Admissible Makespan Estimates for PDDL2.1 Temporal Planning

Patrik Haslum
Australian National University
patrik.haslum@anu.edu.au

Abstract

Though recent renewed interest in "expressive" temporal planning, as exemplified by PDDL2.1, has led to advances, few such planners to date can guarantee optimality, or even bounded suboptimality, w.r.t. plan makespan. As a step towards improving planners' awareness of the quality of their plans, we describe three ways of obtaining lower bound functions, i.e., admissible heuristics, for plan makespan for PDDL2.1 problems. Two directly make use of heuristics for other planning models (conservative temporal and additive cost planning), thus enabling PDDL2.1 planners to take advantage of current and future developments of such heuristics.

Introduction

Planning concurrent activities, planning to meet deadlines or to exploit windows of opportunity, and managing other temporal constraints are often cited as important capabilities for automated planners to tackle realistic problems. The PDDL2.1 planning problem modelling language is expressive enough to capture many features of such problems, though perhaps lacking a bit in modelling convenience.

Currently, it appears the most effective way of planning for PDDL2.1 is to decouple the logical and temporal aspects of the problem as far as possible, searching the space of event (action start and end) sequences and maintaining only enough temporal information to ensure that the chosen sequence is schedulable (Long and Fox 2003; Halsey, Long, and Fox 2004; Cushing et al. 2007a; Coles et al. 2008). The advantage of this decoupling lies in that it enables planners to guide search using state space planning heuristics so as to reduce the number of "steps" taken to reach the goal, leading to less search and faster plan generation. The downside is that since the length of the event sequence is not generally correlated with plan execution time (makespan), this strategy does not guide search towards plans with small makespan and leaves the planner unable to give any guarantees of optimality (or even bounded suboptimality), except by exhausting the entire space of plans.

Although it is understood in principle how to combine the temporal expressivity of PDDL2.1 with guaranteed makespan optimality, most practical planners that generate makespan-optimal plans have adopted the simpler, "conservative", temporal planning model (e.g. Smith and Weld 1999; Haslum and Geffner 2001; Vidal and Geffner 2004). Exceptions include TPSys (Garrido and Onaindia 2006), which works with the PDDL2.1 model and is optimal w.r.t. makespan for a subclass of problems, though not for the full range of problems expressible in PDDL2.1, and CPPlanner (Dinh, Smith, and McCluskey 2003), which is like TPSys planning graph-based and assumes a model slightly different from PDDL2.1, but which is also temporally expressive. Recently, Huang, Chen & Zhang (2009) have proposed a SAT encoding of PDDL2.1 planning which bounds plan makespan, and hence can ensure makespan optimality by iteratively trying encodings with increasing bounds.

In this paper, we discuss three ways of deriving lower bound functions (i.e. admissible heuristics) for plan makespan for PDDL2.1 problems. Through the use of suitable relaxations, admissible makespan estimators for PDDL2.1 planning can be constructed from admissible heuristics for conservative temporal planning and additive cost plannning. This enables PDDL2.1 planners to take advantage of recent and future developments in the design of such heuristics. Thus, we demonstrate that makespan estimators for PDDL2.1 do not need to be developed from scratch. We also show how a PDDL2.1 makespan estimator can be constructed following the same underlying principle which has yielded the h^m lower bound functions for other planning objectives.

An admissible heuristic alone typically does not make an efficient optimal planner. It is even more unlikely to do so for makespan-optimal planning, since the search spaces associated with this objective often contain many more different states with (apparently) equal value. A search-based PDDL2.1 planner will need to incorporate also effective branching and pruning mechanisms (as exemplified by, e.g., the recent work of Coles, Coles, Fox & Long 2009).

However, lower bound functions have other important uses. For one, they provide a simple way to estimate the relative quality of a given plan, i.e., how far off optimal it is. Even this simple use can make suboptimal planners aware of the quality of the plans they generate, and thus help guide decisions such as whether to be content with the best plan found so far. Thayer and Ruml (2009) show that information from an admissible cost estimate and an estimate of distance in the search space (not necessarily admissible) can be combined to improve efficiency of bounded suboptimal search. Incorporating lower bound information into SAT or CSP en-

codings can also be beneficial (e.g. Vidal and Geffner 2004).

It is difficult to say with any precision how accurate the makespan estimators defined here are, since there are no others to compare them with, and not many existing domain encodings that make full use of PDDL2.1's temporal expressivity. We present only limited experimental results relating two of the heuristics with each other. However, these estimators are simple, and thus make a suitable baseline for future work to improve on.

