Cost-Sensitive Concurrent Planning Under Duration Uncertainty for Service-Level Agreements

Amanda Coles and Andrew Coles
Dept. of Computer and Information Sciences,
University of Strathclyde, Glasgow, UK
email: {amanda,ac}@cis.strath.ac.uk

Allan Clark and Stephen Gilmore
School of Informatics,
University of Edinburgh, UK
email: {a.d.clark,stephen.gilmore}@ed.ac.uk

Abstract

This paper brings together work in stochastic modelling, using the process algebra PEPA, and work in automated planning. Stochastic modelling has been concerned with verification of system performance metrics for some time: given a model of a system, determining whether it will meet a service-level agreement (SLA). For example, whether a given sequence of transitions on a network will complete within 5 seconds 80% of the time. The problem of deciding how to reconfigure the system most cost-effectively when the SLA cannot be met has not been widely explored: it is currently solved manually. Inspired by this, we consider how planning can be used to automate the configuration of service-oriented systems. Configuring these stochastic systems presents new challenges to planning: building plans that meet SLAs, but also have low cost. To this end, we present a domain-independent planner for planning problems with action costs and stochastic durations, and show how this can be used to solve both traditional planning domains, and within the framework of configuring a larger process algebra model.

1 Introduction

In this paper we bring together two established areas of computer science research to solve the problem of system configuration to meet *service-level agreements* (SLAs). SLAs give guarantees about service time for a system (e.g. completion within 10 seconds 90% of the time), given that there is uncertainty in the execution time of each component.

SLAs have wide applicability, for example a large online retailer may wish to offer a delivery service with a guaranteed deadline (paying a penalty if the deadline is missed); but there is uncertainty in the time taken to select, pack and send items. SLAs are also important in Internet and network service provision, where guarantees are sought on the performance of a larger system that are combinations of smaller ones. For example, an online travel agent may need to search flight, hotel and taxi systems. Expected system performance here is key: customers will leave if response is slow.

Work on stochastic modelling, using process algebras, has long considered the question of whether a system will meet an SLA. The strength of process algebra models is their ability to model uncertainty accurately in potentially large systems, with looping behaviour and many concurrent users. We use the process algebra PEPA (Hillston 1996) because it, uniquely, has a differential equation semantics which can be applied to modelling large-scale systems without the need for complete state space enumeration. A range of techniques including model-checking, discrete and continuous simulation can be used to determine whether a given system meets an SLA. If the SLA is met, the system can be deployed with the desired performance guarantee. The difficulty arises, however, when the SLA is *not* met: the techniques used are designed for *measuring* properties of models; rather than *creating* such systems. For example, there may be the opportunity to upgrade parts of the system (at some cost) and the challenge is to find a low cost (ideally the lowest cost) set of upgrades that achieve some SLA. Currently this is done by manual generation of different models, followed by the use of PEPA to determine if they satisfy the SLAs.

In contrast, the traditional strength of planning is in automated decision making. This provides an opportunity to apply planning, in combination with PEPA, to the task of system configuration. We begin by considering the problem of configuring large systems, illustrating relationships between PEPA models and stochastic planning problems with action costs. In small systems, with non-complex interactions, it is possible to create a planning problem capturing the whole system. But, as PEPA can be used to model systems much larger than those planners can reason with, and with complex interactions between components, we develop a system that iterates between planning and PEPA analysis. The automatic configuration of large systems is made possible by this iteration: neither approach can achieve this in isolation.

Within this framework, two key challenges are posed to the planner: reasoning about uncertainty in action durations in highly concurrent systems (a relatively young field); whilst optimising with respect to some plan metric. Thus, we present a new planner *stochastic-popf* capable of solving such problems effectively, using a combination of anytime search and a novel temporal-and-cost-sensitive heuristic.

2 Context

In this work, we build on the PDDL2.1 (Fox & Long 2003) representation of temporal planning. Here, the start and end of a durative action A can be considered as two instantaneous snap actions, $A_{\vdash}$ and $A_{\dashv}$. These are temporally sep-

arated according to a duration constraint: an upper- and lower-bound on the time between them. Each snap-action a has preconditions $pre(a)$ that must hold prior to its execution, and effects that occur upon execution: $del(a)$, the facts deleted; $add(a)$, the facts added; and $num(a)$, effects on state numeric variables. Additionally, there may be conditions $inv(A)$ that must hold throughout the action's execution. A goal state is one satisfying some constraints on facts and numeric variables, in which no action is executing.

We extend this model to stochastic durations by introducing new keywords to PDDL. Stochastic durations are represented by the function keywords `exponential` or `stochastic-gaussian-mean`. Written thus, for a duration constraint `(= ?duration (exponential ?r))` (where `?r` is an action parameter), the initial value of the function is taken as the mean value of the action's duration. Gaussian distributions further require a definition of `(stochastic-gaussian-standarddeviation ?r)` specifying the standard deviation. Although not relevant to this paper, the syntax can be trivially extended to other types of distribution, e.g. `stochastic-erlang-mean`. Note also that a deterministic planner can parse and run on domains described thus, using the given mean values as (fixed) action durations. SLAs are either given as a command-line parameter, for all goals, or through the use of the `(within)` syntax borrowed from PDDL3 (Gerevini *et al.* 2009). We change the meaning of this: `(within 10 p)` specifies p must hold from time 10, subject to the given (command-line) confidence bound.

