Quality of Solutions to IPC5 Benchmark Problems: Preliminary Results

Patrik Haslum
NICTA & Australian National University
patrik.haslum@nicta.com.au

Abstract

In the 5th International Planning Competition, one objective was to emphasise plan quality in the evaluation of participating planners. However, as most of the competition problems were not solved by any optimal planner, the question of how good the plans found by competing planners are, in absolute terms, remains unanswered. This paper presents the results of efforts to find optimal solutions, or improved lower and upper bounds on the quality of solutions, to problems in the IPC5 domains, by a variety of methods.

Introduction

Although plan quality has always been a consideration in research on automated planning, and consequently in the evaluation of planners participating in the International Planning Competition (IPC) series, it has often taken a secondary role, with time required to produce a plan viewed as the most important criterion. One of the objectives of the 5th International Planning Competition (IPC5), held in 2006, was to place a greater emphasis on plan quality. The version of PDDL used in IPC5, known as PDDL3, extended previous versions with several constructs aimed at allowing more flexibility in specifying constraints and "desiderata" on plans (Gerevini & Long 2006).

In every competition since the beginning of the IPC series, there have been some participating planners offering a guarantee of optimality, *w.r.t.* some measure of plan quality. Mostly, however, optimal planners' ability to solve problems of increasing size within the time limit set in the competition has not been able to stay on par with that of suboptimal planners. This is understandable, as optimal planners solve a much harder problem, but unfortunate, since it means that for the majority of problem instances, optimal plan quality is not known. Although suboptimal planners can be, and indeed normally are, compared with respect to the quality of the plans they generate, this is not a fully satisfactory evalution, as it does not answer one important question: How good are the plans in absolute terms? Put in a different way, if two suboptimal planners find plans of similar quality, does it mean they are equally good or equally bad?

Leaving this question unanswered for the results of IPC5 is particularly unsatisfying, due to its stated objective of raising the importance of plan quality. This paper presents the results of efforts to obtain improved bounds on the quality of solutions, and in some cases optimal solutions, to problems in the IPC5 domains, by a variety of methods, including some domain-specific algorithms, and compares the quality of solutions obtained with that of solutions found by planners participating in the competition. Results are preliminary, in the sense that only a few of the IPC5 domains are considered and in some domains results are not complete. Particular emphasis is placed on domains that make use of the new constructs in PDDL3 to specify the quality metric.

The next section describes the domains considered, the methods used to find solutions and quality bounds in these domains, and the results obtained. The section after presents the comparison with the results of the competition. Conclusions are reserved for the last section.

Domains

Ever since the 3rd planning competition it has been customary to present in competitions several "versions" of each competition domain, each making use of a different subset of language features, a tradition followed also in IPC5. However, in most cases these different "versions" encode radically different problems, and thus should properly be considered as different domains (sharing only a common "theme"). This is particularly true of the IPC5 domains considered here. Moreover, the choice of PDDL constructs to use in specifying a domain is often somewhat arbitrary, since the same problem can often be equivalently expressed in several ways. From this perspective, the IPC tradition of distinguishing different "tracks" based on language features is unnecessarily limiting, which is also reflected in the competition results.

A general property of the domains considered is that they all encode optimisation problems. It is significantly easier (in some cases trivial) to find a solution that only satisfies the "hard constraints" of a problem instance: the true difficulty lies in finding a solution that also has a high quality. It should also be noted that although some domains are "inspired" by applications, none of them model actual application problems. In this respect, the competition problem instances are "artificial", and moreover they are specifically designed to offer challenging optimisation problems.

product sequence:	1	2	3	5	4
order 1 ($\{1,2\}$):	X	X			
order 2 ($\{1,3\}$):	X	–	X		
order 3 ($\{2,4\}$):		X	–	–	X
order 4 ($\{3,5\}$):			X	X	
order 5 ($\{4,5\}$):				X	X
# of open stacks:	2	3	3	3	2

Figure 1: Illustration of how the number of open stacks is calculated from a a product sequence. An "X" denotes that the order includes a request for the corresponding product; a "–" that the order is open at a point in the sequence, even though it does not include a request for the product made at that point.

Openstacks

The Openstacks (Propositional) domain is based on the "minimum maximum open stacks" combinatorial optimisation problem, which can be stated as follows:

A manufacturer has a number of orders, each for a combination of different products. Only one product can be made at a time, but the total required quantity of that product is made at that time. From the time that the first product requested by an order is made to the time that all products included in the order have been made, the order is said to be "open" and during this time it requires a "stack" (a temporary storage space). The problem is to order the making of the different products so that the maximum number of stacks that are in use simultaneously, *i.e.*, the number of orders that are in simultaneous production, is minimised.

Figure 1 illustrates the relationship between orders, the product sequence, and the number of open stacks in a small example problem.

This and several related problems have been studied in operations research (see, e.g., Fink & Voss, 1999). It is a pure optimisation problem: for any instance of the problem, every ordering of the making of products is a solution, which at worst uses as many simultaneously open stacks as there are orders. The problem is known to be equivalent to several other problems, including an NP-hard problem (Linhares & Yanasse 2002). Recently, it was posed as a challenge problem for the constraint programming community, and as a result, a large library of problem instances, as well as data on the performance of a number of different solution approaches, is available (see Smith & Gent, 2005).

The Openstacks planning domain is a direct encoding of the openstacks problem. There are two different formulations of the domain. In the "plain" version of the domain, the encoding is done in such a way that the length of a plan equals the maximum number of open stacks plus a problem-specific constant (equal to twice the number of orders plus the number of products). Thus, minimising the number of actions in the plan also minimises the objective function, *i.e.*, the maximum number of open stacks. Because no plan quality metric can be specified in the propositional (STRIPS/ADL) fragment of PDDL, a different formulation had to be used in the competition: in this, the "sequenced" version, additional constraints ensure that no two actions can be executed in parallel, so that minimising the number of "parallel steps" is equivalent to minimising the number of actions. The constant offset between the number of steps and the maximum number of open stacks is larger in the sequenced domain (equal to twice the number of orders plus twice the number of products). The problem instances used in the competition are a selection of instances from the constraint modelling challenge problem library, plus five extra instances of trivially small size (`p01`–`p05`). Table 1 summarises some facts about the competition problems.