Temporal Planning In PDDL2.1

PDDL2.1 (Fox and Long 2003) is a durative action formalism, founded on a classical state/transition model. Transitions caused by a durative action, i.e., its effects, are instantaneous, and may take place at the start or at the end of the action. Executability conditions may similarly be required to hold at the start and/or end of the action, or to hold during the (interior of the) interval of execution (the latter called invariant conditions).

The semantics of a durative action a is defined by breaking it up into three instantaneous actions, each a classical STRIPS action described by a (conjunctive) precondition, add and delete sets: a_{start} and a_{end} represent the start and end transitions, respectively, of a, while a_{inv} has no effects but requires the invariant condition of a to hold. In a valid plan, each occurrence of a_{start} must be uniquely matched to an occurrence of a_{end} after an interval of time compatible with the duration of a. (PDDL2.1 allows action durations to be constrained to an interval rather than a single value, meaning that the planner may choose the duration within this interval.) An instance of a_{inv} is placed in the space between each pair of transitions in between a_{start} and a_{end} to ensure that the invariant condition holds throughout. Instantaneous actions may take place simultaneously if they are commutative (i.e., neither adds or deletes an effect or precondition of another). Non-commutative instantaneous actions must be separated by a positive but arbitrarily small amount of time, commonly denoted ϵ. The placement of these infinitesimal separations can, in principle, alter the makespan of a plan (though their relative impact can of course be made arbitrarily insignificant by choosing a sufficiently small value for ϵ). In our heuristics we assume $\epsilon = 0$, which eliminates this dependency. Though PDDL2.1 semantics requires a strictly positive value, it is a valid assumption from which to derive lower bounds.

PDDL2.1 can express problems that can only be solved by plans with concurrent actions: in this respect, differs from classical (STRIPS) planning, and the so called conservative temporal planning model (discussed in the next section). The worst case complexity of deciding plan existence for PDDL2.1 is higher than for classical planning – EXPSPACE instead of PSPACE – but the worst case can arise only in problems where an exponential number of instances of the same action must execute concurrently (Rintanen 2007). Cushing *et al.* (2007a; 2007b) studied conditions separating temporal planning problems which may require concurrency in their solutions from ones that do not. The ones that do not are those that have the following property.

Definition 1 (Cushing et al. 2007a) *A PDDL2.1 planning problem P is* inherently sequential *iff for every plan S for P, there exists a causally equivalent*[1] *rescheduling of S that has no concurrently executing actions.*

In addition to – and largely orthorgonally to – its temporal aspects, PDDL2.1 extends classical, discrete, STRIPS planning with real-valued state variables ("fluents"). The makespan estimators we define are unaffected by numeric fluents, except that lower bounds obtained by adapting an underlying (conservative temporal or additive cost) heuristic obviously depend on how well this heuristic is able to cope with fluents, and that numeric fluents can confound the (syntactic) conditions for identifying sequential action sets.

The Conservative Model

The conservative planning model extends classical STRIPS planning only by attaching a duration to each action. The model makes minimal assumptions about action execution: preconditions are required to hold at the start of execution and must remain non-interfered with, i.e., not destroyed by a concurrent action, throughout; effects (adds and deletes) take place at unspecified times in the interior of the interval of execution, and can be relied on to hold only at the end. This model is less expressive than PDDL2.1, in that planning problems under this interpretation are always inherently sequential (in the sense of definition 1) and therefore deciding plan existence remains in PSPACE.

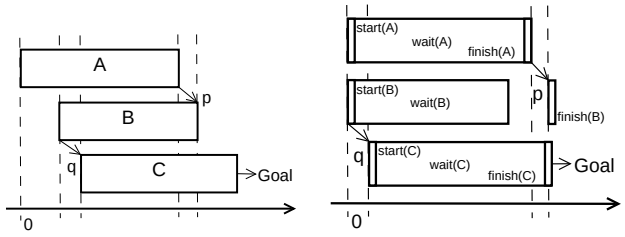
Nevertheless, the conservative model is very useful. In particular, a number of admissible makespan estimators, e.g., the temporal h^m heuristic (Haslum and Geffner 2001), and estimators based on variations of the planning graph (Smith and Weld 1999) or constraint propagation (Vidal and Geffner 2004), are known for this model.