2.1 Deterministic Planning with Deadlines

Planning to meet deadlines has received limited specific attention. Deadlines can be modelled explicitly through timed initial literals (TILs), introduced in PDDL2.2, by deleting a fact that makes achieving the goal possible. Alternatively, using required concurrency (Fox, Long, & Halsey 2004; Cushing *et al.* 2007), a fixed-length action can be used to restrict the time available in which other actions can be applied, indirectly enforcing the deadline. Required concurrency is supported by few planners, e.g. (Gerevini, Saetti, & Serina 2010; Coles *et al.* 2008; 2010); whereas TILs are explicitly supported by a number of planners e.g. (Kavuluri & Senthil 2004; Gerevini, Saetti, & Serina 2006) from IPC4.

Although TILs and required concurrency can model deadlines, they have many other uses, and planners to date have typically focussed on general-purpose mechanics, rather than guidance specifically for deadlines. An exception to this is the planner CRIKEY3 (Coles *et al.* 2008), which uses techniques within its heuristic to note when facts will no longer be available. Its successor POPF (Coles *et al.* 2010) exploits reduced commitment through expansion of a partial order making search particularly effective in problems with deadlines in two key ways. First, it avoids unnecessary sequential orderings, favouring concurrent execution, thus improving the temporal efficiency of considered plans. Second, the partial order informs a modified temporal RPG heuristic of the earliest points at which facts can be added/deleted, under the restriction that the partial order is always expanded forwards. This favours construction of relaxed plans whose actions can be added earlier in the plan.

2.2 Temporal Planning Under Uncertainty

Planning under uncertainty in domains with time has received much attention in recent years. There is a great deal of variation, however, in the precise nature of the uncertainty different systems reason with. The level of temporal expressivity varies widely: some planners handle actions with durations (some fixed, others stochastic); others deal with actions occurring in parallel but do not consider durations. Further, some consider uncertainty in action outcomes. We consider stochastic action durations; but not uncertainty in action outcomes. The most common existing optimisation metric is a weighted sum of cost and makespan. However, the challenge we wish to address is different: to meet a fixed SLA (deadline with a confidence bound) with a plan of minimal cost. These are very different, we do not wish to minimise makespan, simply meet the SLA; indeed the lowest cost plan will often be the longest that meets the SLA.

FPG (Buffet & Aberdeen 2009) focusses mainly on the issue of generating plans in the face of uncertain action outcomes with some support for concurrency, using a machine learning approach to generate MDP policies. Two other planners consider uncertainty in action outcomes, Prottle (Little, Aberdeen, & Thiébaux 2005) and Temptastic (Younes & Simmons 2004); both can also reason with uncertainty in action durations. Prottle builds a policy by searching through a space of and/or trees, using MDP algorithms to produce plans that meet a deadline whilst minimising probability of plan failure. Temptastic builds an initial plan ignoring uncertainty and then builds a semi-MDP policy around this, achieving the goal by a deadline with a specified confidence level (it does not optimise cost). The expressivity of both of these planners extends that of our planner to support outcome uncertainty (although neither solves the precise problem we are tackling). Our model of uncertainty, however, requires only uncertainty in durations, not in outcomes, so we can hope to build more scalable systems for these problems, without the need for policies and MDPs.

A difficulty in using MDP-based approaches for temporal planning is the need to discretize time, making it difficult to handle concurrency efficiently, limiting scalability. Mausam & Weld (2008) and Rachelson et. al. (2008) explored several techniques for reducing the blow-up caused by this. The scalability of MDPs is still not comparable to that expected of non-MDP-based planners; but they can offer guarantees that such planners cannot. The model of uncertainty used by Mausam & Weld is, like ours, concerned with uncertainty in action durations but not outcomes; however they optimize a weighted sum of makespan and cost.

The final approach we note, the planner RTU (Beaudry *et al.* 2010), is perhaps the most closely related, using forward-chaining search for planning under resource and time uncertainty. A Bayesian network is used to manage stochastic temporal constraints. Our framework is similar, but we focus on a different optimisation criterion (they use the weighted sum of makespan and cost). RTU uses a heuristic based on that of Sapa to attempt to minimise plan cost; we place much greater emphasis on guidance to optimise cost in this setting, introducing powerful heuristic pruning and anytime search.

$$
\begin{aligned}
BL &\stackrel{def}{=} (\textit{requestLender}, \top).BL_{req} \\
BL_{req} &\stackrel{def}{=} (\textit{transferToLender}, r_{BL}).BL_{wait} \\
BL_{wait} &\stackrel{def}{=} (\textit{transferFromLender}, \top).BL_{resp} \\
BL_{resp} &\stackrel{def}{=} (\textit{responseLender}, r_{BL}).BL \\
LE &\stackrel{def}{=} (\textit{transferToLender}, \top).LE_{think} \\
LE_{think} &\stackrel{def}{=} (\textit{transferFromLender}, r_{LE}).LE
\end{aligned}
$$

$(BL \underset{\mathcal{L}}{\bowtie} LE)$, $\mathcal{L}$=$\{$*transferToLender*, *transferFromLender*$\}$

Figure 1: Example PEPA Model (from GetLoan)

2.3 Performance Modelling Using PEPA

Process algebras provide an effective means for modelling and reasoning with the performance of a system, and share with planning the idea of domain independence. We work with the algebra PEPA (Hillston 1996), one application of which in the literature is the GetLoan system (Bocchi *et al.* 2009). GetLoan models the stages of a loan enquiry, passing data from a customer (CR) over a network (CB) to a broker (BR); then on a network (BL) to a lender (LE); then back, returning a decision. The inner stages are shown in Figure 1. The first four lines define the four states of the network BL, and transitions between them. For instance, *transferToLender* takes it from BL_{req} to BL_{wait} at rate r_{BL} (where $1/r_{BL}$ is the mean of an exponential distribution). As can be seen, BL has looping behaviour: the transition out of BL_{resp} returns it to BL. The definition of LE, the lender, is simpler, alternating between being ready and thinking.

