The AIPS-98 Planning Competition: Competitors' Perspectives

Blackbox team: Henry Kautz[1] and Bart Selman[2]
HSP team:[3] Blai Bonet and Héctor Geffner
IPP team:[4] Jana Koehler, Michael Brenner,
Jörg Hoffmann and Frank Rittinger
SGP team:[5] Corin R. Anderson. Daniel S. Weld and David E. Smith
STAN team:[6] Maria Fox and Derek Long

Contributions editor: Derek Long

April 10, 1999

1 Introduction

The international conference, Artificial Intelligence Planning Systems, held at Carnegie Mellon University, Pittsburgh in June 1998 (AIPS-98) played host to the first world planning competition, in which competitors were invited to come and compete on a collection of domains and associated problems, sight-unseen, using whatever planning technology they wished. Tracks were offered for STRIPS, ADL and HTN planning, but in the event only STRIPS and ADL were entered. Indeed, ADL saw only two competitors: IPP [17] and SGP [1], with IPP demonstrating a convincingly superior

[1] AT&T Shannon Laboratory. kautz@research.att.com

[2] Department of Computer Science, Cornell University. selman@cs.cornell.edu

[4] Institute for Computer Science, Albert Ludwigs University, Am Flughafen 17, 79110 Freiburg, Germany, koehler@informatik.uni-freiburg.de

[3] Depto. de Computación, Universidad Simón Bolívar, Aptdo. 89000, Caracas 1080-A, Venezuela. {bonet,hector}@usb.ve

[5] {corin, weld}@cs.washington.edu and de2smith@ptolemy.arc.nasa.gov

[6] Department of Computer Science, University of Durham, UK.
{Maria.Fox,D.P.Long}@dur.ac.uk

performance over SGP in a single round playoff. The STRIPS track originally attracted some 9 or 10 declarations of intent to take part. This paper gives an account of the competition as seen by the competitors who actually arrived in Pittsburgh and took part in the two tracks.

Competitions as a way to evaluate and to promote progress in various fields have precedents, such as the MUC (Message Understanding Competition) series, the TREC (Text Retrieval Competition) competitions, the Turing Competitions and others. These have stimulated work, but they also represent a serious investment of effort for the competition organizers and competitors. It is a formidable task to create a collection of tasks which is both realistically within the reach of the existing technology and yet which represents an adequate challenge and which points the way for the field to develop. Administrative problems represent a huge overhead to this task: a common language must be developed that allows problems to be specified and results evaluated, scoring mechanisms must be determined and the environment must be selected and competitors forewarned. Tribute should be paid to Drew McDermott for the role he played in almost single-handedly executing all these tasks, with support from the competition committee.

For the competitors, the competition represents a challenge to the robustness of their software, and demands work in meeting the specifications for both input and output formats, while continuing to develop and enhance the basic functionality of their systems. The development of PDDL [23] as the common language for the competition problem specifications was an important step in the progress of the competition. A challenge to the competitors was to adapt to the many minor changes in this language as it steadily stabilized. Another important problem was anticipating the demands of the competition domains. All of the planners that eventually competed are domain-independent planners that require little or no manual guidance in selecting run-time behavior, but the performance of all of them can be dramatically affected by the design of the encoding of the domain and problem specifications. The criteria by which success was to be judged were also volatile: tradeoffs between the planning time and the optimality of the plan produced were a controversial balancing act. Optimality was determined to be measured purely by the number of steps in the plan for the STRIPS track - so planners producing optimal parallel plans might well find themselves significantly outperformed by planners concentrating on sequential plan optimality. Furthermore, the selection of domains to be used in the competition is hard. All of the competitors have some collection of favored domains which showcase the characteristics of their planners. In principle

all competitors had the opportunity to propose domains for use, but the constraints on the time available for the competition made it impossible to use all of these.

These various challenges thinned the field so that the competition eventually hosted four STRIPS planners and two ADL planners (IPP taking part in both tracks). It was apparent that there were several others who would have liked to have participated but were unable to meet the input and output criteria within the timescales that were imposed by the organization of the competition.

2 The Competing Planners

The competition highlighted several facets of the current state of the planning research field. Firstly, by comparison with even a couple of years ago, the technology has advanced dramatically in terms of the size of problems in standard benchmark domains which planners can realistically be expected to handle. All of the planners in the competition were solving some instances of problems in times measured in milliseconds, including non-trivial instances. Secondly, the collection of planners which are sufficiently robust for release into the general research community, require no special purpose additional domain encoding beyond STRIPS or ADL and which can tackle significant instances in reasonable time is still small. The qualifications are important, however, since there are clearly many more planners actively being worked on which are either still insufficiently robust to be entered into a competition event, or else require too much careful encoding of their domains to be able to compete automatically across large numbers of problems, sight unseen. It is to be hoped that this situation changes as the competition is repeated in future years, with a widening of the field and broadening of the range of technology being exposed. This latter point is particularly poignant, since, at AIPS-98, three of the five planners involved were directly built on GRAPHPLAN [2] (IPP, SGP and STAN), one exploited GRAPHPLAN technology (**blackbox**) and only one was independent of GRAPHPLAN in every way (HSP).

Interestingly, all five planners used full instantiation of actions as a prelude to search and this explains, in part, the difficulties all of the planners experienced with problems which involved huge numbers of ground operator instances. This point is further discussed in Section 3. It is unfortunate that no partial-order planning strategy was entered to offer a comparison of

performance. Some informal experiments with Prodigy [5] during the competition suggested that it might have performed significantly better than the other planners in some domains, but was completely outperformed in others. This pattern hints at one of the most interesting issues the competition raised, which is discussed in more detail in Section 3: different planning technologies, despite being domain independent, are actually highly sensitive to both domain and problem instance in determining actual performance.

The remainder of this section is devoted to a more detailed examination of the individual planners that entered the competition.

2.1 Blackbox

It has often been observed that the classical AI planning problem (that is, planning with complete and certain information) is a form of logical deduction. Because early attempts to use general theorem provers to solve planning problems proved impractical, research became focused on specialized planning algorithms. However, the belief that planning required such *specialized* reasoning algorithms was challenged by the work of Kautz and Selman on planning as propositional satisfiability testing [15, 16]. SATPLAN showed that a general propositional theorem prover could be competitive with some of the best specialized planning systems. The success of SATPLAN can be attributed to two factors:

- The use of a logical representation that has good computational properties. Both the fact that SATPLAN uses propositional logic instead of first-order logic, and the particular conventions we suggested for representing time and actions, are significant. Differently declarative representations that are semantically equivalent can have quite distinct computational profiles.

- The use of powerful new general reasoning algorithms such as Walksat [29]. Many researchers in different areas of computer science devise new algorithms and implementations for SAT testing each year, and freely share ideas and source code. As a result, at any point in time the best general SAT engines tend to be faster (in terms of raw inferences per second) than the best specialized planning engines.

An approach that shares a number of features with the SATPLAN strategy is the GRAPHPLAN system, developed independently by Blum and Furst [2]. Comparisons with SATPLAN show that neither algorithm is strictly superior.

For example, SATPLAN is faster on a complex Logistics domain, they are comparable on the Blocks World, and on several other domains GRAPHPLAN is faster.

GRAPHPLAN bears an important similarity to SATPLAN: both systems work in two phases, first creating a propositional structure (in GRAPHPLAN, a plan graph, in SATPLAN, a CNF wff) and then searching that structure. The propositional structure corresponds to a fixed plan length, and the search reveals whether a plan of that length exists. Furthermore, in Kautz and Selman [16] it is shown that the plan graph has a direct translation to CNF, and that the form of the resulting formula is very close to the original conventions for SATPLAN. It is hypothesized that the differences in performance of the two systems can be explained by the fact that GRAPHPLAN uses a better algorithm for *instantiating* the propositional structure, while SATPLAN uses more powerful *search* algorithms.

SATPLAN fully instantiates a problem instance before passing it to a simplifier and a solver. By contrast, GRAPHPLAN *interleaves* plan graph instantiation and simplification. Furthermore, GRAPHPLAN employs a powerful planning-specific simplification algorithm. The algorithms corresponds to a limited application of resolution where one of the clauses is restricted to be binary and negative. The use of this rule in GRAPHPLAN is called *mutex computation*, because it is used to determine that pairs of actions or pairs of facts are mutually exclusive. The set of mutexes is used both to prune nodes from the graph during instantiation and to prune branches of the search tree.