A difficulty arises when two instantaneous actions execute simultaneously, and one establishes a precondition of the other: is this a valid plan? Smith & Weld (1999) and Haslum & Geffner (2001) both avoid the problem by disallowing instantaneous actions (requiring durations to be strictly positive); PDDL2.1 avoids it by disallowing the two actions being scheduled concurrently. Here, we'll adopt the convention that a plan is valid in the conservative model if sets of concurrent instantaneous actions in the plan are non-interfering, and there exists a sequencing of them that is valid, in the classical sense. This is adequate for the purpose of deriving admissible makespan estimates, since it is a permissive ("optimistic") assumption.

Conservative Relaxation of PDDL2.1 Problems

A subclass of PDDL2.1 problems have the same set of plans when interpreted according to the conservative model. We'll call such problems *conservative equivalent*. This implies inherent sequentiality, but the opposite is not true (i.e., it is a stronger requirement). Figure 1(a) illustrates.

[1] Causally equivalent plans have the same set of causal links, and hence the same order between establishers and users of conditions that hold at some point during execution of the plan.



(a) (b)

Figure 1: (a) Optimal plan for a PDDL2.1 problem. (Arrows between actions denote causal links.) The problem is not conservative equivalent because this plan is invalid under a conservative interpretation of the actions. The problem is inherently sequential because rescheduling actions A, B and C in sequence also yields a valid plan.
(b) Optimal plan for the conservative relaxation of the same problem. The relaxation allows the duration of action B to be stretched, and thus the goal to be reached in less time.

For any PDDL2.1 problem, P, a conservative equivalent problem P' can be constructed that is a *relaxation* of P, in the sense that any plan for P corresponds to a plan for P', with equal makespan (definition 2). Thus, the estimate of any makespan heuristic that is admissible for the conservative model applied to the conservative relaxation is a lower bound for the original PDDL2.1 problem P. This makes it possible to apply existing admissible makespan heuristics also to PDDL2.1 problems.

Definition 2 *Let P be a PDDL2.1 planning problem.*
(i) The conservative relaxation of P *is the following problem P': For every action a of P, P' has three counters, $\#started(a)$, $\#expired(a)$ and $\#active(a)$, and three actions* $\mathrm{start}(a)$, $\mathrm{wait}(a)$ *and* $\mathrm{finish}(a)$.
$\circ$ *Preconditions and effects of* $\mathrm{start}(a)$ *are the at-start conditions and effects of a, and the effects $\#started(a)$ += 1 and $\#active(a)$ += 1. The duration of* $\mathrm{start}(a)$ *is zero.*
$\circ$ *Invariant conditions* $\mathrm{wait}(a)$ *are the invariant conditions of a and $\#started(a) > 0$, and the sole effect of* $\mathrm{wait}(a)$ *is $\#expired(a)$ += 1 at-end. The duration of* $\mathrm{wait}(a)$ *equals the (minimum) duration of a minus 2ϵ.*
$\circ$ *Preconditions and effects of* $\mathrm{finish}(a)$ *are the at-end conditions and effects of a, the condition $\#expired(a) > 0$ and the effects $\#started(a) -= 1$, $\#expired(a) -= 1$ and $\#active(a) -= 1$. The duration of* $\mathrm{finish}(a)$ *is zero.*
The initial state of P' is the same as that of P, with all counters initiallised to zero, and the goal of P' is the goal of P plus the requirement $\#active(a) = 0$ for every action a.
(ii) The propositional relaxation of P' *is a problem P'' obtained by replacing in P' each counter $\#c$ with propositions $\#c > 0$ and/or $\#c = 0$, as required to express action conditions and goal. Actions that increment $\#c$ add $\#c > 0$ and delete $\#c = 0$, while actions that decrement $\#c$ add $\#c = 0$.*

Definition 2(*i*) leaves open how counters are represented in the planning problem. In PDDL2.1, they may be represented by numeric-valued fluents. If the maximum value each counter can reach is bounded, they may be represented by sets of propositions. Interestingly, due to the restricted form of conditions on the counters (only the goal tests for equality with zero), P' even with unbounded counters falls within a class of problems that are decidable (Mayr 1984).

Theorem 3 *Let P be a PDDL2.1 planning problem, P' the conservative relaxation of P and P'' the propositional conservative relaxation of P: P' and P'' are conservative equivalent and relaxations of P.*

The conservative relaxation enforces all of PDDL2.1's requirements for plan validity except the upper bound on action duration, since the start, middle and end of each PDDL2.1 durative action is split into separate actions which are not forced to follow each other consecutively. Although enforcing minimum action durations is normally more important for obtaining accurate lower bounds on makespan, there are situations in which the relaxation allows shorter plans than the original problem. Figure 1(b) illustrates.