The final line of the figure notes that BL and LE cooperate over the transitions in $\mathcal{L}$, i.e. the transition can be made *iff* both are in appropriate states. Thus, *transferToLender* can only occur if the positions of the components are BL_{req} and LE. The transition rate is the slower of the two: r_{BL} as LE has the (infinite) top rate $\top$ (hence the other component in the cooperation is slower). Intuitively, this means the network speed, r_{BL}, is the factor limiting the rate of data transfer to the lender. In the full GetLoan model, such cooperation occurs between all adjacent stages of the transaction.

PEPA can be used to measure whether an SLA is met, e.g. the customer receives a response within 5 seconds, 80% of the time; this is done by running execution simulations. Suppose that an SLA is not met but several system components could be upgraded to work at faster rates (at some cost), e.g. investing in network infrastructure. If, for example, each component has four possible rates, there is a large space of rate combinations to search to find cost-effective combinations of upgrades: this is where planning can be used.

PEPA is suitable for use on much larger models than this, for instance process arrays can be used to duplicate components, with $BL[100]$ denoting that the network link to the lender is duplicated a hundred times. This has important consequences because in PEPA a given transition occurs probabilistically at the specified rate if its conditions are satisfied. If several BL components are ready to make the transition *transferToLender*, but only one LE is ready, then there is a probabilistic choice of which BL gets to make the transition. We refer to such cases as *race conditions*.

PEPA semantics consider states to have a residence time and transitions to be instantaneous; whereas planning has action durations and instantaneous states. The duration of an action in planning is analogous to state residence time in the state before that activity occurred in PEPA. For each state of a PEPA component we can compute, through simulation, the state residence time in that state (e.g. in Figure 1 we expect BL_{req} to remain for n units before *transferToLender* fires and it becomes BL_{wait}). This is not necessarily the reciprocal of the rate, as the transition may have to await another component. Transition times are calculated using Little's law: the average time spent in a state, S, (the average transition time out of S) is equal to the proportion of all users in S on average divided by the rate at which S is entered.

3 Integrating Planning and PEPA

In this section we explore how stochastic planning can be combined with PEPA models to configure large systems. Each approach has its respective strengths: a planner can automatically search the space of possible system configurations, producing a plan representing system structure; whilst PEPA can effectively scale to large systems, thousands of components, and accurately model performance.

3.1 Common Ground between PEPA and PDDL

To configure sections of PEPA models using a planner we define a mapping from PEPA to PDDL. Intuitively both languages are concerned with states and the transitions between them. In the PEPA model in Figure 1, transitions have implied preconditions and effects. A PDDL action derived from *transferToLender* can be written:

```
(:durative-action transferToLender
:parameters (?r - rate)
:duration (= ?duration (exponential ?r))
:condition (and (at start (BL-req))
  (at start (LE))
  (at start (rateset transferToLender ?r)))
:effects (and
(at start (not (BL-req))) (at end (BL-wait))
(at start (not (LE))) (at end (LE-think)) ))
```

Note the parameter `?r`: as we are concerned with making decisions about PEPA model configuration, we use the predicate `rateset` to record the rate assigned to each transition. These are set by additional dummy actions, which increase plan cost, and irreversibly fix the rate to one from those specified as available (in the problem file).

The key difference between the semantics of PEPA and this PDDL form is where there are multiple actions with equivalent preconditions but different effects and/or durations. The planner can dictate which of these actions is 'applied'; however, in PEPA, which of these actions 'occurs' is a stochastically-timed race: the action whose stochastic delay finishes first gets to make the transition, instantaneously (rather than being temporally separated into start and end). Thus, we can define a direct mapping from PEPA to PDDL only if no such race conditions exist; i.e. the actions chosen in the plan will definitely happen, and the ordering constraints between them will be respected.

3.2 Representing Plans in PEPA

We now describe translation of a plan into a PEPA model that captures its execution semantics. A plan consists of a se-

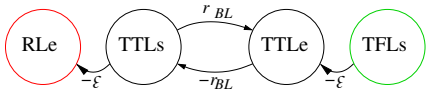

Figure 2: Portion of the STN for GetLoan

$$
\begin{aligned}
E_{i,j} &\stackrel{def}{=} (Oa_j, \top).Eready_{i,j} \\
Eready_{i,j} &\stackrel{def}{=} (E, \top).Epost_{i,j} \\
Epost_{i,j} &\stackrel{def}{=} (Ib_i, \top).Edone_{i,j} \\
(Oa_j \underset{\mathcal{L}_1}{\bowtie} E_{i,j}) & \qquad \mathcal{L}_1 = \{\, Oa_j \,\} \\
(E \underset{\mathcal{L}_2}{\bowtie} E_{i,j}) & \qquad \mathcal{L}_2 = \{\, E \,\} \\
(Ib_i \underset{\mathcal{L}_3}{\bowtie} E_{i,j}) & \qquad \mathcal{L}_3 = \{\, Ib_i \,\}
\end{aligned}
$$

Figure 3: Components Added to PEPA Model

ries of snap actions, with ordering constraints between them due to action durations (separating the start and end of an action) or to ensure that preconditions are met and conflicts are avoided. Such constraints can be represented as a Simple Temporal Network (STN). Without loss of generality, we assume vertices with no outgoing edges have a zero-weight edge back to a vertex denoting time zero; and vertices with no incoming edges have an incoming zero-weight edge back from a vertex denoting that execution has finished.

