| | Act | 4.01 | .004 | 30.65 | .001 | | Act | 37.014 | .0001 | 55.67 | .0001 |
| | Pred | 20.44 | .001 | 61.43 | .001 | | Pred | 61.51 | .0001 | 66.48 | .0001 |

**Table 2.** ANOVA and Chi-square test results for dependence of performance on planner and domain/problem features

## 5 Meta-Planner

BUS is a meta-planner that schedules problem solving by six other planners. The key idea is that if one planner can solve a subset of planning problems then multiple planners should be able to solve more. The current version is a prototype for exploring combinations of planners and control strategies. More importantly, BUS is a platform for empirically comparing planner performance.

To solve a problem, BUS first determines the order in which the planners should be tried. It calculates an expected run time for each planner to solve the problem and an expected probability of success. The control strategy is to order algorithms by $\frac{P(A_i)}{T(A_i)}$ where $P(A_i)$ is the expected probability of success of algorithm $A_i$ and $T(A_i)$ is the expected run time of algorithm $A_i$. In the general case, this strategy minimizes the expected cost of trying a sequence of $n$ algorithms until one works [12].

In our prototype, the models of expected time and success are linear regression models of the performance data. As described in the last section, we analyzed the data to determine the problem/domain features upon which each planner's performance most depends. The best features were incorporated in a regression model, which provides an intercept and slopes for each of our features. Fortunately, all the features were interval metrics. We created one model for each of the planners on each of the two performance metrics.

The time models are composed of four features. Models of all the features tended to produce either negative times or high times for some problems. Four appeared to provide the best balance between enough information to predict and not too much to lead it astray.

For the success models, we computed five separate regression models for each planner, one for each feature. The dependent variable was *success rate per feature*; because it is relative to a specific feature, we could not compute a multiple linear regression model. Instead, we combined the statistically significant linear regression models using a uniform weighting. Another complication is that probabilities vary only from 0 to 1.0. To compensate, we added a ceiling and floor at these values for the probability models.

These models are linear, which, based on visualizing the data, is not the most accurate model. The $R^2$ values for the time models were: 0.46 for Blackbox, 0.19 for IPP, 0.26 for SGP, 0.35 for Stan, 0.76 for UCPOP and 0.51 for Prodigy. The $R^2$ values for the individual success models varied from 0.04 to 0.51. Still for the prototype, we relied on the linear regression models as they were easily obtained and implemented and could be justified by the data.

The core of BUS is a process manager. Each planner is run as a separate process. Planners are run in a round robin like scheme ordered by the control strategy. A planner is pulled off the front of the queue and allocated a time slice. The duration of the time slice is the expected run time needed for the particular planner to solve the proposed problem. If the planner solves the problem, the planning halts. If the planner fails, then it is removed from the round robin. If the time slice finishes without solution or failure, the process manager checks whether the current planner has taken as much time as the next planner in the queue requires. If not, the current planner as well as the proceeding planners are each allocated additional time slices until either one of them solves the problem or exceeds the time expected for the next planner. When control passes to the next planner, the current one is suspended, and its computation time so far is recorded. Computation time is accumulated until the overall amount exceeds a threshold (30 minutes for our trials). BUS translates the PDDL representations for UCPOP and Prodigy.

## 6 Performance of Meta-Planner

To test the efficacy of BUS, we ran it on a subset of problems solved in the comparison study plus some new problems. The comparison problems were all of the benchmark suite problems and the software testing problems on just one domain; these were included to set a baseline for performance. The new problems were included to determine how well the control strategy generalizes and to show that BUS can solve problems with representations that were not in the intersection of all of the planners. The new problems came from the AIPS98 competition test generators and the UCPOP distribution. We generated 10 each new logistics, mystery, and mprime problems. The UCPOP problems were from the travel, ferry, tire, and get-paid domains. We recorded the total computation time used by all the planners that BUS ran.

In expectation, BUS should be able to solve all of the problems albeit somewhat slower than the fastest single run times. The additional time accounts for overhead and for trying several planners before one solves it. BUS solved 133 of 151 problems from the benchmark suite and 11 of 25 problems from the software testing set. On the software problems, the strategy appeared to allocate too much time to planners other than UCPOP.

On the study test suite, we compared BUS's times to the best, average and worst times for each problem for the single planners. BUS performed better than average on 54 problems. BUS took longer than the best posted times on average: the mean for BUS was 72.83, the mean for the best solutions was 11.36. However, BUS required less time than the average and worst times across the planners ($T = -1.86, P < .066$ for comparison to average times).

For the new problems, 19 of the 30 generated problems were unsolvable, meaning that no individual planner could solve them in under 4 hours. The individual planners were allotted up to 4 hours, but SGP is the only one which required more than 30 minutes to solve two of its problems. As Table 3 shows, BUS did well in comparison

| Planner | # Solved | $\mu$ time solved | Planner | # Solved | $\mu$ time solved |
|---|---|---|---|---|---|
| Blackbox | 26 | 37.87 | IPP | 22 | 10.62 |
| SGP | 24 | 364.85 | Stan | 8 | 2.26 |
| UCPOP | 21 | 3.98 | Prodigy | 12 | 5.94 |
| BUS | 32 | 70.83 | | | |

**Table 3.** Number of problems solved and solution times for individual planners and BUS on new problems

to the other planners. It solved more problems than any individual planner albeit at the cost of extra computation time.