Relaxed Regression Heuristics for PDDL2.1

Admissible makespan estimators for the conservative temporal planning model have been constructed based on different principles, such as planning graphs and the h^m relaxation. Those same underlying principles can be applied to the PDDL2.1 model. As an example, we consider the "relaxed regression" idea that underpins the h^m heuristic.

The h^m heuristic is defined as the (point-wise greatest) fixpoint of the equation

$$h^m(c) = \begin{cases} 0 & \text{if } c \text{ holds initially} \\ \min_{R(c,d,c')} h^m(c') + d & \text{if } |c| \leqslant m \\ \max_{c' \leftarrow c, |c'| \leqslant m} h^m(c') & \text{if } |c| > m \end{cases}$$

where $R(c, d, c')$ is the regression relation, i.e., c' is a condition such that if c' can be achieved at some cost t, then c can be achieved at cost $t + d$, and $c' \leftarrow c$ is a relaxation relation, i.e., c' is a "simpler" condition implied by c. The definition also depends on a measure of condition size, $|c|$. In classical (additive cost) planning, conditions are conjunctions (sets) of atoms, $R(c, d, c')$ holds iff $c' = (c - \mathrm{add}(a)) \cup \mathrm{pre}(a)$ for some action a such that $\mathrm{del}(a) \cap c = \emptyset$ and $d = \mathrm{cost}(a)$, and the relaxation relation is $\subseteq$.

For conservative temporal planning, conditions take the form of a pair $c = (E, F)$, where E is an atom set (subgoal to be achieved) and F a set of concurrently planned actions with start times relative to the time at which c is achieved, regression allows for concurrent actions (compatible with those in F) and no-ops, and the relaxation relation considers several ways of reducing (E, F) to a simple atom conjunction, to which $\subseteq$ is applied. (For example, action a starting at δ time units before condition c implies condition $pre(a) \cup c$ at time $+\delta$. The heuristic is described in detail by Haslum, 2006.)

In PDDL2.1, conditions need to be sets of atomic point and interval conditions and effects, distributed in time relative to some point, which we arbitarily choose to be the

earliest among them. An atomic point condition, i.e., p true at t, can be achieved by an action a that starts at (or before) t, if a adds p at-start, or at $t - \mathrm{dur}(a)$, if a adds p at-end. For a to start at t, its at-start and invariant conditions must be achieved by t, and its at-end conditions by $t + \mathrm{dur}(a)$. This implies each atomic condition at the respective points. Taken together, we have the following definition of h^1 for PDDL2.1 temporal planning:

$$\begin{aligned} h^1(p) &= 0 \quad \text{if } p \text{ holds initially} \\ h^1(p) &= \min\big(\min_{a:p\in\text{start-add}(a)} h^1(a), \\ &\qquad \min_{a:p\in\text{end-add}(a)} h^1(a) + \mathrm{dur}(a)\big) \\ h^1(a) &= \max\big(\max_{q\in\text{start-cond}(a)\cup\text{inv}(a)} h^1(q), \\ &\qquad \max_{q\in\text{end-cond}(a)} h^1(q) - \mathrm{dur}(a)\big) \end{aligned}$$

Like h^1 heuristics for other kinds of planning, this is a weak bound. However, it does yield the optimal estimate for the example in figure 1(a), which conservative temporal h^1, applied to the conservative relaxation in figure 1(b), does not. PDDL2.1 versions of the general h^m heuristic can be similarly defined, though they become more complex.

Bounded Concurrency

Recently, swift advances have been made in the design of accurate admissible heuristics for planning with additive action costs (e.g. Karpas and Domshlak 2009; Helmert and Domshlak 2009). When no concurrency is possible, the additive objective function, taking minimum duration as action cost, equals makespan. In general, a lower bound on makespan can be derived from an admissible estimate of "sum of durations" whenever the amount of possible concurrency can be (non-trivially) bounded. Clearly, this is not possible for every planning problem. Consider, for example, the resource production problem described by Chan *et al.* (2007), where more resources allow more concurrency, which in turns allows faster production of even more resources, which further increase concurrency. But the method can yield a good makespan estimate in problems that feature some form of statically limited and highly contended resource. As an example, results are presented for instances of the Satellite domain.

The key concept is that of a *sequential action set* (definition 4). Intuitively, this is a set of actions that can not appear concurrently in any plan that is optimal w.r.t. additive cost (sum of durations). The maximum concurrency among some set of actions in any such plan can then be bounded by the number of sequential action sets needed to cover the actions. Most admissible heuristics for additive cost assume sequential plans, so if the PDDL2.1 problems requires a concurrent solution, the heuristic may consider it unsolvable and return an estimate that is too high (typically, ∞). If the problem is not (known to be) inherently sequential, the heuristic can be applied to a conservative relaxation of the problem instead.