Each STN edge *E*, of the form $b \xrightarrow{w} a, w \leq 0$, denotes a transition E from a to b, with rate $-1/w$. For instance, Figure 2 shows part of the STN for a solution to GetLoan. The vertices TTLs and TTLe are the start- and end-points of *transferToLender*, with the edge labelled $-r_{BL}$ denoting the duration of this transition. In the general case, for an edge E with weight w, this translates to a basic PEPA component:

$$E \stackrel{def}{=} (E, -1/w).E_{fin} \tag{1}$$

We must ensure the transition *E* can only be performed after its predecessors; and following its execution, its successors can execute. For instance, in Figure 2, the *transferToLender* edge must follow the small gap (duration ϵ) back to RLe (the end of *requestLender*) and precede the ϵ before $TFLs$ (the start of *transferFromLender*). In general, for an *E* represented by edge $b \xrightarrow{w} a, w \leq 0$ we need:

- O, the edges out of a to predecessor vertices; and,
- I, the edges into b from successor vertices.

For each edge in O and I, we create a basic component, as Equation 1. Then, to enforce the correct placement of *E*, for each pair of edges $(Ii, Oj) \in (I \times O)$ we add the components in Figure 3. By synchronizing the $Eready_{i,j} \rightarrow Epost_{i,j}$ transition with the component E over *E*, we ensure that *E* cannot occur until after all such predecessor edges Oa_j have been traversed; and, similarly, until *E* has occurred, we block the traversal of the successor edges Ib_i. With this formulation, a solution plan to a planning problem can be represented as a PEPA model, where the execution traces corresponds to a non-deterministic topological-order traversal over the partial order.

Returning to Figure 2, consider $(TTLe, TTLs)$, for *transferToLender*. Here, O=$\{(TTLs, RLe)\}$) and I=$\{(TFLs, TTLe)\}$), so there is only one pair (Oi, Ij). Thus, to sequence *transferToLender* correctly, Figure 3 is instantiated once. The transition to $Eready_{i,j}$ occurs alongside the transition in O, at which point *transferToLender* can occur; after which, $Epost_{i,j}$ is reached, allowing the transition in I to occur (the ϵ gap before *transferFromLender*).

3.3 Configuring Systems with Planning & PEPA

For more complex systems, with race conditions or many hundreds or thousands of components, the planner cannot accurately capture the model structure, or scale to the extent needed to meet SLAs. In these cases, we alternate between PEPA and planning to solve the problem.

We identify two types of race condition. In the first the race affects only execution duration, this is generally a result of contention: e.g. in GetLoan, if two customers are waiting for one broker, both eventually follow the same path through the system. The second is where a race determines the path taken through the system, for example if the customer could use one of several asymmetric brokers, with each following a different system trajectory to different lenders. Such outcome uncertainty is modelled in PEPA using different rates: from state A a transition is made either at rate $r1$ to state A', or at rate $r2$ to state A''. The ratio between the rates determines the likelihood of each occurring (winning the race).

We focus on the first of these race conditions in this paper, demonstrating iteration between planning and PEPA to configure systems that planning alone could not scale to. The second we leave to future work, extending the planner to consider multiple trajectories. The techniques to allow scalability will be important for both cases. We characterise the subset of PEPA models amenable to our current iteration approach as being *trajectory safe*, an interesting subset of PEPA leading to challenging configuration problems.

A PEPA model is trajectory safe *iff* for each component state there is only one transition out of that state *and* there is no choice over the component(s) definitions with which to interact over it. Note that arrays of processes are permitted under this definition, e.g. there may be several identical copies of the customer, lender and/or broker. Race conditions can therefore arise; e.g. in GetLoan, several customers may be ready to be served but no broker available. We call a trajectory safe model with no process arrays a *linear* model (e.g. GetLoan with a single customer, broker and lender). Such models can be configured using planning alone, using the translation described above, without iteration.

A trajectory safe model can be relaxed to a linear model by ignoring process arrays, i.e. assuming there is only one of each component. Clearly this relaxed model no longer accurately represents the real system, but we can directly build a planning model from it. We obtain more accurate estimates for the average (of the exponentially distributed) action durations, under contention in the non-relaxed model, by performing numerical integration with PEPA and setting them according to the average time for each transition.

Transition times are no longer simply $1/r$ for transitions at rate r due to contention. Consider a narrow bridge (passable slowly) with a (fast) road approaching it. Increasing the speed limit at which cars can drive down the road would appear to have a significant effect on meeting an SLA (on arrival at the other side of the bridge) in the presence of only 1 car; but, in the presence of 100 cars this will simply increase the arrival rate at the bridge, generating a queue. In

the PEPA model this queuing time will be included in the average transition time from the start of the road, to being ready to cross the bridge. In planning this corresponds to extending the duration of the drive action to include time spent queueing. Counter-intuitively, this means that the duration of the drive action may in fact be more dependent on the bridge crossing speed than the road's speed limit.

As a result of contention, we must consider the effect of rate changes on the whole model. For each rate r with upgrade options $[r_1...r_n]$, we use sensitivity analysis on the PEPA model. The other rates are fixed to nominal (initially, non-upgraded) values, and the model is evaluated with r=r_i (for each $i \in [1..n]$) giving average response times corresponding to action durations $[d_j^{r=r1}..d_j^{r=rn}]$ for each action a_j. This process is automated, and much cheaper than solving the configuration problem as it considers changing only one rate at a time. When this has been done for all rates, we identify a rate R such that for each transition t, varying R had the greatest impact on the average response time for t. For the a_j corresponding to t (with duration $d_j^{r=r1}$ when r is $r1$), we select the R that maximises $|d_j^{R=R1}$-$d_j^{R=Rn}|$. Then, a_j has n possible durations, $[d_j^{R=R1}..d_j^{R=Rn}]$, the duration being fixed when the choice to use R=R_i is made during search. In our example, the time to drive along the road may well be taken from the average values in the sensitivity analysis for upgrading the bridge crossing rate.