	(a)	(b)	(c)	(d)	(e)
`p01`		5	3**	15	20
`p02`		5	3**	15	20
`p03`		5	3**	15	20
`p04`		5	3**	15	20
`p05`		5	3**	15	20
`p06`	wbop_10_10 #11	10	5**	30	40
`p07`	wbop_10_10 #18	10	6**	30	40
`p08`	nwrsSmaller #3	15	7**	55	80
`p09`	nwrssmaller #4	15	7**	55	80
`p10`	wbop_20_20 #12	20	9*	60	80
`p11`	wbop_20_20 #14	20	9*	60	80
`p12`	wbop_20_20 #21	20	12*	60	80
`p13`	wbop_20_20 #35	20	16*	60	80
`p14`	wbop_20_20 #37	20	15**	60	80
`p15`	Shaw #4	20	13**	60	80
`p16`	Shaw #24	20	14**	60	80
`p17`	nwrsLarger #1	20	12*	70	100
`p18`	nwrsLarger #3	25	10*	109	168
`p19`	sp4 #1	25	9*	75	100
`p20`	wbop_30_30 #17	30	10*	90	120
`p21`	wbop_30_30 #20	30	9*	90	120
`p22`	wbop_30_30 #26	30	15*	90	120
`p23`	wbop_30_30 #37	30	20*	90	120
`p24`	gp50by50 #2	50	40*	150	200
`p25`	gp50by50 #4	50	30*	150	200
`p26`	sp4 #2	50	19*	150	200
`p27`	sp4 #3	75	34	225	300
`p28`	gp100by100 #2	100	75*	300	400
`p29`	gp100by100 #4	100	60*	300	400
`p30`	sp4 #4	100	54	300	400

Table 1: Facts about the Openstacks problem instances: (a) corresponding 2005 Constraint Modelling Challenge problem; (b) number of orders (also an upper bound on the number of open stacks); (c) number of open stacks in best known solution; one star indicates the number is optimal, two that the problem (in plain formulation) was solved by an optimal planner; (d) the constant offset between plan length and number of open stacks, for the plain formulation; (e) ditto, sequenced formulation.

In the 2005 Constraint Modelling Challenge, the dynamic programming algorithm by Garcia de la Banda & Stuckey (2005) stood clearly out from the rest in terms of performance. It solved all problems in the challenge library except two: "sp4 #3" and "sp4 #4". Naturally, both of these were

included among the problems used in the planning competition. For one of these problems, #3, SGPlan found a solution using one stack less than the best previously known solution. However, the optimal number of stacks remains unknown.

Solutions to problems in the Openstacks domain were obtained with a re-implementation of Garcia de la Bandas & Stuckeys algorithm. However, the MIPS-BDD planner, configured to ensure optimality *w.r.t.* number of actions in the plan, was also able to solve a decent number of problems using the plain domain version.

Openstacks SimplePreferences

The Openstacks SimplePreferences (SP) domain models a problem similar to, yet radically different from, the original openstacks problem. The main ingredients are the same: a set of products to be made in sequence, a set of orders, each for some subset of products, and the constraint that an order is "open", and requires a "stack", from the point where the first product requested by the order is made to the point where the last such product is made. The difference lies in the objective function: in this problem, the number of stacks that may be used is fixed to a constant in each problem instance, and the constraint that all requested products must be included in each order is "soft", *i.e.*, it does not have to be satisfied for a plan to be valid, but the plan is given a penalty for each violation. The objective is to minimise the total penalty for unsatisfied product requests.

The encoding of this problem makes use of *preferences*, one of the new features in PDDL3. Briefly, (simple) preferences allow a plan metric to be defined in terms of the truth or falsity of atoms in the state at the end of plan execution. Preferences are used to specify the penalties for not delivering requested products. The same objective could also have been expressed using numeric state variables.

Two different models for the penalty associated with unsatisfied product requests were used, each in roughly half the instances. In the "uniform" model, the penalty is 1 for every request in every order, so the objective is simply to minimise the number of unsatisfied requests. In the other, the "exponential", model, products requested by each order were given an (arbitrarily chosen) "order of importance": the value of satisfying only the request for the most important product is 1, and each following request satisfied adds twice the value of the previous. For example, if an order requests the set $\{1, 6, 9\}$, the value of delivering $\{1\}$ is 1, the value of delivering $\{1, 6\}$ is 3, and the value of delivering the full set is 7. Note that the penalty for failure to satisfy product requests follows an opposite pattern: in the example, not delivering product 1 implies a total penalty (value lost) of 7, while failure to deliver only product 9 gives a penalty of 4.

The intent when creating problem instances for this domain was to make the number of stacks in each instance slightly smaller than the minimum required to accommodate all product requests, thus forcing a solution to choose a subset of the requests in each order to satisfy. However, because optimal solutions to all the original openstacks problems were not known when instances were constructed, some instances ended up having a sufficient number of stacks to permit solutions with zero penalty. This unintentional flaw led to some interesting results.

Table 2 summarises some facts about the competition problems.

The Min-Penalty-Fixed-Sequence Problem Openstacks SP is a difficult optimisation problem, because it has many degrees of freedom: a solution must choose both an ordering of the products and which product requests to leave unsatisfied, with the fixed maximal number of open stacks as the only constraint. Fixing one of these choices, *i.e.*, either fixing the sequence of products or the requests to drop, makes attacking the remaining problem easier, and the quality of a solution to any such "partially fixed" problem is an upper bound on the obtainable quality. The chosen approach was to enumerate product sequences and minimise penalty for each fixed sequence.

The "minimum penalty for a fixed product sequence" (MPFS) problem is solved by a branch-and-bound algorithm. To reduce by one the number of open stacks at any point in the sequence, it is necessary to drop from one of the orders open at that point either all requests for products made earlier in the sequence ("to the left") or all requests for products made later in the sequence ("to the right"), as well as for the product made at the point, if the order includes it (see Figure 1). Each such "fix" carries a penalty, and the penalty for reducing the number of open stacks at a point by some $k > 1$ is no less than the sum of the k such "fixes" that have the smallest individual penalties. The algorithm selects a point in the sequence where the number of open stacks exceeds the limit and branches on the fix to apply. The lower bound is fairly weak, as it considers only "the cheapest fix to the most costly problem". Stronger bounds are possible: if, for example, the sets of orders open at two points where the number of open stacks exceeds the limit are disjoint, the sum of the lower bounds on the penalties to fix both points is also a lower bound on the total penalty of a solution.