Definition 4 *Two actions a and b that can never execute concurrently in any valid plan are called* non-concurrent.

Two actions a and b such that whenever a and b appear concurrent in a valid plan, one of them can be removed without invalidating the plan, are called concurrent redundant.

A sequential action set *is a set of actions A such that for any two actions $a, b \in A$ (including where a and b are the same action), a and b are either non-concurrent or concurrent redundant.*

That two actions "execute concurrently" here means that there is a point common to the interior of their intervals of execution. In PDDL2.1, it is possible to write actions whose end points may be concurrent while at the same time no overlap between the interiors of their execution intervals is possible. We do not consider these actions to be able to "execute concurrently", because the amount by which they overlap is infinitesimally small, so the increase in makespan required to properly separate them is negligeable.

The exemption of concurrent redundant actions is motivated by the fact that they appear in existing benchmark domains, though in most cases it seems such actions could be reformulated so as to be non-concurrent instead, without violating the intent of the domain definition. The key property of concurrent redundant actions is that as a source of possible concurrency they can be ignored when considering optimal plans.

Lemma 5 *Let P be a PDDL2.1 planning problem and let S be a plan for P. There exists a plan S' for P that contains no concurrently executing concurrent redundant actions, such that neither the makespan nor the sum of action durations of S' is greater than that of S.*

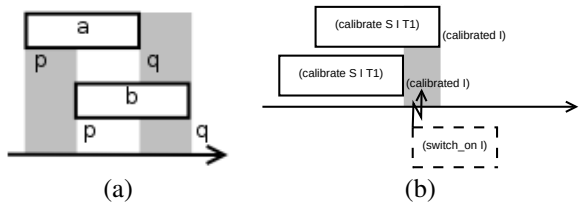
Proof: If S contains a pair of concurrent redundant actions executing over overlapping intervals of time, by definition one of them can be removed without invalidating the plan. This is repeated until no such pair remains. Neither makespan nor sum of action durations can be increased by removing actions from the plan. □

Theorem 6 *Let P be an inherently sequential PDDL2.1 planning problem, and $A_1, \ldots, A_n$ a collection of (not necessarily disjoint) sequential sets of actions in P. Let $U = A_1 \cup \ldots \cup A_n$. For any heuristic function h that is admissible w.r.t. additive cost, taking the cost of each action in U to equal its minimum duration and the cost of each action not in U to be zero, and for any state s and condition c of P, $h(s,c)/n$ is a lower bound on the makespan of any plan achieving c from s (with s and c as the initial state and goal condition, respectively, of P as a special case).*

Proof: Let S be a plan for c from s, of minimum makespan T. By lemma 5, there exists a plan S', of equal or lesser makespan, which contains no concurrently executing concurrent redundant actions.

At no point during execution of S' can more than n of the actions in U be in progress: if there were more, at least two of them would have to belong to the same sequential action set, which is impossible since concurrent redundant actions have been eliminated. Therefore, the sum of durations of actions belonging to U in S' is at most nT, and since h is admissible w.r.t. additive cost, $h(s,c) \leq nT$. Hence, $h(s,c)/n \leq T$. □

There is a trade-off between the number of sequential action sets and the set of actions whose durations are counted (U).



(a) (b)

Figure 2: Concurrent redundant actions: (a) illustration of condition R1; (b) an example from the Satellite domain.

Several strategies for deciding which to use are conceivable: one is to select a minimum number of sets to cover all actions in the problem, another is to search (e.g., greedily) for a set of sequential action sets that maximises the estimate. Which method yields the best result naturally varies between problems.

Finding Sequential Action Sets

To apply the h/n estimate, a collection of sequential action sets must be known. This may be part of the problem description, if it is formulated in a language that models concurrency by parallel composition of sequential subsystems (as is the case for many model checking formalisms, e.g., the Promela language of the SPIN model checker). For an action-centered language like PDDL2.1, however, sequential action sets must be found by analysis of the domain and problem descripions. We demonstrate only a few simple conditions for proving action sets sequential. Putting them together into an algorithm is straightforward. These conditions are not exhaustive: it is easy to find examples that they do not cover. In particular, as they are "syntactical" in nature, their applicability is sensitive to the problem formulation. It is often possible to formulate the actions of a planning domain in different ways that are equivalent, in the sense of admitting equivalent plans, but such that the extraction of sequential action sets works for one formulation but not another. This difficulty is common to all reasoning that relies on the *form* of action definitions.