These observations have led to the creation of a new system that combines the best features of GRAPHPLAN and SATPLAN. This system, called **blackbox**, works in three phases:

1. A planning problem (specified in a standard STRIPS notation) is converted to a plan graph;

2. The plan graph is converted to a CNF wff;

3. The wff is solved by any of a variety of fast SAT engines.

Figure 1 illustrates the translation of a plan graph to axiom schemas. The plan graph is layered from left to right, moving from the past to the future. Solid arrows indicate precondition or effect links, and the dashed line indicates a mutex relationship between actions. The first schema states that a fact implies the disjunction of the actions that add or maintain it

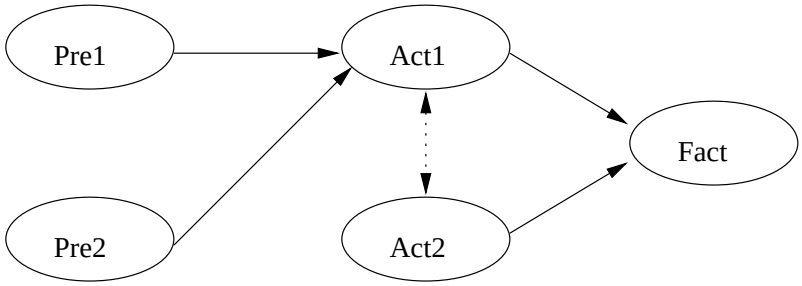

Figure 1: This fragment of a plan graph translates to the axiom schemas

$$\text{Fact} \supset \text{Act1} \vee \text{Act2}$$
$$\text{Act1} \supset \text{Pre1} \wedge \text{Pre2}$$
$$\neg\text{Act1} \vee \neg\text{Act2}$$

(a kind of frame axiom). The second states that an action implies its preconditions. The third states that mutex actions cannot occur at the same time slot. (Non-mutex actions may occur in parallel.) Together with a set of unit clauses that assert that the goals hold in the final (rightmost) layer, the translation guarantees that any model of the axioms corresponds to a solution to the plan graph, and vice-versa.

The wff generated from the plan graph can be considerably smaller than one generated by translating STRIPS operators to axioms in the most direct way, as was done by the earlier MEDIC system of Ernst, Millstein, and Weld [4]. Furthermore, the plan graph's computed mutex relationships can be directly translated into negative binary clauses, which can make the formula easier to solve by many kinds of SAT engines.

The competition version of **blackbox** included the local-search SAT solver Walksat and the systematic SAT solver satz [20], as well as the original GRAPHPLAN engine (that searches the plan graph instead of the CNF form). In order to have robust coverage over a variety of domains, the system can employ a *schedule* of different solvers. For example, it can run GRAPHPLAN for 30 seconds, then Walksat for 2 minutes, and if still no solution is found, satz for 5 minutes.

The **blackbox** system introduces new SAT technology as well, namely the use of *randomized complete search methods*. As shown in Gomes, Selman, and Kautz [10], systematic solvers in combinatorial domains often exhibit a "heavy tail" behavior, whereby they get "stuck" on particular instances.

Adding a small amount of randomization to the search heuristic and rapidly restarting the algorithm after a fixed number of backtracks can dramatically decrease the average solution time, often from hours to seconds.

This randomization/restart technique was applied to the version of satz used by **blackbox**. The variable-choice heuristic for satz chooses to split on a variable that maximizes a particular function of the number of unit propagations that would be performed if that variable were chosen (see [20] for details). The **blackbox** version, satz-rand, randomly selects among the set of variables whose scores are within 40% of the best score.The solver schedule used in the competition was to run the GRAPHPLAN engine for 20 seconds, then to convert the problem to CNF and run satz-rand for 10 restarts with a cutoff of 100 backtracks. Note that no one cutoff value is ideal for all domains. One way to address the problem is to specify a sequence of increasing cutoff values in the solver schedule.

The newest version of **blackbox** includes an additional solver, relsat [13] based on dependency-directed backtracking, as well as a technique for reducing the size of the CNF encodings by suppressing the generation of clauses that are logically redundant.

2.2 HSP: Heuristic Search Planner

HSP is a planner based on the ideas of heuristic search. Heuristic search algorithms perform forward search from an initial state to a goal state using a heuristic function that provides an estimate of the distance to the goal. The 8-puzzle is the standard example of heuristic search and is treated in most AI textbooks [25, 26]. The main difference between the 8-puzzle and the approach to planning adopted in HSP is in the heuristic function. While in domains specific tasks like the 8-puzzle the heuristic function is given (for example, as the sum of the Manhattan distances); in domain independent planning, it has to be derived from the high-level representation of the actions and goals.

2.2.1 Heuristic

A common way to derive an heuristic function $h(s)$ for a problem P is by relaxing P into a simpler problem P' whose optimal solution can be computed efficiently. Then, the optimal cost for solving P' can be used as an heuristic for solving P [26]. For example, if P is the 8-puzzle, P' can be obtained from P by allowing the tiles to move into *any* neighbouring

position. The optimal cost function of the relaxed problem is precisely the Manhattan distance heuristic.

In STRIPS planning, the heuristic values for a planning problem P can be obtained by considering the "relaxed" planning problem P' in which all *delete lists* are ignored. In other words, P' is like P except that delete lists are assumed empty. As a result, actions may add new atoms but not remove existing ones, and a sequence of actions solves P' when all goal atoms have been generated. As in recent planners such as SATPLAN [16] and GRAPHPLAN [2], action schemas are assumed to have been replaced by their ground instances, and variables are not dealt with.

It is not difficult to show that for any initial state s, the optimal cost $h'(s)$ to reach a goal in P' is a lower bound on the optimal cost $h^*(s)$ to reach a goal in the original problem P. The heuristic function $h(s)$ could therefore be set to $h'(s)$ and obtain an informative and *admissible* (non-overestimating) heuristic. The problem, however, is that computing $h'(s)$ is still NP-hard.[8] Therefore, an approximation is used: the heuristic values $h(s)$ are set to an *estimate* of the optimal values $h'(s)$ of the relaxed problem. These estimates are computed as follows.

Starting with $s_0 = s$ and $i = 0$ s_i is expanded into a (possibly) larger set of atoms s_{i+1} by combining the atoms in s_i with the atoms that can be generated by the actions whose preconditions hold in s_i. Every time an action that asserts an atom p is applied, a measure $g_s(p)$ is updated that is intended to estimate the difficulty (number of steps) involved in achieving p from s. For atoms $p \in s$, this measure is initialized to 0, while for all other atoms $g_s(p)$ is initialized to ∞. Then when an action with preconditions $C = r_1, r_2, .., r_n$ that asserts p is applied, $g_s(p)$ is updated as:

$$g_s(p) := \min [\ g_s(p)\ ,\ 1 + \sum_{i=1,n} g_s(r_i)\ ]$$

The expansions and updates continue until these measures do not change. When all preconditions involve a single atom, this is a Bellman-Ford procedure for computing costs in a graph from a given set of sources. The nodes are the atoms, the sources are the atoms p such that $g_s(p) = 0$, and edges $p \rightarrow q$ with cost 1 exist when an action with precondition p asserts q.

The heuristic function $h(s)$ used by HSP is defined then as:

$$h(s) \stackrel{\text{def}}{=} \sum_{p \in G} g_s(p)$$

[8] This was first pointed out by Bernhard Nebel.

where G stands for the set of goal atoms. This definition assumes, like decompositional planners, that subgoals are *independent.* The added value of the heuristic approach is that subgoals are weighted by a "difficulty" measure that makes it possible to regard certain decompositions as better than others. A result of this assumption is that the heuristic function $h(s)$ is not *admissible.* On the other hand it is often quite informative and can be computed reasonably fast.