The algorithm is reasonably efficient when the difference between the number of stacks available and number used by the input product sequence is small, but degrades as this difference grows.

Lower Bounds Since the MPFS algorithm does not prove optimality of the solution, other methods are needed to find lower bounds on the minimum penalty attainable. With the exception of instances `p15`–`p18`, the stack limit in the Openstacks SP problems is less than the number required to satisfy all product requests, so a simple lower bound is the smallest penalty for dropping any single request. In surprisingly many instances, this bound is met by the MPFS solution.

Several lower bounds on the minimum number of stacks can be obtained from the *co-demand graph* (Smith & Gent, 2005, give a summary). This graph has a node for each order and an edge between two nodes iff the corresponding orders have a product in common. The minimum degree of any node in the co-demand graph plus one and the size of any clique in the graph are both lower bounds on the number of stacks needed. To lower either bound edges must be removed from the graph, which corresponds to removing each of the shared product requests from one of the two orders.

	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)	(k)	
p01	wbop_10_10 #11	E	70	4 (5)	14.0	4 / 4 / –	6	38.0	70.0	70.0	59.0	(2,3)
p02	wbop_10_10 #18	E	70	5 (6)	11.6	4 / 4 / –	4*	35.6	60.6	69.6	52.8	(3)
p03	nwrsSmaller #3	U	90	6 (7)	12.8	1 / 1 / 1	2	24.8	84.8	89.6	70.0	(5)
p04	nwrsSmaller #4	U	100	6 (7)	14.2	1 / 1 / 2	4	29.2	96.2	99.4	79.8	(4)
p05	wbop_20_20 #12	E	140	8 (9)	15.5	4 / 0 / –	4*	39.5	120.5	139.5	98.5	(5)
p06	wbop_20_20 #14	E	140	8 (9)	15.5	4 / 0 / –	4*	39.5	120.5	139.5	94.0	(4)
p07	wbop_20_20 #21	E	300	11 (12)	25.0	8 / 8 / –	8*	110.0	265.0	300.0	248.0	(8)
p08	wbop_20_20 #35	E	620	15 (16)	38.7	16 / 16 / –	24	364.7	565.7	619.2	593.1	(13)
p09	wbop_20_20 #37	E	620	14 (15)	41.3	16 / 16 / –	16*	347.3	568.3	619.5	554.3	(11)
p10	Shaw #24	U	120	13 (14)	8.5	1 / 0 / 0	1*	24.5	114.2	119.0	90.5	(9)
p11	Shaw #4	U	120	12 (13)	9.2	1 / 0 / 0	1*	27.2	115.5	119.6	88.4	(7)
p12	nwrsLarger #1	U	153	11 (12)	12.75	1 / 0 / 0	1*	32.75	151.75	153.0	122.0	(8)
p13	nwrsLarger #3	U	223	9 (10)	22.3	1 / 1 / 1	3	43.3	216.3	223.0	185.6	(2)
p14	sp4 #1	U	65	7 (9)	7.2	1 / 0 / 1	3	27.6	55.2	64.8	42.8	(4)
p15	wbop_30_30 #17	E	210	11 (10)	17.5		0*	40.5	164.5	175.0	138.0	(6)
p16	wbop_30_30 #20	E	210	11 (9)	17.5		0*	40.5	171.5	157.5	120.5	(5)
p17	wbop_30_30 #26	E	450	17 (15)	25.0		0*	131.0	400.0	375.0	331.0	(11)
p18	wbop_30_30 #37	E	930	21 (20)	42.2		0*	390.2	848.2	844.0	650.8	(14)
p19	gp50by50 #2	U	1581	37 (40)	39.5	1 / 1 / 3	96	84.5	1580.5	1580.0	1524.0	(6)
p20	gp50by50 #4	U	1348	27 (30)	44.9	1 / 3 / 3	127	86.9	1347.9	1347.0	1336.3	(7)

Table 2: Facts about the Openstacks SP and QP problem instances: (a) corresponding 2005 Constraint Modelling Challenge problem; (b) penalty model ("Uniform" or "Exponential"); (c) maximum total penalty for unsatisfied product requests; (d) fixed number of stacks in the SP domain; (in parentheses: minimum number of stacks needed to accommodate all product requests); (e) penalty/stack in the QP domain.
Openstacks SP domain: (f) lower bounds: smallest single-product penalty / by co-demand graph min degree / by co-demand graph max cliques (problems with uniform penalty model only); (g) best MPFS solution; a star indicates the solution is optimal (matched by one of the lower bounds). In problems p15–p18 the stack limit is large enough to accommodate all product requests, making a penalty of zero possible.
Openstacks QP domain: (h) lower bound; (i) upper bound by single-stack construction (using approximate weighted independent set); (j) upper bound by optimal solution to original openstacks problem; (k) best known solution (in parenthesis: number of stacks used by this solution). For instances p13, p14, p19 and p20, the best solution is one submitted by a planner in the competition. In the remaining instances, it was found by trying the MPFS solver with different fixed numbers of stacks.

Thus, each edge can be assigned a weight, equal to the minimum penalty incurred if it is removed, and lower bounds on the minimum penalty overall obtained by finding (or lower bounding) the least weight set of edges to remove that brings both bounds on the number of stacks down to the limit.

MPFS solutions and lower bounds for the competition problems are summarised in Table 2, columns (f) and (g). The lower bounds on penalty are fairly weak, in particular for problems with uniform penalties.

Openstacks QualitativePreferences

The Openstacks QualitativePreferences (QP) domain combines the two objective functions of the openstacks and Openstacks SP problems, by a weighted sum. That is, a solution may use any number of stacks and may drop any set of product requests, but must minimise the sum of a price per stack used and the total penalty for unsatisfied requests. The encoding of the problem makes some use of PDDL3 plan constraints, to keep track of the number of stacks used in a plan, but that is not an essential feature of the domain: this part of the objective function could equally well have been expressed using only simple (goal state) preferences, or numeric state variables (in fact, the latter is done in the Openstacks MetricTime domain).