Another possibility, if a single pair of rates affects the duration of one action is to consider these rates as a pair and run the sensitivity analysis with each combination of upgrades to get the average transition times. Clearly the decision about when to combine rates is important since if we reduce to all combinations of rates we are not using an approximation and have not made a saving on enumerating all models.

The resulting planning model approximates how the system performs, but only around the baseline configuration evaluated by PEPA (i.e. with nominal rates). If multiple rates are upgraded, the approximation becomes less accurate, thus we need to iterate. The planner generates a plan assuming the durations from the first sensitivity analyses; the rates in the PEPA model are then set based on this plan; and average transition times (i.e. action durations) are recalculated again. This gives a better approximation of model performance in solutions close to the last plan found. The planner plans, again, with the new durations, until a plan is produced that PEPA confirms meets the SLA. Due to approximations we cannot guarantee optimality; however, the results are adequate for our test problems, and further, using anytime search returns a number of candidate solutions.

4 Temporal Uncertainty and SLAs in POPF

Having now considered the nature of the planning models arising through integration with PEPA, we extend the approach of POPF to accommodate uncertainty in the duration of actions. We augment the forwards partial order expansion with estimates of the time to attain effects, using these with a temporal RPG heuristic to find temporally efficient relaxed plans, and prune states from which SLAs cannot be met.

4.1 Forwards Partial-Order Expansion

Forwards partial-order expansion in POPF requires a number of annotations for each fact p and each state variable v, these record information relating p and/or v to steps of the plan. For full details, we refer to (Coles *et al.* 2010), in summary:

- $F^+(p)$ ($F^-(p)$) gives the index of the step in the plan that most recently added (deleted) p;
- $FP(p)$ is a set of pairs, each $\langle i, d\rangle$, recording conditions on p. Here, i denotes the index of a step in a plan, and d is either 0 or ϵ. If $d = 0$, then p can be deleted in parallel to step i: this corresponds to the end of a PDDL `over all` condition. If $d = \epsilon$, then p can only be deleted epsilon after i: the end of an `at start` or `at end` condition.
- $V^{\mathit{eff}}(v)$ gives the index of the step in the plan that most recently had an effect upon variable v;
- $VP(v)$ is a set, containing the indices of steps in the plan depending on v. A step depends on v if it either has a precondition on v; an effect needing an input value of v; or is the start of an action with a duration depending on v.

These annotations are updated as each step is added to the plan, and are used as the basis of the ordering constraints added to the partial order. To meet its preconditions, ordering constraints are added to the relevant F^+ entries for propositions, or $V^{\mathit{eff}}(v)$ values for numeric preconditions (ensuring the value is known and defined, and thus the precondition is met). To ensure its effects do not conflict with steps already in the plan, it is ordered after previous steps depending on the affected facts or variables. Finally, when ending an action, the interval between its start and its end is set to obey its duration constraint.

If durations are deterministic, the ordering constraints over the plan steps take the form of a Simple Temporal Problem (STP). In the weighted digraph analogue of an STP, a Simple Temporal Network (STN), each constraint $lb \leq B - A \leq ub$ is encoded as a pair of edges: one $A \rightarrow B$ with weight ub; and one $B \rightarrow A$ with weight -lb. The STP is inconsistent, i.e. the interleaving of plan steps is temporally infeasible, if, between two points, the minimum time exceeds the maximum. Within the digraph, inconsistency appears as negative-cost cycles from a node back to itself.

Inconsistencies cause additional issues in the presence of stochastic durations. As a simple example, if the execution of action A was ordered within the execution of another, B, then even if the expected duration of B exceeds that of A, there is a chance that B could finish first. As we are concerned with plan performance and cost, rather than likelihood of success, we forbid such cases, insisting that in the STNs for the plans we produce, the only permissible cycles of non-infinite length are those from a start/end of an action back to itself via the end/start of the action, respectively. Following (Cushing *et al.* 2007), this subset of problems is known to be sufficient for almost all standard benchmarks[1].

4.2 Outcome Time Estimation

Within our supported subset of partial-order STPs, a Bayesian network can be used to estimate the time by which

[1]Excepting domains containing TILs (or their compilation).

each step will have completed. This can be performed using an adaptation of the approach of (Beaudry *et al.* 2010) to actions with distinct start–end points. For each STP variable $t_i \in [t_0..t_n]$ we add a variable t_i to the Bayesian network, representing the time at which it can occur. Then, for each pair t_i and t_j representing the start and end of action A, respectively, we add a variable $d_{i,j}$, constrained to obey the stochastic duration constraint of A. Finally, we constrain each t_i according to the STN. If step i is the start point of an action, then $t_i = ts(i)$ where:

$$ts(i) = \max\{t_b - w \mid \exists \text{ an STN edge } i \xrightarrow{w} b \ \wedge w \leq 0\}$$

Simply, an action cannot start until its predecessors finish. If step j ends an action beginning at i, then, $t_j = te(j)$:

$$te(j) = \max(ts(j), ts(i) + d_{i,j})$$

Thus, as well as requiring its predecessors to finish, the time since the start of the action must have elapsed. Now, during search, each time we extend the plan, the ordering constraints are updated (as in Section 4.1), and the Bayesian network updated to reflect this extension of the STP.