Actions that are not relaxed reachable or relevant (in the sense of being reachable by back-chaining from the goal) can be ignored, as an optimal plan without such actions always exists. A relevant effect is an effect that may contribute to achieving a condition of a (relevant) action or a goal.

The first thing required is a test for concurrent redundancy between actions, in particular to detect when an action is concurrent redundant with itself.

R1. Two actions, a and b, are concurrent redundant if they have the same relevant at-start and at-end effects, none of which are cumulative numeric effects, and any event (action start or end) that can negate any of those effects either destroys, or has a precondition that is mutex with, the invariant conditions of a and b.

Proof: Suppose a and b execute over partially overlapping intervals, as shown in figure 2 and that both add p at-start (resp. q at-end). No event deleting p (q) can take place in the time between the points where the two actions add it (shaded areas in figure 2) because the invariant condition of at least one of a and b has to hold throughout. Hence, if the later action is removed, p (q) persists until the point where it would have been added the second time. □

Next, we'll need the concept of a *mutex condition set*: a set of conditions, $X = c_1, \ldots, c_k$, at most one of which holds in any reachable state. This is a simple generalisation of the well-known "at-most-one" invariant, in which all conditions are single atoms. If X and Y are mutex condition sets then so is $X \times Y = \{c_x \wedge c_y \,|\, c_x \in X, c_y \in Y\}$. Note that any singleton set is a mutex condition set. We say that a sequential action set A is *associated with* mutex condition set X iff A contains all actions that add some atom appearing in X and each action in A is non-concurrent or concurrent redundant with any action which invariant condition implies some $c \in X$. This is a stricter requirement than A being a sequential action set, but useful in the process of identifying larger sequential sets.

S1. The action set $\{a\}$ is sequential if a is non-self-concurrent or concurrent self-redundant.

A1. Let X be a mutex condition set and $A(X)$ the set of all actions that add some atom appearing in X. If every action in $A(X)$ requires some $c \in X$ at-start, deletes c at-start and only adds some $c' \in X$ at-end (if at all), then $A(X)$ is a sequential action set, and associated with X.

A2. Let p be a proposition such that $\neg p$ is not relevant (i.e., does not appear in any (relevant) actions condition or the goal) and $A(p)$ the set of all actions that add p. If $A(p)$ is a sequential action set, and each action in $A(p)$ has no other relevant effects, adds p at-end, and is concurrent self-redundant according to R1, then $A(p)$ is associated with the mutex set $\{\neg p, p\}$.

C1. If A and B are sequential action sets, such that any pair of actions $a \in A$ and $b \in B$ have mutex invariant conditions, then $A \cup B$ is a sequential action set. Moreover, if A and B are associated with mutex condition sets X and Y, respectively, then $A \cup B$ is associated with $X \times Y$.

C2. If A is a sequential action set associated with mutex condition set X and B is a sequential action set such that every action in B requires some $c \in X$ over-all, then $A \cup B$ is a sequential action set. Moreover, if B is associated with mutex condition set Y then $A \cup B$ is associated with $X \times Y$.

The first three conditions provide basic building blocks, while C1 and C2 allow for sequential action sets to be combined into larger sets. If condition A2 looks like a complicated special case, that is because it is: if the domain is reformulated so that actions that add p perform an explicit durative transition from $\neg p$ to p, A1 applies instead. For well structured domains, conditions A1 and C2 are likely to be sufficient.

Partially Dependent Action Sets

The h/n makespan estimate optimistically assumes that the sum of action durations lower-bounded by h can be evenly distributed over the n action sets. Typically, this is not the case. With a more detailed analysis of the "reasonable" action sequences in each set it is sometimes possible to lower the divisor, and thus increase the makespan estimate.

Definition 7 *Let A and B be sequential action sets, A is associated with a mutex condition set X, and $b_1, \ldots, b_k$ a sequence of actions in B. The sum of durations of actions b_i in the sequence that have an invariant condition which implies some $c \in X$ divided by the total duration of the sequence is the* dependency ratio *(on A) of the sequence. The minimum over all sequences of actions in B that may appear in an optimal (w.r.t. makespan) plan is the* dependency ratio of B on A.

Note that if every action in B has an invariant condition on X, the dependency ratio is 1. The union of the two sets is then a sequential action set (*cf.* condition C2).