The competition problems for this domain are based on the same instances as in the Openstacks SP domain and use the same two penalty models (uniform and exponential). The price per stack in each instance was set to the total penalty for unsatisfied product requests divided by the optimal (or, in the case of problems p15–p18, best known at the time) number of stacks required to accommodate all requests, with the aim of making the two extreme solutions roughly equal in value so that, hopefully, the best solution would be a trade-off somewhere in between. The price per stack for each problem is listed in column (e) in Table 2.

Upper and Lower Bounds The Openstacks QP domain has essentially no hard constraints, which, while making it hard to optimise, makes it easy to construct feasible solutions, and therefore to derive upper bounds. One is given by the cost of a solution that delivers all requested products using an optimal number of stacks. Another set of solutions are given by the best solution to the Openstacks SP problem induced by fixing the number of stacks at any value less than the optimal. For large instances, however, this method can only find solutions that use a relatively large number of

stacks, because the efficiency of the MPFS algorithm degrades as the discrepancy between the stack limit and the optimal number of stacks for the original problem increases.

At the other end of the spectrum, a solution using a single stack can be constructed by selecting a set of orders that form an independent set in the co-demand graph. As no two orders in this set request the same product, they can be accommodated on a single stack, if all requests of all orders not in the chosen set are dropped. Because the value (*i.e.* penalty avoided) of satisfying an order may vary between orders, this amounts to solving a weighted independent set problem. This is an NP-hard problem, but good solutions can be found by approximation algorithms (Halldórsson 2000; Kako *et al.* 2005).

A lower bound can be obtained by the methods described in the last section, minimising over all numbers of stacks less than the minimum required to satisfy all product requests. This bound, however, inherits the weakness of the lower bounds on Openstacks SP problems. Lower and upper bounds obtained for the competition problems are listed in Table 2, columns (h)–(k).

Openstacks Time and MetricTime

The Openstacks Time and MetricTime domains again have the same elements as the original openstacks problem but very different objective functions. In the Openstacks Time domain, the objective is to minimise plan makespan. Making each product takes a different amount of time, but any number of products can be made in parallel as long as all orders requesting the products are simultaneously open. In the Openstacks Time domain the maximal number of stacks in use is fixed, while in the MetricTime domain it is unlimited and the objective function is a weighted combination of makespan and the number of stacks used. There are no "soft" goals: satisfaction of all product requests is mandatory.

The problem was encoded in the "timed" fragment of PDDL. Numeric state variables were used to keep track of the number of stacks used in the MetricTime domain, but, again, this part of the objective function could have been formulated differently (*e.g.*, as done in the Openstacks QP domain). Problem instances for the competition were based on the same set of openstacks problems as in the Openstacks SP and QP domains. The time to make each product was set randomly, in the range 1 to 10. The encoding of the problem has a fairly large number of "mandatory" actions, meaning any valid plan must contain them; to ensure that the scheduling of the product-making actions dominates plan makespan, their durations were scaled up by a factor equal to the number of orders in the problem, while all other actions were given a duration of 1. The fixed number of stacks available in instances of the Openstacks Time domain was set to a value close to the upper bound (number of orders). For the MetricTime domain, the "price per stack" was determined by comparing the makespan of the best plans found with different fixed numbers of stacks, and choosing a value equal to the average decrease in makespan per stack added, following again the principle of making the extreme points on the spectrum of trade-offs roughly equal in value.

The Openstacks Time domain is too hard for any current makespan-optimal planner to solve. The solutions used in the construction of the MetricTime problems were obtained with the LPG planner (Gerevini, Saetti, & Serina 2006), by allowing it to run for a long time on each problem. In many cases, though not all, these solutions are still the best known. Lower bounds on makespan were obtained with the temporal h^2 (admissible) heuristic. For problems in the MetricTime domain, the combination of the known minimum number of stacks required and the h^2 estimate of makespan (which, due to the nature of the heuristic, is the same for any fixed number of stacks greater than or equal to two) yields a lower bound.

Facts about the problem instances, best known solutions and lower bounds are summarised in Table 3.

Rovers MetricSimplePreferences

The Rovers MetricSimplePreferences (MSP) domain is nominally based on the Rovers domain, introduced in IPC3 (Long & Fox 2003), but more importantly it is an instance of the class known as *net benefit maximisation* problems. These are "over-subscribed problems", in which the task of the planner is not to plan for all the given goals, but to select and plan for a subset of goals in a way that maximises some measure of "return" within given constraints. Problems of this kind have been studied in scheduling and have also attracted interest among planning researchers recently (*e.g.* Smith 2004; Do *et al.* 2007).

The net benefit of a plan is defined as the value of the goals achieved by the plan minus the cost of the plan. There are no "hard goals", *i.e.* goals that must be achieved. The cost of a plan is the sum of the (independent and constant) costs of actions in it, and the value of the set of goals achieved is given by associating to each of a set of atomic potential goals a constant value. The objective is to maximise net benefit. Problems of this kind are straightforward to encode in PDDL3, using preferences to assign values to the goals and one or more numeric state variables to keep track of plan cost.

Problem Instance Construction Instances of the Rovers MSP domain for IPC5 were created by a method aimed at generating "interesting" problems, having balanced costs and values for each subset of goals and thus non-obvious optimal solutions. It is a general method, applicable to any domain.

First, (random) base problem instances with action costs assigned and a relatively large number of potential goals are generated. Second, the real cost of achieving small sets of goals (single goals and pairs of goals) is found, by optimally solving the corresponding planning problems. Third, goals (and, in some cases, conjunctions of two goals) are assigned "base values", using the costs to estimate the kind and strength of "interaction" between goals, in a way intended to make the achievable net benefit of all goal sets roughly equal. Final goal values are then determined by randomly adding or subtracting a percentage from the base value (for the competition problems, up to $\pm 99\%$). Note that, as there are no hard goals, the plans generated as part of the construc-