2.2.2 Algorithm

The heuristic function defined above allows us to deal with any STRIPS planning problem as a problem of heuristic search. This means that planning could be carried out using algorithms such as A*. A*, however, might take exponential memory and approaches the goal too cautiously. In HSP, where the heuristic is recomputed from scratch in every node, it is necessary to use algorithms that can get to the goal with as few evaluations as possible. For this reason HSP uses a form of *hill-climbing* search. Surprisingly, hill-climbing works very well in many problems, and often produces good plans very fast. Sometimes, however, it gets stuck in local minima. In such cases, it is convenient to proceed with the search until a number of such impasses has been encountered, restarting the search if necessary, up to some specified maximum number of times. The algorithm used in the competition is a variation of this idea that also uses memory to keep track of the states that are visited. Current effort is directed towards identifying ways to speed up the evaluation of the heuristic, so that more systematic search algorithms could be used.

2.2.3 Implementation

HSP is implemented in C. Likewise, a preprocessor converts any STRIPS problem in PDDL into a C program that is then compiled, linked, assembled and executed. This usually means a time overhead in the order of a second or two in small planning problems but pays off in larger ones.

2.2.4 Related Work

HSP is based on the planner reported in [3]. This planner, called ASP, uses the same heuristic function but a different search algorithm based on Korf's LRTA* [18] designed for real-time planning.

An independent proposal that also formulates planning as heuristic search is McDermott's [22]. His system, UNPOP, also searches forward from the initial state to the goal but derives the heuristic values from a regression analysis. The resulting heuristic estimates, however, do not appear to be as informative, and the range of problems that can be solved and the quality of the solutions that are found do not appear to be as good as those obtained by HSP.

2.2.5 Current Work

The bottleneck in HSP is the computation of the heuristic values which are obtained afresh in every new state. Currently work is exploring methods that avoid this recomputation and that can speed up the algorithm considerably. At the same time, the quality of the resulting plans can be improved by continuing the search in a "branch and bound" manner after finding the first solution. Further details and code can be found at the HSP website[9].

2.3 IPP

The IPP planning system extends the fundamental use of planning graphs underlying GRAPHPLAN [2] to include conditional effects in actions [17]. Based on the original GRAPHPLAN code, several extensions and improvements have been made:

1. Changes in data structures, and in particular in the way that information about exclusive pairs is handled, have been made in the C source code of IPP 3.3. This has lead to a significant speed up and important reduction in memory consumption. For example, IPP 3.1. solved the Towers of Hanoi problem with 8 discs in 93 minutes using 900 Mbytes memory on a Sun Ultra I/170. IPP 3.3. solves the same problem in 8 minutes using 227 Mbytes. Nevertheless, the STAN planner used even more efficient data structures and saves several of the computations that IPP unnecessarily does repeatedly. These techniques are currently incorporated into IPP 3.4 and reduce memory consumption to 9 Mbytes and runtime to 40 s for this example.

2. An improved instantiation procedure determines the valid ground instances of actions by taking into account so-called *inertia*, i.e., facts

[9] `http://www.ldc.usb.ve/~hector`

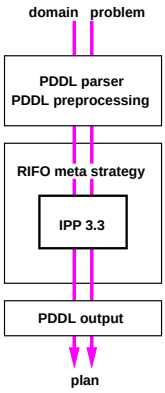

Figure 2: IPP architecture

that are true in all states. For example, in the Grid domain used in the competition only those instances of *move* actions are generated for which *connected* relations are found in the problem description.

3. A complete subset test [12] replaces the original incomplete test used in GRAPHPLAN and helps to identify more unsolvable goal sets without performing any search.

To run IPP in the competition, a parser for PDDL was developed using flex and bison. Since IPP took part in both the STRIPS and ADL tracks, a preprocessing phase based on [8] is included in the parsing process that translates PDDL first-order formulas into the disjunctive normal form required by IPP. [10].

Based on RIFO [24], a meta-strategy has been "wrapped" around the planner, which is activated when two threshold parameters, the number of objects and the number of ground actions, are exceeded by a planning problem. These parameters can be set by hand.

RIFO has been developed to cope with the problem of irrelevant information in planning graphs. When building graphs exhaustively starting in the initial state, a GRAPHPLAN-based planner will add lots of facts and actions that are totally irrelevant in solving the planning problem at hand.

[10] Further information about IPP can be found in the IPP home page http://www.informatik.uni-freiburg.de/~koehler/ipp.html

By building a backchaining tree, RIFO can exclude irrelevant information at various degrees of strength, but at the price of incompleteness: sometimes too many actions are ruled out making the planning problem unsolvable. The meta strategy runs RIFO on the problem input using the strongest reduction heuristic, which selects one cardinality-minimal set of initial facts and objects and only those ground actions that were used in the backchaining tree from the goals to the initial state. If IPP determines the reduced problem to be unsolvable, which it usually does very quickly, because of the effective termination test, the search space is enlarged by applying RIFO again but using a weaker heuristic, restoring all the previously excluded actions, which use objects that are potentially relevant. If IPP still cannot solve the problem, the original search space is restored to retain completeness of the planner. In the worst case, this leads to some overhead if both RIFO heuristics turn a solvable planning problem into an unsolvable one or if an unsolvable problem is investigated.

2.3.1 Handling of Conditional Effects in IPP

IPP represents conditional effects directly in planning graphs and also propagates mutual exclusion relationships between facts occurring in conditional effects and their effect conditions. But it does not extend GRAPHPLAN's notion of interference between actions. This has attracted some criticism in [1] where such a propagation of mutex relations over actions is proposed.

The approach in the SGP system described in [1] relies on the "factored expansion" of actions. This means that each action is split into its separate conditional effects, each with the same preconditions as the original action, and each with their own effect conditions as effects. All facts are made conditional; that is, unconditional effects have an empty effect condition. This is almost identical to the representation of effects in IPP where each effect points to its effect condition, while the unconditional effect points to the empty effect condition. However, actions in IPP are still treated as atomic units, while SGP creates "components" similar to GRAPHPLAN's STRIPS actions that represent each conditional effect in the graph. This opens the possibility to mark component nodes as exclusive, which is claimed to lead to much better performance of a planner.

A careful analysis of the SGP *induced mutex relations* that are calculated between components reveals that they do not lead to any search space reduction when compared to the approach that IPP uses. Consider the following example with the two actions

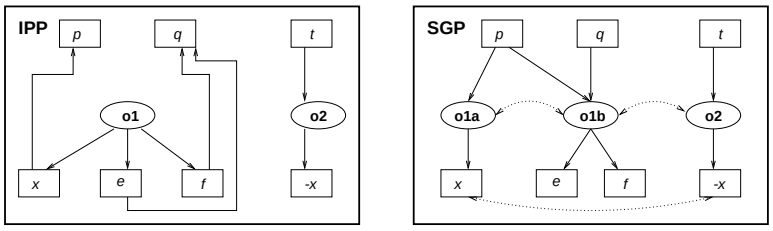

Figure 3: Representation of Conditional Effects in IPP compared to SGP.

$$\mathbf{o_1} : \top \qquad\qquad \mathbf{o_2} : t$$
$$p \Rightarrow x \qquad\qquad \top \Rightarrow \neg x$$
$$p, q \Rightarrow e, f$$

Figure 3 shows the different representations in both planners. IPP keeps a single action node o_1 that has three facts as conditional effects, but no precondition. The effect nodes point to their effect conditions. SGP splits o_1 into two component nodes o_{1a} and o_{1a}. Each of them has preconditions and unconditional effects and can therefore be considered as a STRIPS action.

In IPP the actions are non-exclusive because only the conditional effect of o_1 interferes with o_2. Consequently, the planner would consider them as a valid choice to reach a goal set.

In SGP, the components o_{1a} and o_2 are mutually exclusive because of their inconsistent effects, which are no longer conditional. In order to be able to propagate mutex relations over components, SGP tests whether one component *induces* another component. In the example, both components o_{1a} and o_{1b} *induce* each other: the precondition of o_{1b} implies the precondition of o_{1a}, therefore o_{1b} induces o_{1a}. The precondition q of o_{1b} cannot be prevented in the graph because $\neg q$ is not contained in the fact level, therefore o_{1a} induces o_{1b}. This means, whenever one component is selected to achieve a goal, the other must be executed as well. Because of the *induced* relation between the two components, o_2 is also made exclusive to o_{1b}, so an *induced mutex relation* is additionally established by SGP.