Theorem 8 *Let A and B be sequential action sets, such that B depends on A with ratio r. Let S be a plan with minimum makespan T. The sum of durations of actions in S belonging to A and B is at most $(2 - r)T$.*

Proof: By lemma 5, we can assume that S contains no concurrently executing redundant actions. Let $b_1, \ldots, b_k$ be the sequence of actions in B that appear in S. The total duration of this sequence is at most T. Because its dependency ratio on A is at least r, for at least rT of the duration of the plan no action in A is executing. Thus, the sum of durations of actions in A that appear in S is at most $(1 - r)T$, and the sum over both A and B at most $(2 - r)T$. □

Thus, $A \cup B$ can be seen as a "semi-sequential" action set with a "weight" equal to $2 - r$, where a sequential set has weight 1. Instead of dividing the value of the additive cost heuristic h by the number of sets, n, we can divide it by their total weight, and still obtain an admissible makespan estimate.

To find the true dependency ratio of B on A would require us to examine all makespan-optimal plans, which is clearly infeasible. But an underestimate of the ratio may be obtained by minimising over a superset of sequences of actions in B. Let $\mathrm{state}(B)$ denote the set of propositions that are added by and only by actions in B and that appear in conditions of actions in B, i.e., $\mathrm{state}(B)$ is the state abstraction that is fully controlled by actions in B and plays a part in determining executability of those actions. Futhermore, let $B' \subset B$ contain actions that have an effect relevant to some action not in B or to the goal. The sequence of actions in B appearing in a (non-redundant) makepan-optimal plan must be executable, starting from the initial state, in the abstraction onto $\mathrm{state}(B)$; end with an action in B'; and, if the sequence loops w.r.t. $\mathrm{state}(B)$, each loop must contain an action in B'. Thus, the minimum dependency ratio of all sequences is lower-bounded by the minimum over (1) all non-looping, w.r.t. $\mathrm{state}(B)$, sequences ending with an action in B', and (2) all reachable simple loops, w.r.t. $\mathrm{state}(B)$, containing at least one action in B', and thus can be found by a simple depth-first search. However, the complexity of the search is exponential in the size of $\mathrm{state}(B)$.

Preliminary Experimental Results

We illustrate the potential (and some flaws) of the h/n makespan estimate with 30 problems of the IPC 2002 Satellite Timed domain. Problem were created using the IPC generator [2] and have 1–2 satellites and 2–19 goals. The additive cost heuristic h is Helmert's & Domshlak's (2009) landmark cut heuristic. Two strategies for selecting sequential action sets are tried: one selects minimum sets to cover all actions in the problem, while the other chooses a collection of sets covering the actions that add a goal proposition, maximising the estimate over all such collections. The baseline is temporal h^2 applied to the propositional conservative relaxation.

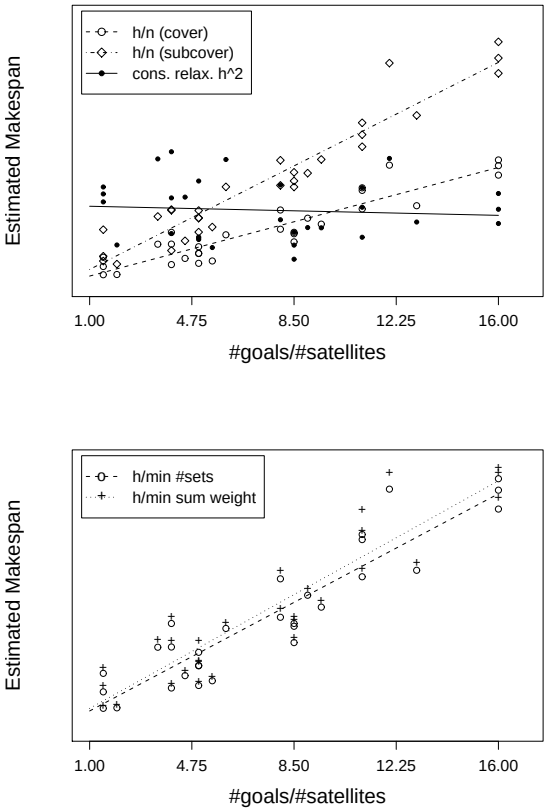


Figure 3: Comparison of makespan estimates in the Satellite domain. Lines show best-fit (sum squared error) linear models for each data set.