	(a)	(b)	(c)	(d)	(e)	(f)	(g)
p01	wbop_10_10 #11	8 (5)	41.0	105	188	310.0	477.0
p02	wbop_10_10 #18	9 (6)	32.0	105	138	297.0	426.0
p03	nwrsSmaller #3	13 (7)	55.5	155	275	543.5	873.0
p04	nwrsSmaller #4	13 (7)	66.5	155	297	620.5	1054.0
p05	wbop_20_20 #12	18 (9)	52.0	205	311	673.0	1247.0
p06	wbop_20_20 #14	18 (9)	27.0	205	261	448.0	747.0
p07	wbop_20_20 #21	18 (12)	59.0	205	330	913.0	1392.0
p08	wbop_20_20 #35	19 (16)	100.0	205	312	1805.0	2212.0
p09	wbop_20_20 #37	18 (15)	54.5	205	322	1022.5	1303.0
p10	Shaw #4	18 (13)	71.5	205	353	1134.5	1640.0
p11	Shaw #24	18 (14)	73.0	205	359	1227.0	1673.0
p12	nwrsLarger #1	18 (12)	88.0	205	403	1261.0	1849.0
p13	nwrsLarger #3	18 (10)	35.3	255	535	610.0	1170.4
p14	sp4 #1	23 (9)	40.5	255	345	619.5	1276.5
p15	wbop_30_30 #17	28 (10)	23.5	305	378	540.0	1036.0
p16	wbop_30_30 #20	28 (9)	34.5	305	390	615.5	1356.0
p17	wbop_30_30 #26	28 (15)	78.0	305	495	1475.0	2679.0
p18	wbop_30_30 #37	28 (20)	100.0	305	532	2305.0	3093.0
p19	gp50by50 #2	48 (40)	120.5	505	833	5325.0	6618.0
p20	gp50by50 #4	48 (30)	144.5	505	775	4840.0	7816.0

Table 3: Facts about Openstacks Time and MetricTime instances: (a) corresponding 2005 Constraint Modelling Challenge problem; (b) fixed number of stacks in the Time domain; (in parentheses: minimum number of stacks required); (c) price/stack in the MetricTime domain.
Openstacks Time domain: (d) lower bound on makespan; (e) makespan of best known solution.
Openstacks MetricTime domain: (f) lower bound on metric value; (g) metric value of best known solution.

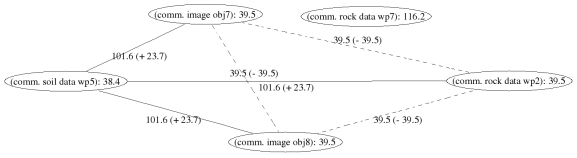

Figure 2: Single-goal and pair-of-goals optimal costs in a small Rovers problem.

tion are all valid plans for the final problem, and so yield a lower bound on the net benefit attainable.

The interaction between two goal atoms p and q is labelled a *synergy* if the cost of achieving $\{p, q\}$ is less than the cost of achieving p plus the cost of achieving q. The synergy effect is the difference between the two. Conversely, if the cost of achieving the pair is greater than the sum of the single-goal costs, the interaction is labelled an *interference*. Figure 2 illustrates the relationships between goal atoms in a small problem instance. The base value of a single-atom goal that has no interactions equals the optimal cost of achieving the goal. The base value of a goal that has only synergy relations to other goals is reduced by half the average synergy effect, while for goals with only interference relations it is increased by the corresponding amount. Goals that are in both synergy and interference relations have their base values reduced in the same way as goals with only synergy relations, but as compensation the conjunction of any pair of such goals that are in an interference relation is given an additional value, equal to the interference effect.

For the Rovers MSP domain, base problems were generated with a slightly modified version of the random problem generator used in IPC3 (STRIPS version). Only the `navigate` actions were given a non-zero cost, and only instances of `comm_rock_data`, `comm_soil_data` and `comm_image_data` were selected as goals. (Some goals turned out to be achievable by zero cost plans, and therefore got a base value of zero; as this was only discovered at a late date, the problem was "fixed" by assigning such goals a small value, 1%–10% of the total goal value.) In general, the decision problem underlying this domain is NP-hard (this can be shown by a reduction from Hamiltonian cycle, similar to that used for the Travelling Salesman problem). However, the roadmaps created by the IPC3 problem generator have a particularly simple form. In the competition, the net benefit maximisation objective was reformulated as a minimisation objective, and also shifted by a problem-dependent constant.

Three sets of problem instances were created: one comprising problems with only synergy relations between goals ("type I"), one with problems having only interference relations ("type II") and one with problems having a mixture of goal relations ("type III"). A "test run", using a simple optimal planner for net benefit problems, was made to filter out problems that were too easy (solved optimally) or that appeared too hard (no improvement over the lower bound from problem construction could be found).

Generating and Solving Decision Problems The planner used for the "test run" in problem construction is too inef-

	(a)	(b)	(c)	(d)
– Type I (synergistic goals) –				
p01	851.1	811.3*		4/ 5
p02	610.8	473.2*		5/ 6
p03	862.2	811.3*		5/ 6
p04	461.9	418.7*		5/ 5
p05	870.0	483.6*		6/ 6
p06	655.7	649.2*		5/ 8
p07	402.2	402.2*		2/ 5
– Type II (interfering goals) –				
p08	978.7	698.4*		6/ 9
p09	432.9	326.2*		9/ 9
p10	873.5	617.1*		6/ 9
p11	698.1	468.6*		7/ 8
p12	425.6	371.9	363.3	9/10
p13	1550.6	755.8	718.6	11/12
– Type III (mixed) –				
p14	578.4	442.2*		6/ 7
p15	3948.4	1004.5	940.6	21/22
p16	4672.3	944.7*		22/22
p17	1768.7	721.9*		18/18
p18	742.9	628.8*		8/10
p19	699.0	345.2*		6/ 6
p20	3183.1	1116.2	924.7	20/20

Table 4: Bounds on Rovers MSP problems: (a) upper bound from problem construction; (b) best known solution; a star indicates it is optimal; (c) lower bound on best possible solution; (d) fraction of soft goals achieved by best known solution. Note that values refer to the metric as specified in the competition domain: the objective is to *minimise* this value.

ficient to be able to provide optimal solutions to the competition instances (indeed, that was one of the criteria by which they were chosen). Instead, improved solutions and bounds were found by iteratively generating and solving decision problems seeking to improve on the quality of the best known plan.