Now let us assume that the goals are $\neg x, e, f$. SGP fails immediately because no non-exclusive set of components can be found to achieve the goals. IPP selects the two action nodes and constructs the new goal set t, p, q from preconditions and the effect conditions of selected conditional

effects. Then it checks for conflicts of conditional effects and finds that o_1 has a conditional effect x that is inconsistent with the goal $\neg x$ and therefore this effect has to be prevented in the state where the action is executed. The only way of doing this, is to add $\neg p$ to the goal set, *confronting* the corresponding effect condition. But this makes the goal set inconsistent and therefore IPP fails on this selection before it enters any search at all.

The Movie domain from the planning competition has been designed to exhibit the property of induced mutex relations. However, in this domain IPP 3.3 was able to detect that the effect condition of the single conditional effect that is contained in the actions in this domain comprises inertia only. Consequently, the inertia are removed and the conditional effect becomes unconditional. Thus, IPP derives additional mutex information and outperformed SGP in the competition in this domain. A joint discussion with Weld and Anderson led to a revised Movie domain in which no inertia can be removed by IPP 3.3 (see the description of SGP below). With this revised action set, IPP is quite slow because almost no mutex relations can be derived and subset memoization fails completely in reducing search because a redundant parameter in all *get*-operators yields many irrelevant action instances. When using RIFO as a preprocessor to filter out all irrelevant information, IPP outperforms SGP again.

It is an open question which of the techniques is better than the other one. In general one can say that induced mutex relations seem to be very rare: there is no other domain in any current benchmark sets, other than the Movie domain, where these relations seem to occur or offer an important advantage. Besides this, the Movie domain has been solely designed with the purpose to prove the usefulness of SGP's action component representation and does not appear to be very natural. On the other hand, removal of inertia and irrelevants seem to be more natural techniques and both phenomena occur quite frequently in planning domains.

2.4 SGP

Sensory Graphplan (SGP) is a sound, complete planner based on Blum and Furst's Graphplan. SGP includes support for conditional effects, universal and existential quantification, uncertainty, and sensing actions. SGP was a participant in the AIPS-98 planning competition, competing in the ADL track. SGP was the only planner written in Lisp to enter the competition, and, despite the performance limitations from the language, competed well. In this section, we discuss the features that SGP supports, how SGP's algo-

rithms work, and SGP's performance in the AIPS-98 planning competition.

Sensory Graphplan includes many features that would be expected in a state-of-the-art planner: support for ADL [27], including conditional effects, universal and existential quantification, and typing; ability to read domain and problem descriptions in PDDL [23]; and some performance optimizations. SGP also has a number of features that set it apart from the other Graphplan-based planners. These unique features, described below, are factored expansion for handling conditional effects and the ability to handle uncertainty and sensing actions. SGP is written in well-documented Lisp code and can be downloaded from SGP's homepage[11]). SGP continues to be supported by the University of Washington, with new bug fixes and new releases being produced as appropriate.

2.4.1 Factored Expansion of Conditional Effects

Many of ADL's expressive features are easy to implement in Graphplan, but handling conditional effects is surprisingly tricky. Conditional effects allow the description of a single action with context-dependent effects. The basic idea is simple: we allow a special `when` clause in the syntax of action effects. `When` takes two arguments, an *antecedent* and a *consequent*; execution of the action will have the consequent's effect just in the case that the antecedent is true immediately before execution (much like the action's precondition determines if execution itself is legal). And this is exactly the reason why handling actions with conditional effects is tricky – because the outcomes of the actions depend on the state of the world when the action is executed.

Factored expansion was first described in [1], where it was compared to full expansion [8] and IPP's [17, 24] condition effects handling method. The idea behind factored expansion is two fold. First, all actions that contain conditional effects are *factored* into components, with a single effect (conditional or unconditional) in each component. Each component can then be treated, more or less, just like a STRIPS action. Second, when the components are being examined for graph expansion and backtracking search, the interaction between components from the same action is considered. In particular, there is the new concept of component *induction*: component C_n *induces* C_k at level i if it is impossible to execute C_n without executing C_k.

Component induction leads to changes in both the Graphplan mutual exclusion (mutex) rules and in how the backtracking search for a solution

[11] `http://www.cs.washington.edu/research/projects/ai/www/sgp.html`

proceeds. The change to the mutex rules is the addition of a new rule that can identify more mutexes: Two components C_n and C_m at level i are mutex if there exists a third component C_k that is mutex with C_m, *and* C_k is *induced* by C_n at level i. The change to the backtracking search requires that, when a component is selected to establish a goal, all other components from the same action instance must also be considered.

An example of a domain where the factored expansion approach shows its strength is the Movie domain[12]. The goal of the Movie domain is to collect snacks and then watch a movie. Before the movie can be "watched", the movie must be rewound and the VCR's counter must be set to zero. Consider what the planning graph looks like after one level has been built: the VCR's counter can be reset, the tape can be rewound, and snacks can be fetched. The key thing, however, is that the actions to rewind the movie and to reset the VCR's counter are mutex *only* if their conditional effects are considered. The IPP method of handling conditional effects does not find this mutex, and thus IPP must perform an exhaustive search of the planning graph to determine that no plan yet exists.

SGP, on the other hand, can identify the induced mutex between the two actions. Thus, SGP immediately knows that no plan yet exists, and can thus proceed to the next planning graph level.

2.4.2 Uncertainty and Sensing Actions

The second novel feature in SGP is its ability to deal with uncertainty and sensing actions (this work is detailed in [32] and [33]). Although this part of SGP wasn't used at the AIPS-98 planning contest, we discuss it here in some detail because it was the motivation for our work on SGP and is the characteristic that distinguishes SGP from the other planners.

Classical planners assume complete knowledge of the world at plan time, and assume that the outcome of actions is certain. Although this assumption simplifies the planning process, it is an unrealistic one. SGP relaxes the assumption that the agent knows the state of the entire world *a priori*.

The first step in relaxing this assumption is to allow the initial state to include propositions whose truth values are uncertain (either T or F). Internally, SGP represents this sort of uncertainty by keeping track of each *possible world*. For example, if both propositions P and Q are uncertain, then there are four possible worlds: $P \wedge Q$, $P \wedge \neg Q$, $\neg P \wedge Q$, $\neg P \wedge \neg Q$. For

[12]The Movie domain was originally distributed with PDDL. A revised version of the domain is available in the SGP distribution.

each possible world, SGP builds and maintains a separate planning graph. This task is complicated by the fact that, because of conditional effects, an action's outcome may be different depending upon the world it is executed in. It should be noted here that SGP makes the assumption that an uncertain proposition may be T or F but does not make any assumptions about probabilities (SGP simply records that a proposition is uncertain, and does not associate a numerical weight with each possibility).

The second step in relaxing the complete knowledge assumption is to limit the agent's observational powers to explicit sensing actions. Sensing actions are actions in the domain whose effects include a special `sense` statement. `Sense` statements are used to let the agent query the world for the truth value of a proposition. Based on the sensed truth value, the agent can then refine the subset of possible worlds it is in (for example, if a proposition P is sensed to be T, then the agent knows that it is in one of the possible worlds in which P is T).

Given the set of planning graphs (one for each possible world) and the set of actions, including sensing actions, SGP builds a plan that will achieve the goals in all possible worlds. To guarantee success in each possible world, the plan may include actions that are to be executed only in some of the possible worlds. Thus, the plan may have branch points contingent on the consequences of sensory actions. The plan is a DAG, with the possibility of branches rejoining with each other. The contingencies in the plan are based on which possible world the agent is in at plan execution time. To resolve which possible world the agent is in, the results of the sensing actions are used. An important feature of SGP is that the planner determines exactly which sensing actions need to be taken to resolve the uncertainty. There are two important outcomes of this point. First, the planner does not assume that the agent has full observational power. Instead, SGP allows only the terms sensed by the sensing actions to be used in resolving uncertainty. Second, the agent does not have to resolve exactly *which* possible world it is in, but rather which of a set of possible worlds it is in. This distinction is important in that, with SGP, the agent needs not necessarily resolve all the uncertainty, but simply "enough".

2.4.3 SGP in the AIPS-98 Competition

SGP made its first public debut at the AIPS-98 planning competition. The original purpose of SGP was as a research vehicle for the exploration of conditional effects and planning under uncertainty, so we did not expect

to pose a serious competition to other, heavily optimized planners. The planning competition demonstrated many specific strengths and weaknesses of our system, the details of which are discussed below.