For each SLA to be met with y% confidence, we must estimate the y^{th} percentiles of the distributions of the time-step variables. We store these time-step estimates for each step. As $F^+(p)$ and $V^{\mathit{eff}}(v)$ refer to the step that provides fact p or the current value of variable v, we can use these to estimate when that fact/value is available. We considered two methods of time-step estimation: sampling, to estimate the distribution, and then taking the appropriate percentile; or, if $y \geq 50\%$, assuming each action takes its expected duration. The former approach gives more accurate estimates; the latter is cheaper, and guaranteed to be admissible, so can therefore be used for pruning. In practice we discovered the most efficient approach is to use expected durations during search; then perform sampling when the goal is achieved to ensure the SLA is met to the required confidence. Note that the definition of a goal state is extended in the presence of uncertainty: for a goal with SLA of x time units $y\%$ of the time, the y^{th} percentile of the earliest time-step from which the goal persists cannot be greater than x.

4.3 A TRPG for Stochastic POPF

In the TRPG of POPF, each fact appears in the fact layer whose timestamp corresponds to when it becomes available (timestamp $F^+(p)$ for propositions and $V^{\mathit{eff}}(v)$ for numeric conditions on v). As a result of this, TRPG actions requiring these facts are delayed to the time corresponding to the earliest point at which the actions could be added to the plan. These modifications preserve the admissibility of the timestamps at which facts become true and thus can be used to prune the search space. They are, in part, responsible for the success of POPF in deterministic domains with deadlines.

We modify this TRPG to support stochastic durations, and goal SLAs. First, the layer in which a fact appears is now derived from the estimated times determined as described in Section 4.2, rather than from the STP. For heuristic purposes, we assume the duration of each action is its expected duration; which, as noted earlier, is optimistic if the percentage criterion on all SLAs is at least 50%. We can then build the TRPG as before, adding successive fact and action layers.

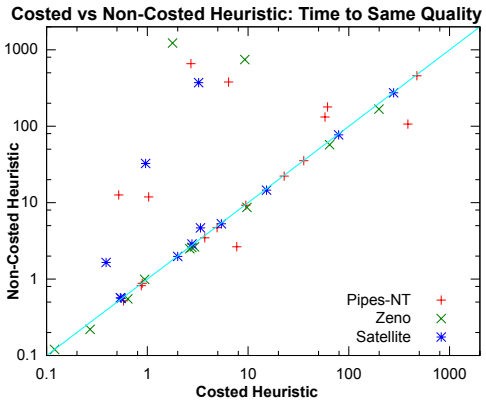

Figure 4: Performance of the Costed Heuristic

In the deterministic case, TRPG expansion terminates successfully if each goal fact g required by time x appears in the TRPG no later than fact layer x. If TRPG expansion does not terminate successfully, the state being evaluated is a dead end. Pruning such dead-end states is completeness-preserving in the stochastic case *iff* the timestamp variable estimates are guaranteed to be admissible. As noted in Section 4.2, we can only strictly make this claim if expected durations are used for these estimates; otherwise, if the upper-bound on the sampling error exceeds the gap between when a goal can be reached in the TRPG and its deadline, the state may incorrectly be deemed to be a dead-end.

5 Cost-Sensitive Planning

Using planning to configure PEPA models is useful only if the decisions made are cost-effective. But, likewise, there is a trade-off between cost and temporal performance: any SLAs must still be met. In this section, we extend stochastic-POPF to any-time search, and modify the heuristic to prune states from which meeting the SLA is too expensive.

5.1 Any-Time Search

In its default configuration, POPF adopts the approach taken by Metric-FF (Hoffmann 2003). It first attempts to find a plan using Enforced Hill Climbing (EHC), a variant of local search; if this fails, it attempts to find a plan using WA*. It returns the first solution found, and then terminates. To extend this approach to perform any-time search, we make a relatively minor modification, searching as follows:

1. Attempt to find a plan using EHC; if one is found, record it; regardless of whether a solution is found, proceed to 2;
2. Attempt to find a plan using WA* search, pruning states whose cost exceeds that of the best solution found thus far; recording any new best plans; and terminating only when the search space has been exhausted.

This search approach is similar to that taken by MIPS-XXL (Edelkamp *et al.* 2006), where WA* is used repeatedly to find solutions bounded to be better than the last found. The key differences are the inclusion of an EHC phase, to prioritise finding *a* solution; and not restarting WA* from the initial state when each new best solution is found. We also modify the heuristic, to prune states using admissible cost and time bounds: this is not done in MIPS-XXL.

5.2 A Cost-Sensitive TRPG Heuristic

In this section we improve the stochastic TRPG of Section 4.3 to allow stronger cost-based pruning. We aim to exclude actions from the TRPG whose application would result in a plan that is more expensive than the previous solution, and in doing so discover earlier when SLAs cannot be met within a cost bound. We consider PDDL 2.1 metric cost functions, specified over the values of the state variables in the goal state reached. We refer to the variables in the cost function as *metric-tracking variables* (MTVs). For our heuristic modification, we consider a subset of these:

Definition 5.1 — Simple Metric-Tracking Variables A vector of variables $\mathbf{v}$ with associated weights $\mathbf{w}$ are simple metric tracking variables *iff*:

1. The plan metric M can be written as `minimize` $\mathbf{v}.\mathbf{w}$ (where . denotes the vector dot-product);
2. Each $v \in \mathbf{v}$ holds a known value in the initial state, and each coefficient in $\mathbf{w}$ is positive;
3. The effects on all $v \in \mathbf{v}$ are to increase one or more such v by a constant (i.e. the effect does not depend on the values of other state variables, or the duration of the action);
4. No $v \in \mathbf{v}$ appears as a condition on an action; but there may optionally be a goal of the form $\mathbf{v}.\mathbf{w} < ub$.