Let $V(G')$ denote the value of a subset of goals G'. Given a bound B, a plan that achieves G' attains a net benefit greater than B iff the cost of the plan is less than $C = B - V(G')$. Thus, the question "does there exist a plan with net benefit $> B$?" can be answered by deciding the solvability of a collection of cost-bounded ordinary planning problems, one for each goal subset. As soon as a plan is found for any one of them, the process is repeated with the bound B on net benefit set to the value of that plan.

The number of goals in the competition problems range from 5 to 22. Enumerating all goal subsets is feasible, but solving all the corresponding decision problems is not: for this approach to be practical it is essential to have good bounds on the cost of reaching each set of goals. Lower bounds can be obtained using various admissible heuristics (for the Rovers domain, the additive h^2 heuristic by Haslum, Bonet & Geffner, 2005, is fairly useful, if a suitable action partitioning is provided). A decision problem with goals G' and cost bound C that has been searched and proven to be unsolvable also yields a lower bound of C on the cost of achieving G', or any superset of G'.

Using all available information, including plans submitted by the competitors, the number of decision problems that needed to be solved could be brought down to a few hundred for most instances. When the cost bound is relatively loose, it was often worth first trying to solve problems using one or more fast suboptimal planners, since any solution found improves the lower bound on net benefit and thus tightens the cost bound in the next iteration. For proving problems unsolvable, however, using an optimal planner was the only viable option. (Some experiments with a domain-specific solver, based on a constraint programming formulation of the problem, were also done.) All but four instances could be solved optimally by this method. Table 4 summarises the results. The results shown in the table are with respect to the plan metric as specified in the competition domain: the objective is to *minimise* the value.

Rovers QualitativePreferences

The Rovers QualitativePreferences (QP) domain is also based on the IPC3 Rovers domain but, again, it models a very different problem. This domain makes extensive use of the *plan constraints* introduced in PDDL3. In fact, it was designed explicitly to test the ability of competing planners to trade off "soft" plan constraints against each other.

PDDL3 plan constraints allow certain forms of "temporally extended goals", meaning conditions on intermediate states visited by the plan, to be expressed. In domains without metric time, the possible plan constraints are `(always` φ`)`, `(sometime` φ`)`, `(at-most-once` φ`)`, `(sometime-before` φ ψ`)` and `(sometime-after` φ ψ`)`, where φ and ψ are state formulas, and conjunctions of these. The modal operators have, for the most part, their intuitive meaning; definitions are given by Gerevini & Long (2006).

Plan constraints in the Rovers QP domain are all "soft", *i.e.*, a plan does not have to satisfy them, but is given a penalty for each unsatisfied constraint. The constraints are "artificial", in the sense that they do not encode any real preferences on plans. Problems also have regular "hard" goals to be achieved in the final state. Plan constraints may contradict each other, or the hard goals: an optimal solution in this domain is one that selects a jointly achievable set of constraints with maximum value (the "value" of a constraint being the penalty avoided by satisfying it).

Problem Instance Construction As in the case of the Rovers MSP domain, the method used to construct problem instances for the Rovers QP domain is general, and aims at producing problems with non-obvious optimal solutions.

For a given base problem, a set of candidate plan constraints is found by "mining" a set of plans for the problem. Testing if a plan satisfies a given constraint is easy, and for a sufficiently restricted class of state formulas it is feasible to determine the set of all possible constraints built with state formulas of the class that are satisfied by a plan. Given a set of plans, the candidate set of constraints are those that are satisfied by at least one plan, but not by all. This ensures that the chosen constraints are all satisfiable but not

trivially implied by the structure of the problem. (Due to bugs in the construction, some competition instances have a few constraints that conflict with the hard problem goals and therefore can not be satisfied.) Additional filtering criteria can also be applied.

The strategy for assigning penalties to chosen constraints is again to calculate a "base value", in a manner intended to make the values of all maximal satisfiable sets of constraints roughly equal, and determine final values by randomly adding or subtracting a percentage of the base value. Actually determining the maximal satisfiable sets of constrains is hard, so the calculation is an estimate, also based on the set of plans used to find the candidate constraints. Let N be the number of constraints in the chosen set, and let M_i be the maximum number of constraints in this set satisfied by any plan that satisfies constraint i: the base value (penalty) assigned to constraint i is $(N - M_i)/N$.

Note that the plans used in the construction are all valid plans for the final problem, and thus yield upper bounds on the minimum penalty attainable.

For the Rovers QP domain, base problems were the instances of the Rovers domain (STRIPS version) used in IPC3, and the plans used to find and rank candidate constraints were the plans submitted by planners participating in that competition. Constraint state formulas were restricted to single atoms. Because the number of "two formula" constraints (`sometime-before` and `sometime-after`) in the candidate set tend to be very large, only a randomly chosen subset of `sometime-before` constraints was included. The penalty for each constraint was chosen randomly within $\pm 50\%$ of the base value.

Plan Constraint Compilation The approach to finding bounds and improved solutions in the Rovers QP domain was again to enumerate and analyse decision problems corresponding to subsets of preferences. To analyse or solve these problems, the plan constraints were compiled away to yield ordinary STRIPS problems.

The approach of compiling away plan constraints was also used by several competing planners (*e.g.* Baier & McIlraith 2006; Edelkamp 2006). The compilation used here is direct (not via finite automata, as is otherwise common), produces plain STRIPS problems and preserves plan length. It exploits the fact that plan constraints in decision problems are hard (for example, a constraint `(always (p))` is compiled by removing all actions that delete `(p)`). Potentially, the compilation may increase the size of the problem (number of actions) exponentially, but this does not happen for the Rovers QP problems due to the restricted form of state formulas in plan constraints in these instances.

Lower bounds were obtained using the admissible h^3 heuristic. As this domain has no plan cost, what matters is only the heuristics ability to detect unreachability of sets of goals. Results are summarised in Table 5.