Figure 3(a) shows the result. The picture is somewhat noisy, due to large random variations in action durations. Therefore, linear models (min sum squared error fit) for each set of data are also shown. Yet, it can be observed that the h/n estimate trends upwards as the ratio of the number of problem goals to the number of satellites (which determines the level of concurrency in this domain) increases, while the temporal h^2 estimate stays flat. Also, for this domain at least, selecting subcovers yields a better estimate than covering all actions.

Figure 3(b) shows the impact of dependency ratio analysis

[2] `http://planning.cis.strath.ac.uk/competition/domains.html`

(for the full action set cover strategy only). It does improve the makespan estimate, but only marginally.

Conclusions

We have described three ways of obtaining lower bound functions for plan makespan for PDDL2.1 problems. Two of them directly make use of heuristics for other planning models (conservative temporal and additive cost planning), so that recent and future improvements of such heuristics are immediately available also for planning in PDDL2.1.

Admissible heuristics alone are unlikely to make an efficent makespan-optimal planner, but they are one important building block in making PDDL2.1 planners better aware of the quality of plans they construct. It remains to explore what are the best ways to make use of this information in a planner. How accurate the makespan estimators are is also an open question. As we have shown, the h/n estimate can be better than the (conservative relaxed) h^2 makespan heuristic for problems with limited and highly contended resources, but also has serious weaknesses: the assumption that the sum of action durations distributes evenly is overly optimistic, and the identification of sequential action sets is highly sensitive to problem formulation.

Sequential action sets may also have further uses. For example, given a makespan bound T and a sequential action set A, it is possible to find the longest sequnce of actions in A with a total duration of at most T (although it is NP-hard; Aho 2000). Thus, from a covering collection of sequential action sets an upper bound on the length of any event sequence that can possibly result in a plan with makespan less than T can be obtained by summing the lengths of such longest sequences. This can help also a planner that searches the space of event sequences prove plan optimality.

References

Aho, I. 2000. Interactive knapsacks. *Fundamenta Informicae* 44(1-2):1–23.

Chan, H.; Fern, A.; Ray, S.; Wilson, N.; and Ventura, C. 2007. Online planning for resource production in real-time strategy games. In *Proc. of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, 65–72.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. Planning with problems requiring temporal coordination. In *Proc. AAAI*.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2009. Extending the use of inference in temporal planning as forwards search. In *Proc. 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. 2007a. When is temporal planning really temporal? In *Proc. 20th International Conference on Artificial Intelligence (IJCAI'07)*, 1852–1859.

Cushing, W.; Kambhampati, S.; Talamadupula, K.; Weld, D.; and Mausam. 2007b. Evaluating temporal planning domains. In *Proc. of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, 105–112.

Dinh, T. B.; Smith, B. M.; and McCluskey, T. L. 2003. CPPlanner: A domain-independent temporal planner with critical paths. In *Proc. 22nd UK PlanSIG*. `http://planning.cis.strath.ac.uk/plansig/index.php?page=past22`.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of AI Research* 20:61–124.

Garrido, A., and Onaindia, E. 2006. Domain-independent temporal planning in a planning-graph-based approach. *AI Communications* 19(4):341–367.

Halsey, K.; Long, D.; and Fox, M. 2004. CRIKEY: A temporal planner looking at the integration of scheduling and planning. In *Proc. Workshop on Integrating Planning into Scheduling (WIPIS'04)*.

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In *Proc. 6th European Conference on Planning (ECP'01)*, 121–132.

Haslum, P. 2006. Improving heuristics through relaxed search – an analysis of TP4 and HSP^*_a in the 2004 planning competition. *Journal of AI Research* 25.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*.

Huang, R.; Chen, Y.; and Zhang, W. 2009. An optimal temporally expressive planner: Initial results and application to P2P network optimization. In *Proc. 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In *IJCAI*.

Long, D., and Fox, M. 2003. Exploiting a graphplan framework in temporal planning. In *13th International Conference on Automated Planning and Scheduling (ICAPS'03)*, 51–62.

Mayr, E. 1984. An algorithm for the general Petri net reachability problem. *SIAM Journal on Computing* 13(3):441–460.

Rintanen, J. 2007. Complexity of concurrent temporal planning. In *Proc. of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, 280–287.

Smith, D., and Weld, D. 1999. Temporal planning with mutual exclusion reasoning. In *Proc. 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, 326–333.

Thayer, J. T., and Ruml, W. 2009. Using distance estimates in heuristic search. In *Proc. 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*.

Vidal, V., and Geffner, H. 2004. Branching and pruning: An optimal temporal POCL planner based on constraint programming. In *Proc. 19th National Conference on Artificial Intelligence (AAAI'04)*, 570–577.