One of the major features of SGP is it's identification of induced mutexes in factored expansion. In [1], we showed the effectiveness of induced mutexes by comparing SGP's performance on the Movie domain to that of IPP. The Movie domain as we designed it was the simplest possible example of a domain in which the discovery of induced mutexes was necessary. In our experiments, we compared SGP to IPP 3.2 (the then-current version). We found that SGP correctly identified the induced mutexes and thus eliminated a great deal of search. IPP, on the other hand, failed to find these mutexes, and performed poorly, requiring much fruitless search. The results from the competition present a different view. In the competition, IPP 3.3 was used, and was able to simplify the Movie domain by removing inertia (this optimization was not available in IPP 3.2). Inertia (also called static propositions) are propositions whose truth values never change as a result of any actions in the domain; thus, these propositions can be "constant-folded" out of the problem. Close examination shows that removing inertia from the Movie domain *completely eliminates* the conditional effects of the actions, and hence IPP 3.3 had no trouble identifying all the mutex conditions!

Although IPP 3.3 performs well on the original Movie domain, we believe the need for induced mutexes is not obviated. Following the competition, we constructed a slightly more complex domain, Movie-2, which cannot be simplified by removing inertia. In our tests, we found that SGP still performed well in Movie-2, but that IPP 3.3's performance deteriorated quickly. See table 4 for the results of this test.

We observe, however, that IPP has other optimizations that allow it to perform better than SGP. In particular, the RIFO [24] method of removing irrelevant objects effectively reduces the planning graph size. Thus, even though IPP has to perform an exhaustive but futile search, it can do so quickly. Additionally, IPP is implemented in C, with much care given to code speed. Thus, even without the advantage of identifying the induced mutexes, IPP can perform quite well.

The planning competition highlighted a number of places where SGP could be improved. One recurring problem we faced was planning graphs that were too large. A few possible solutions have been discussed before:

- **In-place graph expansion.** Nodes in the planning graph are not replicated at each level, but rather annotated with the level where

	SGP 1.0d	IPP 3.2	IPP 3.3	IPP 3.3+RIFO
movie-watching				
5	0.01	0.06	0.01	0.01
8	0.01	0.53	0.01	0.01
11	0.03	2.53	0.01	0.02
14	0.03	11.79	0.01	0.03
movie-watching-2				
5	0.02	0.12	0.10	0.02
8	0.03	0.73	0.65	0.03
11	0.05	2.97	2.68	0.03
14	0.07	8.94	7.99	0.03

Figure 4: Comparison of SGP and IPP on Movie domains. All times are seconds of wall-clock time on a 300MHz Pentium II with 256MB of memory.

they appear. This idea was discussed in [30] and implemented in [21] and as part of [31].

- **Removing irrelevant objects.** RIFO [24] is a method of removing objects in the domain from consideration when expanding the planning graph. Backward focusing [14] is a way of analyzing the problem so as to limit the graph expansion to exclude any objects or actions that cannot possibly contribute to achieving the goals.

2.5 STAN

STAN (STate ANalysing planner) is based on GRAPHPLAN, but extends this planning technology in several ways:

1. A more sophisticated data structure is used to store the graph and to aid in its construction than was utilized in earlier versions of GRAPHPLAN.

2. More analysis is carried out on the graph and the problem it encodes to reduce search branches, including *goal ordering analysis*, *symmetry analysis* and *resource analysis*.

3. A *wavefront* is exploited at the fix-point of the plan graph to avoid unnecessary additional work.

4. State-analysis techniques, implemented as a module of STAN which can be exploited as a planner-independent system, TIM, are used to acquire information about a domain and problem encoding which is exploited in instantiation and filtering of the the plan graph.

The implementation of the plan graph in STAN is based on a careful reuse of much of the central graph structure, to avoid copying layer-independent information repeatedly, together with bit-level operations to support graph construction and careful filtering of the layer-dependent structures to reduce the retesting carried out at subsequent layers. This implementation, the most recent version of which is described in full in [21], is efficient and relatively compact, although the competition version of STAN (STAN version 1.0) still allowed unnecessary wastage of space.

The plan graph is carefully analyzed during construction to produce several auxiliary relationships between goals and between actions at each layer. In particular, STAN identifies ordering relations between pairs of goals reflecting the order in which they must be satisfied so that both are true at a given level. This allows considerable reduction of the search space by automatically selecting *noops* for goals which must be satisfied earlier than the current layer during search. Furthermore, certain chains of ordered goals can prevent an entire goal set from being achieved at a given layer without any search at all. STAN also identifies other goal sets as unachievable when they exceed *resource* limitations imposed by the domain. Resources include *numbers of objects* in certain configurations, the *rate* at which certain objects can be put into key configurations during the plan execution and other factors as well as the physical resources available to the planning agent (such as grippers, fuel, containers and so on). Limits on the availability of abstract resources, such as the rate at which objects can be configured, arise from the limits imposed by the physical resources of the agent and are expressed, for example, in terms of constraints on the number of plan graph levels that must have been built before a certain goal configuration can be achieved. The resource analysis performed by STAN can dramatically reduce search, by identifying the minimum number of graph layers that must be built, in some domains including the TSP domain discussed below. Finally, STAN exploits structural features of the problem domain, such as *symmetry*, to reduce search. The competition version of STAN performed only a very preliminary symmetry analysis, in which symmetric objects (those which are indistinguishable and hence do not form interestingly different action instantiations) were identified to reduce the number of action instantiations

produced. Although STAN was able to detect certain forms of symmetry, the competition version was not able to exploit it fully. In STAN version 3 we are working on a much more advanced utilization of the symmetric structure of domains.

One of the most important features of STAN that distinguishes it from other GRAPHPLAN-based planners is its use of a highly efficient *implicit* representation of the graph beyond the fix-point. STAN avoids both construction and search of the plan graph beyond the fix-point, where the graph is static. Instead, search is built on a collection of candidate goal sets which are generated as failure sets at the fix-point and which are pushed one layer forward to be retried. This process in which failed goal sets are promoted forward forms a kind of rolling collection of goal sets at the fix-point, which we call a *wave-front* [21]. The efficiency gains this mechanism offers can be dramatic in certain problems and can also significantly reduce the memory demands during the process of constructing a solution.

The exploitation of the results of *state analyses*, of various kinds, is an important feature of STAN. State analyses can be done in a pre-processing stage, using techniques which are planner independent, and the results fed into the planning process and used to reduce, and even eliminate, some of the more resource intensive aspects of planning. Symmetry analysis, in which symmetric objects and actions are identified in order to prune redundant search, is one such form of state analysis. Another technique, currently being integrated with STAN version 3, is the types and state invariant inference analysis performed by the TIM module [6]. The competition version of STAN was not fully integrated with TIM and therefore could not make use of the inferred state invariants. It could exploit the type structure inferred by TIM, but the competition domains had types supplied so this did not give STAN as large an advantage over the other competitors as can be achieved with some domain encodings.

During the development of the competition version of STAN a pattern was established of exploring the behavior of the planner on a family of problems in order to understand what made them hard, followed by the construction of techniques to tackle the source of the difficulty in a domain-independent way. An important lesson learned from this process, and from the competition itself, is that problems are hard for a wide variety of reasons and that these different sources of difficulty can lead to the development of techniques which are powerful in certain contexts but are simply useless overhead in others. Of particular interest to us were problems which appeared hard for STAN and yet were easy for other planners, or are intuitively easy.

One of the reasons problems can be hard, affecting GRAPHPLAN-style planners in particular, is that domains can contain huge collections of instantiated actions which have no useful role in the plan. This leads to both a large cost in the construction phase and also, often, a large cost in the search phase when many branches must be explored. These branches often express the same fundamental planning decisions but, because of the search in grounded actions, differ in details which are insignificant from the planning point of view.

STAN attempts to deal with this problem in several ways: type-inference leads to a potential reduction in the numbers of instantiated actions; the use of static conditions in instantiation and some preliminary work on filtering to remove some objects in some domains.

There remains much work to do in this area: RIFO offers IPP a huge benefit in many problems, by filtering out many irrelevant objects, actions and facts, but at the price of possible incompleteness. As can be seen in Section 3, all of the planners were confounded when confronted with problem instances containing large numbers of ground operator instances. The ability to reduce these by intelligent filtering would appear to be a critical element of successful planning.