	(a)	(b)	(c)
p01	68.0393	79.3947	68.0393
p02	32.6666	38.1111	38.1111
p03	29.19	57.11	29.19
p04	23.8857	38	38
p05	65.1469	266	160.971
p06	28.608	68.901	68.901
p07	20.9077	116.196	87.9637
p08	484	856	620
p09	427.835	1558.74	884.375
p10	219.467	1484.63	473.361
p11	624.776	1559.42	996.759
p12	238.404	1282.68	250.716
p13	288.357	5309.38	2110.62
p14	246.787	929.05	513.937
p15	971.361	3108.81	2211.16
p16	468	3234	2406
p17	1162.69	5101.07	2553.72
p18	1240	8390	4971
p19	517.949	3896.32	2150.26
p20	567.899	30590	11282.5

Table 5: Bounds on Rovers QP problems: (a) lower bound; (b) upper bound from problem construction (*i.e.*, the value of the best IPC3 plan); (c) current best known solution. Except for problems `p01`, `p02`, `p04` and `p06`, current best solutions are plans submitted by competitors.

Comparison with Competition Results

Figure 3 shows comparisons between the quality of plans submitted by planners participating in IPC5 and the current best known solutions, lower and upper bounds. To avoid clutter in the graphs, individual planners are not identified. Graphs showing comparisons of the results of different planners can be found on the website of the IPC5 deterministic track.[1]

In the Openstacks domain, best quality plans are all by SGPlan, though shared with the optimal planners MIPS-BDD, FDP and MaxPlan, and with IPPLAN-G1SC, in some instances. In the Openstacks SP domain, best plans are by MIPS-BDD in two instances, and by SGPlan in the rest (the only other planner attempting this domain is MIPS-XXL, which in this domain finds plans of much worse quality). In the Openstacks QP domain, best plans are mostly by SGPlan, but in a few instances by MIPS-BDD or HPlan-P (MIPS-XXL also solves a few problems; in most instances, the quality of all planners' plans is quite close). In the Openstacks Time domain, best plans are all by Yochan^{PS} (MIPS-XXL and SGPlan in shared second place), while in the Openstacks MetricTime domain best plans are all by SGPlan (with MIPS-XXL again a very close second). In the Rovers MSP domain, best plans are mostly by MIPS-XXL and SGPlan, with a few by Yochan^{PS} (no other planners attempted this domain).

Even in the light of the information now available about the possible quality of solutions to the problems in these domains, it is hard to spot any general trend in the results of the competing planners. One observation is that, perhaps not so surprisingly, they mostly fare better in domains with a single quality criterion to optimise, curbed only by hard

[1] `http://ipc06.icaps-conference.org`

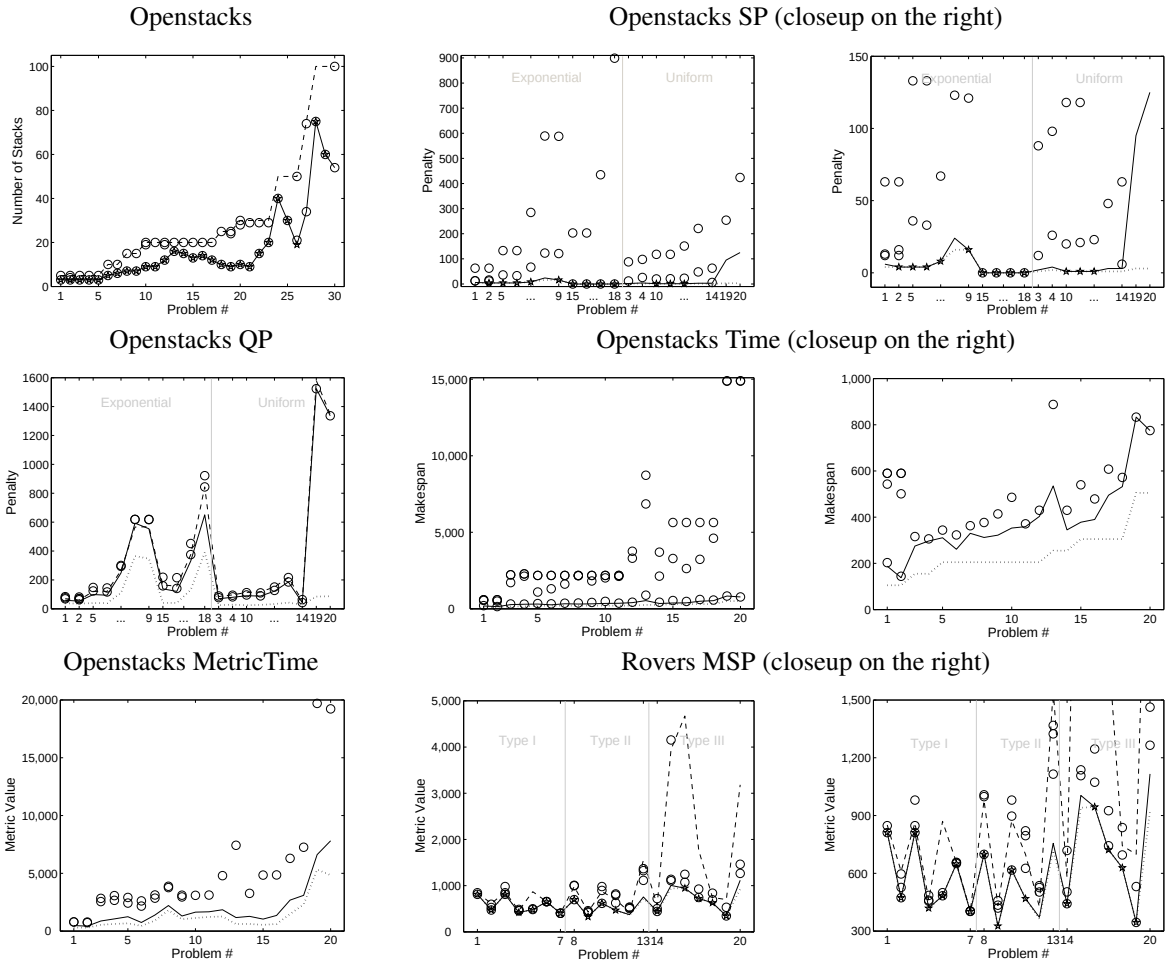

Figure 3: Comparison of the quality of plans found by competitors with current best known solutions and bounds. Note that all domains are minimisation problems. The solid line (—) displays the best known solution (a star on the line indicates that the solution is optimal), the dashed line (– –) the lowest upper bound, and the dotted line ($\cdots$) the highest lower bound (both bounds are not present in all graphs). Note that "upper bound" refers to "low complexity" bounds (*e.g.*, bounds obtained as a side effect of problem construction) and does not include all known solutions. Plans submitted by competing planners are indicated by circles.

constraints, than in domains where the objective function encodes a trade off different quality critera (such as number of stacks *vs.* satisfied requests or makespan, plan cost *vs.* value of goals achieved). Such problems are hard. Note that the solution methods developed in this paper for the Openstacks QP, Openstacks MetricTime and Rovers MSP domains are all, in one way or another, based on fixing one criterion while optimising the other.