Following 3 we can define the metric cost for each action, $cost(a)$, as the weighted sum of its increase effects on the variables in $\mathbf{v}$. From 4, we can insist that if we already have a solution with metric value m, new solutions have lower cost, i.e. $ub = m$. This is effectively implicitly adding a new goal to the problem requiring a lower metric value.

In its standard form, the metric RPG heuristic disregards effects on simple MTVs when expanding the RPG. If $\mathbf{v}.\mathbf{w} < ub$ in the state evaluated, effects on any $\mathbf{v}$ (which necessarily worsen plan cost) are relaxed, i.e. ignored. Thus, if there is no path to the goal without exceeding ub, this only becomes apparent when ub has already been reached before heuristic evaluation; from which point the quality goal cannot be met.

The planner Sapa (Do & Kambhampati 2003) introduced a technique for admissibly estimating the costs of facts based on the costs of the the actions that add them, and that of their preconditions. Using $pcost(p,t)$ to denote the cost of fact p in fact layer t, if snap-action a in action layer t adds f, it is reachable at the next layer $t+\epsilon$ with cost:

$$pcost(f, t+\epsilon) = cost(a) + \max_{p \in pre(a)} pcost(p,t)$$

The lowest cost way to achieve f at layer $t+\epsilon$ is the minimum across all such $pcost(f, t+\epsilon)$ values for actions adding f in t. In Sapa, these estimates serve to favour relaxed plans with lower-cost actions. Here, we use the cost estimates to delay the point at which actions appear in the TRPG to the layer in which their effects can be obtained without necessarily exceeding the incumbent bound on solution cost, ub. Thus, the earliest action layer t in which action a can appear is after a fact layer t satisfying its preconditions, where:

$$cost(a) + \max_{p \in pre(a)} pcost(p,t) < ub$$

If every goal can be reached with cost $< ub$, a solution is extracted from the TRPG, using the relevant achievers. To refine our cost estimates further, we consider goals with *direct-achievement costs*: goals that never appear in action preconditions; and are added by actions with one add effect (the goal fact), and numeric effects only on simple MTVs. For the subset of such goals G' not true in the state being evaluated, the minimum cost of achievement is:

$$\sum_{g \in G'} \min_{g \in add(a)} cost(a)$$

The cost of achieving all goals by layer $t+\epsilon$ is then at least ($g^\dagger$ denotes the goal in G' with the largest $pcost$ value):

$$pcost(g^\dagger, t+\epsilon) + \sum_{g \in G', g \neq g^\dagger} \min_{g \in add(a)} cost(a)$$

This is an admissible estimate of the cost of reaching G, using actions applicable by TRPG layer t. (Proof sketch: reduction to additive h^{max} (Haslum, Bonet, & Geffner 2005), one operator partition O_i for each $g_i \in G$ with direct-achievement costs (containing operators adding g_i); one operator partition O' containing any other operators.)

This exploitation of costs in the TRPG leads to two types of state pruning. First, if reaching a goal can be seen to require excessive cost, it never appears in the TRPG, and the state will be considered a dead-end; whereas previously, search would continue with a heuristic estimate based on a too-expensive relaxed plan. Second, delaying facts to later layers pushes goals closer to their SLA deadlines. Thus, states are pruned where goals are reachable, but the time to reach them with a sufficiently low cost exceeds the SLA.

6 Evaluation

Our evaluation considers the use of stochastic-POPF on planning benchmarks, and then the integration with PEPA.

6.1 Evaluation on Costed Planning Domains

To determine whether the heuristic in stochastic-POPF is effective, we compare to the non-cost-modified heuristic. Variants of three benchmark domains were used (using competition problem sets), with limits of 1.5GB of RAM and 1800s of CPU, recording in each case the quality of the best solution found, and CPU time taken to reach that solution. Enabling the cost modifications, we ran stochastic-POPF again, recording the time taken to find a solution of equivalent (or better) quality than that found by the control. The results are shown in Figure 4. As can be seen, the new heuristic can vastly improve performance. Cases where it did not, correspond to large problems, where finding any solution is difficult; so optimisation was not possible with the approach used.

6.2 Combination with PEPA

We now evaluate the integration of planning and PEPA to make cost-effective configuration decisions. We consider two models. First, our running example, GetLoan, described in Section 2.3. We can choose whether to invest in each step — slow, medium or fast — with the slow rate carrying zero cost, and medium and fast carrying increasing costs but reducing the transition's duration by 20% and 40% respectively. The customer must receive a response in 5s, 80% of the time. Second, we consider a retail scenario typical of pharmacy suppliers or take-away food shops, where the

investment choice is in staffing levels — how many times system components should be duplicated — rather than investment changing the rates directly.

GetLoan, in its most basic form — with a single customer, broker and lender — is a linear model, so can be converted to PDDL: each transition maps to one action; and the plan directly maps to a PEPA model with the desired performance. A solution to this problem (having exhausted the search space and thus shown optimality) is obtained in 1s; automated configuration of even such small systems is a breakthrough in the PEPA community.

More challenging is the case with contention: 1000 customers and brokers, but only 100 lenders. A model of this size is beyond the capabilities of planning, but within those of PEPA. We employ our iterative approach: the planner considers a single customer, and action durations are set according to PEPA sensitivity analysis, as discussed in Section 3.3. As might be expected, the action affected by 1000:100 broker–lender contention is *transferToLender*, the broker–lender network transfer step shown in Figure 1. Its duration increases because of the implicit queue for lenders. Under the contention, two investment choices have a notable impact on the time of this transfer: network latency, and the time lenders spend processing queries. Thus, for each pair of investment choices available for this network and the lender, we determine the average time for the broker–lender step in the PEPA model, this becomes the mean of the (exponentially distributed) action duration of broker–lender in the planning model for each upgrade pair.