In contrast, some problems are hard not because of the cost of construction of the graph, but because of the multiplicity of search paths in the problem. This is particularly problematic in cases in which the problem appears to be solvable long before it actually is (the graph contains the non-mutex goals long before the solution layer). An example of a problem like this is the Complete-Graph Traveling Salesman Problem in which the graph is completely connected so that the traveler is unconstrained in the order of visits he makes. In this case, the problem is that all of the destinations could be visited within a single step and any pair could be visited after two steps. Thus, the problem appears solvable after two steps, but is not actually solvable until n steps have passed, where n is the number of sites to be visited. However, as n grows the number of possible paths the traveler might have taken explodes exponentially, so that the search problem quickly becomes intractable. STAN uses resource analysis to determine that only one destination can be visited per step, so does not search for a plan until n layers are constructed and can also ensure that every layer is used for a visit to a hitherto unvisited site.

Unfortunately, there are other problems with similar character which are not yet adequately tackled by STAN. The Gripper problem used in the competition is an example: STAN correctly determines that no more than

two balls can be deposited per time step. However, it does not allow for the fact that the two balls must also be picked up and transported, requiring an additional two steps (and a third to return to the source for another load), so the the resource analysis does not help as much as one might hope.

GRAPHPLAN style planners also suffer during search from a problem of premature commitment. This occurs because the use of *ground* action instances forces selection of objects to play particular roles in a plan often before constraints which would govern their choice become apparent. Constraint-solving planners can benefit in these problems by making choices in the highly constrained parts of the plan and propagating them into the less constrained parts of the plan. Forward- and backward-chaining planners do not direct their planning strategies by exploiting the most highly constrained parts of a plan structure and this can lead to costly mistaken choices which must be retracted.

Some problems are hard because they have an inherent combinatorial cost, while others are, in principle, easy, yet prove hard to current planning technology, or at least to some current planning strategies. The Gripper domain is a good example of the latter – a domain in which problems are trivial for human problem solvers and yet an optimal plan for this domain eluded all of the planners in the competition for instances larger than a dozen balls. More recent work on STAN has explored the exploitation of symmetry in problems such as Gripper which can reduce the difficulty dramatically [7]. Nevertheless, there is clearly much work to be done on recognizing and exploiting features of problems that humans appear to identify with ease. Furthermore, the representation of solutions to problems such as these remains an issue: no human would represent the solution to problems in the Gripper domain as an explicit sequence of steps, but as a method for generating those steps during execution ("Transport the first two balls from room a to room b, then return and repeat until all balls are transported." – the fact that the "first two balls" are not identified by name is an indication of the role of symmetry for human problem solvers).

3 Review of Results

One of the tasks the Competition committee faced was to determine a strategy for evaluating planner performance. Before the competition a formula was proposed which combined weighted values for plan length and planning time, adjusted to reflect the relative performance of different planners

on the same problem (to give due credit to planners which quickly solved problems that defeated many of the others). In the event, this formula was judged to give counter-intuitive results and it was essentially abandoned. This left a void in the final evaluation of performance and it remains, in the STRIPS track, a difficult task to assess the relative performances of the planners. A summary of the results was presented at the event but this is crude, in that it fails to differentiate between good performance on simple problems and good performance on very hard problems. This masking is amplified by the fact that each of the planners faced minor problems due to program bugs which made what would have otherwise been simple problems appear hard for those planners. In addition, in the Mystery domain several problems were unsolvable (and proven such by some of the planners) but these problems were ignored in the results summary. Overall, the results of the individual planners all showed strengths and weaknesses and it is not surprising that a simple direct comparison proved unsatisfactory in the competition. This remains an unsolved challenge for future competition events and the community must be wary of setting up targets (in whatever form) which over-simplify the objectives of planners. The problem that evaluation represented suggests that these objectives remain a complex and clouded issue.

It is worth emphasizing that three of the planners running in the STRIPS track (**blackbox**, IPP and STAN) all produce *parallel optimal plans* (although IPP does not guarantee to do so when using its RIFO machinery), which, when linearized, will not always lead to optimal sequential plans. This can explain the discrepancies in plan lengths discovered by these planners. In general, the length of the linearized plan is difficult to control when using a mechanism which produces optimal parallel plans. HSP produces linearized plans, but does not support claims for optimality. An interesting example to consider is the seventh Logistics problem in the first round of the competition, where HSP produced the only plan found by any of the planners. This was 112 steps long. STAN has subsequently demonstrated that there is a 37 step plan!

3.1 The First Round: STRIPS track

The competition involved the use of five domains in the first round and three in the second. The first round used the Gripper domain, the Movie domain, Logistics, the Mystery and Mystery-Prime domains. The last two are variations on transportation domains, with limited fuel. In the last

domain the fuel can be piped between nodes in the transportation network, while in the Mystery domain the fuel is held at its starting depots. In the first round, 30 problems were presented of each type, except for Gripper, in which just 20 problems were presented.

The characteristics of all of the last three domains are similar, in that they are all transportation problems involving moving objects around a network of locations as efficiently as possible. Interestingly, these all had a similar performance limit for all of the planners: no planner could solve a Logistics problem with more than 10000 ground action instances, and all the problems with fewer than 10000 ground action instances were solved by at least one of the planners.[13] Mystery-Prime proved more tractable, with problems including as many as 24290 ground action instances being solved by some planners (although the problem instance with 24290 actions involved producing a plan with only four steps). However, problems with over 10000 ground action instances still proved hard in general, with several planners failing on these large problems and inconsistent performance being demonstrated between the threshold of 10000 ground action instances and the largest solvable problems. More than half of the Mystery problems presented in the competition were under 10000 ground action instances in size. Of the 13 problems that exceeded this size, five are unsolvable and three were solved in the competition by at least one of the planners. The other five problems are all solvable, at least with STAN, although buffer sizes were set too small in the competition configuration for it to solve them. STAN uses an object filtering mechanism which worked successfully in both the Mystery and Mystery-Prime domains to cut the numbers of ground action instances so that in the Mystery domain none of the problems presented actually produced more than 8000 ground action instances.

Interestingly, of the 30 Mystery domain problems presented in the competition, 11 were proved unsolvable by at least one planner, rather than simply proposed unsolvable because of a lack of resources. This distinction was not used in the competition (presumably because of the difficulty in distinguishing an accurate claim that a problem is unsolvable from a lucky guess when resources run out), but the three GRAPHPLAN-based planners used in the competition are capable of identifying problems of this kind (at least in principle), while **blackbox** can identify some unsolvable problems (those

[13] The numbers of ground action instances have been computed using STAN, with all filtering mechanisms turned off. STAN uses TIM to generate a type structure for each domain and this can result in fewer ground action instances being generated than would be the case with raw instantiation.

in which some of the goals are unreachable, or are pairwise unreachable). STAN was fastest in demonstrating 10 of the 11 problems to be unsolvable (on average it was 15 times faster than its nearest rival at showing these problems unsolvable), and IPP was fastest on the remaining problem.

The Movie domain presented no difficulty to any of the planners and performance times were so small that they cannot really be usefully compared. This domain was included in order to consider its effect in the ADL track, as was discussed in Section 2.4. The Gripper domain is peculiar in that it is a domain which is intuitively easy to solve – the problems present no difficulty for a human planner – and yet only HSP was able to solve instances involving more than 12 balls. Performance of all the other planners deteriorated exponentially with the increasing problem size. IPP used RIFO in this problem and this allowed it to solve more problems than would otherwise have been possible, by excluding one gripper from consideration. HSP produced similarly sub-optimal solutions by carrying only one ball on each trip. The reason this problem is so hard is that there are so many ways in which the actions can *almost* solve the problem with a shorter sequence than is actually required to completely solve it and these garden-path sequences increase exponentially with two grippers, despite the fact that even the hardest problem instance presented in this domain contains only 340 ground action instances! One of the reasons this problem appears to be so simple for a human planner is that a human can see the essential symmetry to the problem and can exploit this to simplify the problem to the extent that it becomes trivial. This problem is one which highlights a critical weakness of the current fast planning technology.