## 7 Observations

Although the current control strategy is simplistic, BUS demonstrated the efficacy of combining planner approaches in a meta-planner. As expected, it solved more problems from the comparison set than any single planner had done. Somewhat surprisingly, the computational cost for BUS was significantly lower than the average computation required by individual planners to solve the same problems. On new problems, BUS solved more problems than any individual planner.

While the evaluation of BUS so far is favorable, BUS needs both more work and more evaluation. The current control strategy was a first attempt, but does not adequately model the relationship between problems and planner performance. We have identified four key deficiencies: coverage of feature values, limited predictability of features, mismatch in the underlying modeling technique and effect of representation. First, even the current control strategy would be much improved by simply having more data; the current set of problems was uneven in the distribution of the feature values. Second, while statistical tests showed a dependency between the features and performance, the relationship is not strong enough to be adequately predictive. The features are shallow and do not relate directly to planner functionality. Finally, clearly, the linear regression models are not the most appropriate for the data; the features did not appear to be linear in predictability. Alternative control strategies could be based on machine learning, smoothing and non-linear regression techniques. The most promising available option is Eugene Fink's method for selecting problem solving methods [3], which might be adaptable to this task. His method has performed well on a domain similar to that used in some of our problems.

Researchers have long acknowledged the effect of representation on performance. In this study, planners were forced to use the STRIPS versions of problems. Additionally, the current version includes a translator for converting PDDL to UCPOP and Prodigy representations. Although the translation process is syntactic, we observed that the planners seemed to do better on the problems that were originally coded in their representation. The next version of BUS will keep a database of problems in their original representations, translating where possible and filtering when not. Also, we will be investigating whether the translator can perform some semantic manipulations as well to address this issue.

To further its utility, BUS needs a wider variety of planners. Initially, the different representations will be handled by the database. Later, we will work on automatic translations of the representations. The next version will also include a checker for filtering out problem types that do not work for some planners.

BUS is a prototype. We tested the assumptions underlying its design in a study on 176 planning problems. From that study, we determined that most planners either succeed or recognize failure quickly and that problem features can be used to predict likelihood of success and expected computation cost, two characteristics necessary for supporting the current design. We also derived a control strategy based on the data from the study. Although it tends to incur additional computational overhead, its performance was shown to be competitive with current state of the art planners. BUS serves both as a vehicle for exploring planner performance (we can determine which planner ultimately solved each problem) and for exploiting different planning approaches.

## 8 Acknowledgments

This research was supported by grants from the National Science Foundation: grant number CCR-9619787 and Career award number IRI-9624058. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. We also thank the reviewers for their suggestions of clarifications and extensions.

## References


1. A. Barrett, D. Christianson, M. Friedman, K. Golden, S. Penberthy, Y. Sun, and D. Weld. UCPOP user's manual. Technical Report TR 93-09-06d, Dept of Computer Science and Engineering, University of Washington, Seattle, WA, November 1996. Version 4.0.
2. A. Blum and M. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.
3. E. Fink. How to solve it automatically: Selection among problem-solving methods. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, June 1998.
4. A.E. Howe, A. von Mayrhauser, and R.T. Mraz. Test case generation as an AI planning problem. *Automated Software Engineering*, 4(1), 1997.
5. S. Kambhampati. Challenges in bridging plan synthesis paradigms. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.
6. S. Kambhampati and B. Srivastava. Universal Classical Planning: An algorithm for unifying state-space and plan-space planning. In *Current Trends in AI Planning: EWSP '95*. IOS Press, 1995.
7. H. Kautz and B. Selman. Blackbox: A new approach to the application of theorem proving to problem solving. In *Working notes of the AIPS98 Workshop on Planning as Combinatorial Search*, Pittsburgh, PA, 1998.
8. J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In *Fourth European Conference in Planning*, 1997.
9. D. McDermott. Aips98 planning competition results. http://ftp.cs.yale.edu/pub/mcdermott /aipscomp-results.html, June 1998.
10. D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. *The Planning Domain Definition Language*, May 1998.
11. M.Fox and D.Long. The automatic inference of state invariants in TIM. *JAIR*, 9:367–421, 1998.
12. H. A. Simon and J. B. Kadane. Optimal problems-solving search: All-or-none solutions. *Artificial Intelligence*, 6:235–247, 1975.
13. UCPOP Group. The UCPOP planner. http://www.cs.washington.edu/research/projects/ai/www/ucpop.html, 1997.
14. M. Veloso and J. Blythe. Linkability: Examining causal link commitments in partial-order planning. In *Proceedings of the Second International Conference on AI Planning Systems*, June 1994.
15. M. M. Veloso, J. Carbonell, M. A. Pérez, D. Borrajo, E. Fink, and J. Blythe. Integrating planning and learning: The prodigy architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1):81–120, 1995.
16. D. Weld, C. Anderson, and D. Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proc. of 16th National Conference on AI*, 1998.