To converge on a suitable model configuration, we follow the process set out in Section 3.3: alternating between duration approximation for different investment choices, using sensitivity analyses; and using the planner to decide which investment options to use to meet the 5s/80% SLA. The investment decisions made by the planner then set the rates in the PEPA model, and the process repeats until the PEPA model, when validated, meets the SLA. In this situation, the process alternates three times; each call to the planner takes around 1s, and sensitivity analysis using PEPA taking between $<$ 1s and 10s, depending on the rates used. Inspecting the action durations obtained by sensitivity analysis, prior to the solution validating, we observe that the duration of *transferToLender* is optimistic, increasing each time the planner is called until there is only a small gap between the time taken for *transferToLender* in the solution model, and that used by the planner.

Moving on to our second domain, the structure of the model is as follows. Over the course of an hour, 20 customers make orders online (SLA: delivery 60 mins, 70%), processed by a member of back-room staff, a packer, and then be delivered. A further 20 make orders in person (SLA: items ready 30 mins, 70%), processed by a cashier and a member of back-room staff. The challenge is to minimise staffing levels to meet the SLAs, given the hourly wage for staff and the (fixed) rates at which each person performs each task. The planning model consists of two essentially separate, linear problems: one of each class of customer. To reflect the contention over staff, the durations for each step at given staffing levels are derived from sensitivity analyses. Thus, even though the plan does not explicitly coordinate the activities of the two customer classes, or model all 20, these are approximated in action durations. Three lots of sensitivity analyses (40s each) and three calls to the planner ($\sim$1s each) happen before convergence. The resulting plan meets the respective SLAs in 27.3 min/52.5 min with cost 84 (the most expensive solution has cost 156). When looking at the intermediate solutions, and the durations used, the key contention in the system shifts as staffing levels change. As each sensitivity analysis is based on the rate settings from the last plan (or nominal values, for the initial analyses), the effect of changing a given staffing level parameter differs at each iteration. One simple example is that if there is one packer, increasing the number of delivery staff barely reduces the time to deliver an order, as delivery staff are not the bottleneck: they are starved of packed orders to deliver.

Acknowledgements

This work was supported by SICSA, and EPSRC fellowship EP/H029001/1.

References

Beaudry, E.; Kabanza, F.; and Michaud, F. 2010. Planning witn concurrency under resources and time uncertainty. In *ECAI '10*.

Bocchi, L.; Fiadeiro, J.; Gilmore, S.; Abreu, J.; Solanki, M.; and Vankayala, V. 2009. A Formal Approach to Modelling Time Properties of Service-Oriented Systems. http://www.scientificcommons.org/53566945.

Buffet, O., and Aberdeen, D. 2009. The factored policy-gradient planner. *Artificial Intelligence* 173(5-6):722–747.

Coles, A. I.; Fox, M.; Long, D.; and Smith, A. J. 2008. Planning with problems requiring temporal coordination. In *AAAI 08*.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *ICAPS*.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. 2007. When is temporal planning *really* temporal planning? In *IJCAI '07*, 1852–1859.

Do, M. B., and Kambhampati, S. 2003. Sapa: Multi-objective Heuristic Metric Temporal Planner. *JAIR* 20:155–194.

Edelkamp, S.; Jabbar, S.; and Nazih, M. 2006. Large-Scale Optimal PDDL3 Planning with MIPS-XXL. In *IPC5 booklet, ICAPS*.

Fox, M., and Long, D. 2003. PDDL2.1: An Extension of PDDL for Expressing Temporal Planning Domains. *JAIR* 20:61–124.

Fox, M.; Long, D.; and Halsey, K. 2004. An investigation into the expressive power of PDDL2.1. In *ECAI'04*.

Gerevini, A. E.; Long, D.; Haslum, P.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic Planning in the Fifth International Planning Competition: PDDL3 and Experimental Evaluation of the Planners. *AIJ*.

Gerevini, A.; Saetti, A.; and Serina, I. 2006. An Approach to Temporal Planning and Scheduling in Domains with Predictable Exogenous Events. *JAIR* 25:187–231.

Gerevini, A.; Saetti, A.; and Serina, I. 2010. Temporal planning with problems requiring concurrency through action graphs and local search. In *ICAPS*.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New Admissible Heuristics for Domain-Independent Planning. In *Proc. AAAI*, 1343–1348.

Hillston, J. 1996. *A Compositional Approach to Performance Modelling*. Cambridge University Press.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating Ignoring Delete Lists to Numeric State Variables. *JAIR* 20.

Kavuluri, B., and Senthil, U. 2004. Timed initial literals using sapa. In *IPC 4 Booklet, ICAPS 04*.

Little, I.; Aberdeen, D.; and Thiébaux, S. 2005. Prottle: A probabilistic temporal planner. In *AAAI05*, 1181–1186.

Mausam, and Weld, D. S. 2008. Planning with Durative Actions in Stochastic Domains. *JAIR* 31:38–82.

Rachelson, E.; Quesnel, G.; Garcia, F.; and Fabiani, P. 2008. A simulation-based approach for solving temporal markov problems. In *ECAI '08*.

Younes, H., and Simmons, R. 2004. Policy generation for continuous-time stochastic domains with concurrency. In *ICAPS*.