In round 1, of the 140 problems presented, 98 problems were either solved or proved unsolvable by at least one of the planners. At least 6 of the remaining problems have been solved by one or more of the planners since the competition. The problem domain which proved difficult for all the planners was the Logistics domain, accounting for 25 of the unsolved problems.

3.2 The Second Round: STRIPS track

In the second round a new domain was introduced: the Grid domain. IPP managed to solve 3 of the 5 problems presented, using strong RIFO pruning and producing sub-optimal plans. The other planners managed only one problem in this domain. The problem sizes ranged from 2609 ground action instances for the simplest through to 16239 (unsolved by any planner). IPP

problem	original	RIFO strong	RIFO weak
log1	571/25 (13)	-	-
log2	502/21 (20)	-	-
log3	958/26 (27)	-	-
log4	3561/42 (-)	189/30 (-)	-
log5	4985/53 (-)	119/26 (31)	-
mprime1	7809/36 (5)	7/9 ($\perp$)	11/9 ($\perp$)
mprime2	3281/32 (8)	-	-
mprime3	97259/68 (-)	-	-
mprime4	8485/42 (5)	7/10 (4)	-
mprime5	6773/22 (6)	-	-
grid1	2610/38 (14)	-	80/11 (20)
grid2	4501/50 (-)	69/19 ($\perp$)	260/19 (31)
grid3	7256/64 (-)	315/21 ($\perp$)	557/21 (-)
grid4	11151/80 (-)	135/24 (47)	-
grid5	16240/98 (-)	1481/49(-)	-

Figure 5: The effects of exploiting RIFO with IPP.

solved an instance with 11150 ground action instances.

The other domains used were Logistics and Mystery-Prime. All but one of the problems presented in these domains were solved by at least one planner. All of the Logistics problems were under 5000 ground action instances (although the two hardest of these involved few planes and, in one, many trucks, making for a big search space). All but one of the Mystery-Prime problems were under 6000 ground action instances, and the exception contained 19730 ground action instances. This instance defeated all of the planners in the competition, although at least one of the planners has since generated a 33 step plan which solves it. With the exception of a single instance (traced to a trivial program bug) all of the other problems were solved by all of the planners.

In the light of the observations already made about the sizes of the problems measured in terms of numbers of ground action instances, it is interesting to consider the behavior of IPP using the RIFO sub-system which filters some objects and action instances from domains prior to planning. RIFO was not used by IPP in the first round of the competition, except in the Gripper domain in which it was turned on by hand. In this domain,

RIFO identifies one gripper hand as irrelevant, so that only one ball is carried at a time and plans become much longer.

In the second round, IPP was run using the RIFO meta-strategy, which significantly reduced the search space for the planner. Table 5 lists the number of ground actions and objects in the original problem descriptions and, in brackets, the length of the plan if IPP could find one given a 10 minute CPU time limit and a 120 Mb memory limit, as in the competition, but on a SPARC 1/170 (which is approximately twice as slow). The table shows the number of selected ground actions and objects after the stronger and weaker RIFO selection processes. $\perp$ means the planning problem became unsolvable, $-$ means RIFO was inactive, $(-)$ means no plan was found because the planner either exceeded the CPU time or memory limit.[14]

As the analysis shows, only 8 problems can be solved without RIFO, but in using the meta-strategy 3 more solutions are found. The meta-strategy combines only 2 out of the 12 selection heuristics. Which combinations work for which examples has to be found out by experimentation. In the competition, no such experimentation was possible and therefore the selection of the heuristics was done long before the competition, following the intuition that one should try the strongest possible heuristic first because it leads to the smallest search space, but then relax it by allowing more actions when incompleteness occurs. The threshold parameters that decide whether RIFO is activated at all were set, to 3500 actions and 35 objects, after a few trials on some of the competition problems.

3.3 The ADL track

In the ADL track the same collection of domains and problems was used as in the first round of the STRIPS track. Of course, the domain encodings were reconstructed to exploit the ADL features. Only SGP and IPP competed in this track.

IPP solved all problems from the Movie domain, the first 5 problems from the Gripper domain containing 20 test problems, 13 out of 30 problems in the Mystery and Mystery-Prime domains, and 3 out of 30 problems in the Logistics domain. In total, it solved 69 problems in approximately 20 minutes, including all the 38 problems SGP solved. RIFO would not have

[14] The weaker RIFO heuristic is activated when either the stronger heuristic makes a planning problem unsolvable as, for example, in the case of the third Grid problem or when the number of ground actions remains below the threshold parameter of 3500 and only the number of objects exceeds 35 as in the first Grid problem.

improved the performance of IPP in this round because on most examples it makes the planner incomplete, so that the planner would have had to find the solution in the original search space after all reduction attempts had failed. In the Movie domain, however, the meta-strategy using the weaker heuristic succeeds in determining that only a handful (between 5 and 9) of the initial facts in each problem are relevant.

3.4 Commentary

Throughout the competition, no planner successfully solved any problem instance which involved more than 60000 ground actions. In fact, without filtering techniques to reduce the number of ground actions, no planner solved problems with more than 25000 ground actions and reliable performance was restricted to problems with fewer than 10000 ground action instances or so. In domains with harder search space growth problems, even fewer action instances could be handled. Although the number of ground action instances is not an infallible guide to the difficulty of problems, it is clearly an important indicator and strongly suggests that, at least for planners which work with ground actions during plan construction, there is much work to be done on the filtering process which could remove unnecessary actions from the problem space. It is interesting to observe that problem 15 of the first round Logistics domain lies beyond the scope of the competition planners even with a manual filtering of the domain objects, removing irrelevant packages and trucks, leaving as few as 3006 ground actions. The difficulty of this problem is not easy to understand, since there appears to be no pressure on the aircraft resources (with 6 aircraft available in just 3 cities), although the fact that the cities each have 6 locations could well be significant.

Although number of ground actions represents an important element in determining planner performance, the number of domain states is also a factor. Indeed, for planning systems which do not instantiate operators before planning, number of states might be a more important feature of the problems. It is not easy to compute the numbers of states for some problems (particularly Mystery and Mystery-Prime domains) since reachable states are non-trivial to determine. However, for Logistics and Gripper it is very straightforward. In the Logistics problems that were *solved* in round 1, the state spaces contained between 10^{10} states (problem 5) through to 8×10^{25} states (problem 11). These are clearly very large state spaces. By contrast, Gripper problems define state spaces containing a mere 256 states (problem

1) through to 68608 states (problem 4) and up to more than 4×10^{15} states in the largest (problem 20), solved only by HSP. The 376832-state problem 5 was beyond all the planners but HSP, and none solved it optimally. This analysis gives an indication of the dramatic contrast between the problems in which the planning technology is performing well and the problems where it demonstrates fundamental weaknesses.

3.5 Other Challenges

Although the size of domains, particularly measured by numbers of instantiated ground actions, represents a critical challenge to planning technology, the domains in the competition and others explored independently by the competitors have revealed other important problems which must be addressed.

For example, the Gripper domain highlights the combinatorial costs of exploring a large (and largely redundant) search space. The search must be reduced when this is possible and in the Gripper domain in particular there is huge potential for reduction in search costs. HSP demonstrated that heuristic search in this space can offer dramatic benefits. HSP solved all of the Gripper problems, where other planners managed at most 4 or 5. This domain alone accounts for three-quarters of HSP's significant lead over the other competitors in number of problems solved. HSP's heuristic effectively ignored the possibility of transporting the balls in pairs and solved the problems by transporting one ball at a time between the rooms. IPP, which succeeded in solving 5 of these problems, used its RIFO machinery which caused it to ignore one of the grippers, also leading to solutions in which only one ball was transported at a time. None of the planners could exploit the incredible degree of symmetry in the problem to cut the search space from its exponential size to reflect the trivial underlying problem.

The Logistics domain has the interesting property that the hardest part of the problem instances usually lies in the middle of the plan. The problems of transporting packages to and from airports, which sandwich the problem of flying packages between cities, are relatively easy but often generate large collections of redundant search paths. A planner which can tackle the core problem, the flying of packages between cities, and propagate necessary constraints outwards to the simpler ends of the plan, will have the advantage. This represents a single instance of a more general issue: many planning problems are not uniformly hard. A planner which can identify the hardest parts of a planning problem and concentrate on solving those parts first,

propagating constraints towards the easier, initially less constrained, parts of the problem will perform far better than a planner which always tackles the problems from the same place.