Nevertheless, in the Rovers MSP domain, all the three planners that submitted plans occasionally found plans of optimal quality. Interestingly, on the instances where they do not, SGPlan and Yochan^{PS} both in most cases pick the same or a very similar set of soft goals to achieve as the optimal (or best known) plan, so it appears that when the quality of their plans in this domain falls short this is due to a failure to optimise plan cost. Also noteworthy is that the quality of the plans found by all three planners is generally further from optimal on problem instances of types II and III (with interfering goals) than instances of type I (with only synergistic goals). No such trend is perceivable in the planner runtimes.

Concerning individual planners, it is easiest to say something about SGPlan, due to the volume of data available: it submitted a plan for every problem in all of the domains studied. A fact that is clear is that its ability to optimise plan quality is far from even over different criteria. The quality of plans found by SGPlan in the Openstacks domain is quite remarkable: in all instances except one it equals the best known solution, and in one instance, the best known solution is the one found by SGPlan. In the Openstacks SP and Time domains, on the other hand, the plans it finds are not particularly good (with the exception of problems `p15`–`p18` in the Openstacks SP domain, where it is able to exploit the fact that all soft goals can be met). Its tendency to find solu-

tions using a small number of stacks is helpful in the Openstacks QP domain, where the stacks appear to be somewhat "overpriced", but not in the Openstacks MetricTime domain, where the gain in metric value of using no more than the optimal number of stacks is not enough to compensate for an inflated makespan.

Conclusions

The obvious use of the results presented in this paper is that they allow for a better evaluation of the results produced by planners competing in the 5th planning competition, by placing the quality of their plans in the context of upper and lower bounds on the best plan quality attainable.

Unfortunately, data on the performance of most of the competing planners across different domains is scarce, as many planners did not attempt to solve more than a few of the domains. A large part of the blame for this must be laid on the decision to base competition tracks on the PDDL fragment used and have each domain represented only in one track. For example, the real difference between the five domains based on the openstacks problem lies in the objective to optimise: number of stacks, product requests met and makespan, and weighted combinations of these. The assignment of each of these problems to only one "language category" is very arbitrary, since all of them could be formulated in ways fit for at least two or three such categories. Doing that, and allowing competing planners the choice of which formulation to solve, would almost certainly have resulted in better data.

The detailed study of the competition problems also reveals some properties of the problem instances, some of which can perhaps be considered "flaws". For example, current results suggest that stacks are overpriced relative to the cost of dropping product requests in the Openstacks QP domain and that there are synergy effects in large goal sets not accounted for by the method of problem construction in the Rovers MSP domain.

A question not addressed in this work is the generality of the results. The competition domains are "artificial", in the sense that they do not model real application problems, and in most cases problem instances were constructed with the explicit aim of posing challenging optimisation problems. As this analysis has also shown, instances in some domains have peculiarities that induce a bias towards certain kinds of plans. It can not be ruled out that the picture would be very different over a different problem set, in particular over a set of "real" problems.

Acknowledgements

My thanks to the co-organisers of IPC5: Alfonso Gerevini, Alessandro Saetti and Yanis Dimopolous; to the competitors, in particular Jorge Baier, J. Benton and Stefan Edelkamp; and to the organisers of this workshop: Minh Do, Malte Helmert and Ioannis Refanidis. All have been very helpful to me in this work.

NICTA is funded through the Australian governements *backing Australias ability* initiative, in part through the Australian research council.

References

Baier, J., and McIlraith, S. 2006. Planning with temporally extended goals using heuristic search. In *Proc. of the 16th International Conference on Automated Planning and Scheduling (ICAPS'06)*, 342–345.

Do, M.; Benton, J.; van den Briel, M.; and Kambhampati, S. 2007. Planning with goal utility dependencies. In *Proc. 20th International Conference on Artificial Intelligence (IJCAI'07)*, 1872–1878.

Edelkamp, S. 2006. On the compilation of plan constraints and preferences. In *Proc. of the 16th International Conference on Automated Planning and Scheduling (ICAPS'06)*, 374–377.

Fink, A., and Voss, S. 1999. Applications of modern heuristic search methods to pattern sequencing problems. *Computers & Operations Research* 26:17–34.

Garcia de la Banda, M., and Stuckey, P. 2005. Dynamic programming to minimize the maximum number of open stacks. In *Constraint Modelling Challenge 2005*. `http://www.dcs.st-and.ac.uk/~ipg/challenge/`.

Gerevini, A., and Long, D. 2006. Plan constraints and preferences in PDDL3. In *5th International Planning Competition Booklet*. Available at `http://zeus.ing.unibs.it/ipc-5/`.

Gerevini, A.; Saetti, A.; and Serina, I. 2006. An approach to temporal planning and scheduling in domains with predictable exogenous events. *Journal of AI Research* 25:187–231.

Halldórsson, M. 2000. Approximations of weighted independent set and hereditary subset problems. *Journal of Graph Algorithms and Applications* 4(1):1–16.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. In *Proc. 20th National Conference on AI (AAAI'05)*, 1163–1168.

Kako, A.; Ono, T.; Hirata, T.; and Halldórsson, M. 2005. Approximation algorithms for the weighted independent set problem. In *WG'05*. `http://www.hi.is/~mmh/papers/WIS_WG.pdf`.

Linhares, A., and Yanasse, H. 2002. Connection between cutting-pattern sequencing, VLSI design and flexible machines. *Computers & Operations Research* 29:1759–1772.

Long, D., and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *Journal of AI Research* 20:1–59.

Smith, B., and Gent, I. 2005. Constraint modelling challenge 2005. `http://www.dcs.st-and.ac.uk/~ipg/challenge/`.

Smith, D. 2004. Choosing objectives in over-subscription planning. In *Proc. 14th International Conference on Automated Planning & Scheduling (ICAPS'04)*, 393–401.