The Mystery domain is a fascinating variation on the transportation theme, introducing resource limits on carrier capacity and fuel, as well as an underlying route planning problem. This domain (and the Mystery-Prime variation) has the potential to produce problems which are hard for a wide variety of reasons. The lack of resources can make the route planning problem dramatically more complex as it interacts with the transportation of multiple packages. Similarly, the capacity limits can interact with fuel shortages to make it necessary to carefully coordinate the actions of carriers to cooperate in the transportation of objects. By varying the size of fuel dumps the problems can range from simple route planning (with abundant fuel) to complex scheduling of interacting carrier actions (with limited fuel).

The Grid domain, used in the second round of the competition, represents a further transportation domain on a grid-shaped network, but with constraints on the access to certain locations based on keys of appropriate shapes for the corresponding locks. This domain represents a difficult search domain, primarily because the domain forces the planner to use only one useful action at each layer. Tackling this problem requires an effective filter to remove ground actions, partly to reduce the cost of manipulating the domain itself, but mainly to reduce the number of redundant search paths, corresponding to multiple actual paths through the grid itself.

4 The Future

The first planning competition proved an extremely stimulating event for the planning community. It has brought into sharp focus the state-of-the-art in domain-independent planning and has offered the opportunity to identify several essential issues for the planning community to address. First of these is the development of a common planning domain description language, currently taking the form of PDDL. Although PDDL must be seen as still under development, the effort already invested in its development is an important step towards allowing planners to be usefully compared and in the construction of a generally useful repository of planning domain problems. Perhaps the closest the community has come to this in the past is the collections of problems included with particular planner releases, where those planners were widely used (UCPOP being an obvious example [28]).

PDDL has not yet been put to the test in its provision of HTN expressiveness, and there remain questions over the ADL components of the language. In particular, it has been proposed that nested conditional effects are an unnecessary element of the language. Provision for the expression of resource constrained planning problems also remains untested.

These observations highlight a second issue for the community to confront: it remains difficult to compare planners on equal footing. Planners can differ widely in terms of the expressiveness of the domain description language they handle, the expressiveness of the plans they produce, the speed of planning and range of domains they can successfully tackle. In order to make coherent progress in the field it is necessary to be able to compare potential advances, to attempt to coalesce different but compatible approaches and to avoid repeated redevelopment of the same basic planning software tools at dozens of different sites. A common domain description language is only one step in addressing this issue. It also requires components of planning systems be made available to the community, particularly in stable and adaptable forms. PDDL parsers are already being made available, and some of the code used in the competition is available as source to be modified, extended or adapted. These are essential steps in supporting the efforts of the community to advance beyond the current state-of-the-art.

A third issue arises from the hope to push the boundaries of the current state-of-the-art in planning: the benchmark domains which are used to establish the current levels of performance and to set targets for the next generation of planners must be chosen with care. Many of the standard benchmark domains were designed with specific agendas. Often they were designed to showcase specific expressive features of particular languages and are uninteresting when expressed in STRIPS (the Movie domain is one example and Pednault's Briefcase World [27] another). Others are designed to showcase particular planning strategies (the Rocket Domain used for GRAPHPLAN, for example [2]), or to demonstrate flaws in certain planning strategies (for example, the Gripper Domain). Although these domains retain some interest for these very reasons, it is important for the planning community to look beyond these "simple" problems and identify more significant benchmarks which represent tasks that demonstrate planning's coming-of-age to the wider research and applications community. The greatest challenge for the community, then, is to take the lessons learned from the competition and from the research that is current and to show how planning can move on from these problem domains.

References

[1] C. Anderson and D. Weld. Conditional effects in Graphplan. In *AIPS-98*, pages 44–53, 1998.

[2] A. Blum and M. Furst. Fast planning through planning graph analysis. *AIJ*, 90(1–2):279–298, 1997.

[3] B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of AAAI-97*, 1997.

[4] M.D. Ernst, T.D. Millstein, and D.S. Weld. Automatic sat-compilation of planning problems. In *Proceedings of IJCAI-97, Nagoya, Japan*, 1997.

[5] E. Fink and M. Veloso. Formalizing the PRODIGY planning algorithm. In M. Ghallab and A. Milani, editors, *New Directions in AI Planning*, pages 261–272. IOS Press (Amsterdam), 1996.

[6] M. Fox and D. Long. The automatic inference of state invariants in TIM. *JAIR*, 9, 1998.

[7] M. Fox and D. Long. Exploiting symmetry in STAN. In *Proceedings of IJCAI*, 1999.

[8] B. Gazen and C. Knoblock. Combining the expressivity of UCPOP with the efficiency of Graphplan. In *ECP-97*, pages 221–233, 1997.

[9] H. Geffner and B. Bonet. High-level plannnig and control with incomplete information using POMDPs. In *Proceedings AIPS-98 Workshop on Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*, 1998.

[10] C.P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proceedings of AAAI-98, Madison, WI*, 1998.

[11] W. Harvey and M. Ginsberg. Limited discrepancy search. In *Proceedings IJCAI-95*, 1995.

[12] Joerg Hoffmann and Jana Koehler. A new method to index and query sets. In *IJCAI-99*, 1999.

[13] R. J. Bayardo Jr. and R. C. Schrag. Using CSP look-back techniques to solve real world SAT instances. In *Proceedings of AAAI-97*, 1997.

[14] R. Kambhampati, E. Lambrecht, and E. Parker. Understanding and extending Graphplan. In *Proceedings of the 4th European Conference on Planning (ECP-97)*, pages 260–272. Berlin, Germany: Springer-Verlag, 1997.

[15] H. Kautz and B. Selman. Planning as satisfiability. In *Proceedings of ECAI-92, Vienna, Austria*, 1992.

[16] H. Kautz and B. Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proceedings of AAAI-96, Portland, OR*, 1996.

[17] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In *ECP-97*, pages 273–285, 1997.

[18] R. Korf. Real-time heuristic search. *Artificial Intelligence*, 42, 1990.

[19] R. Korf. Linear-space best-first search. *Artificial Intelligence*, 62, 1993.

[20] C.M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of IJCAI-97, Nagoya, Japan*, 1997.

[21] D. Long and M. Fox. The efficient implementation of the plan-graph in STAN. *JAIR*, 1999.

[22] D. McDermott. A heuristic estimator for means-ends analysis in planning. In *Proceedings of Third Int. Conf. on AI Planning Systems (AIPS-96)*, 1996.

[23] D. McDermott and the AIPS Planning Competition Committee. *PDDL – The Planning Domain Definition Language*. Draft, 1998.

[24] B. Nebel, Y. Dimopoulos, and J. Koehler. Ignoring irrelevant facts and operators in plan generation. In *ECP-97*, pages 338–350, 1997.

[25] N. Nilsson. *Principles of Artificial Intelligence*. Tioga, 1980.

[26] J. Pearl. *Heuristics*. Morgan Kaufmann, 1983.

[27] E. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 324–332. San Francisco, Calif: Morgan Kaufman, 1989.

[28] J. Penberthy and D.S. Weld. UCPOP: A sound and complete partial order planner for ADL. In *Proceedings of the third International COnference on Principle of Knowledge Representation and Reasoning (KR-92)*, 1992.

[29] B. Selman, H. Kautz, and B. Cohen. Noise strategies for local search. In *Proceedings of AAAI-94, Seattle, WA,*, pages 337–343, 1994.

[30] D. Smith and D. Weld. Incremental Graphplan. Technical Report TR 98-09-06, University of Washington, 1998.

[31] D. Smith and D. Weld. Temporal Graphplan with mutual exclusion reasoning. Technical Report To appear in IJCAI-99, University of Washington, 1998.

[32] D.E. Smith and D.S. Weld. Conformant Graphplan. In *Proceedings of 15th National Conference on Artificial Intelligence*, pages 889–896. Menlo Park, Calif: AAAI Press, 1998.

[33] D.S. Weld, C.R. Anderson, and D.E. Smith. Extending Graphplan to handle uncertainty and sensing actions. In *Proceedings of 15th National Conference on Artificial Intelligence*, pages 897–904. Menlo Park, Calif: AAAI Press, 1998.