TADA: Transition Aligned Domain Analysis

J. Benton and **Kartik Talamadupula** and **Subbarao Kambhampati**
Department of Computer Science
Arizona State University
Tempe, AZ 85287 USA
{j.benton, krt, rao}@asu.edu

Abstract

In this paper we introduce a planner called *TADA* (Transition Aligned Domain Analysis). *TADA* is a heuristic temporal and net benefit planner that enables planning with multi-valued fluents. These multi-valued fluents, known in the language definition as *object-fluents*, are an addition to the existing fragment of PDDL timed to coincide with the sixth edition of the International Planning Competition in 2008 (IPC-2008). Our planner handles this extension, as well as certain other changes to the existing fragment of PDDL, and compiles them into a domain transition graph (DTG) for each variable. The heuristic uses the DTGs thus generated from the domain definition to estimate a heuristic value on the distance of the goal from the current state.

Introduction

We introduce *TADA* (Transition Aligned Domain Analysis), a heuristic temporal and net benefit planner that enables planning with multi-valued fluents. Our heuristic represents each variable in the given planning problem as a Domain Transition Graph (Helmert 2006). The domain transition graphs (DTGs) of variables that appear in the goal are solved in order to produce a heuristic estimate. Other previous work that uses this approach includes (Benton, van den Briel, and Kambhampati 2007), (van den Briel et al. 2007) and (Helmert, Haslum, and Hoffmann 2007).

The algorithm that solves the DTGs is designed to facilitate the interleaving of actions in a relaxed plan that is built along with the generation of the heuristic estimate. Action interleaving is a useful feature that previous methods have largely ignored. Our heuristic tries to preserve the interleaving of actions in an effort to return better estimates. In this sense, our heuristic is quite similar to the planning methodology used in (Chen, Huang, and Zhang 2008); however, while they are constrained to performing exact plan synthesis on DTGs, we follow a relaxed approach by virtue of using DTGs *only* to compute our heuristic.

Previous work has also focussed mostly on the values that a heuristic generates; the actions used by the heuristic have been used as part of the actual plan π only nominally – for example, as *helpful* actions (Hoffmann and Nebel 2001) (Helmert 2006). We present the first heuristic that can use interleaved actions as part of a lookahead strategy to enable faster plan synthesis. We use the relaxed plan R generated by our heuristic to guide a lookahead strategy similar to the one used in (Vidal 2004). This strategy may greatly reduce the time needed to find a satisficing path from the initial problem configuration I to some subset S_g of the goal configuration G.

Interleaving of actions is also useful when planning with durative actions, which our planner supports. We believe that being able to interleave effectively may lead to the synthesis of better makespan estimates that closely mimic $h^\star$, the perfect heuristic function. This is a valuable property, especially in problems with durative actions, since actions may execute concurrently with each other and may require to be interleaved to produce plans of good quality. Better estimates on the makespan will enable efficient planning when deadline goals (which can also be supported by our planner) are present in the problem formulation.

The paper is organized as follows: we begin with some background on using domain transition graphs to extract good heuristics for planning. We then detail the changes to PDDL that play a role in the design of the planner in general and the heuristic in particular; this is followed by the formulation of the problem. The general heuristic idea is then presented, followed by a specialization for domains and problems with durative actions. We conclude with a discussion on extensions including a lookahead strategy and other miscellaneous issues.

Background

Many of the existing domain-independent planning systems rely on good heuristics to guide their search algorithm. This guidance takes varied forms; for example, the pruning of unpromising states and/or the selection of the next node for expansion from the search queue.

In this sense, although the heuristic guiding the search often makes use of the factored representation of the domain provided, the search algorithm itself is only concerned with a state-space representation of the problem. The importance of heuristics that can accurately model the perfect heuristic $h^\star$ cannot be overstated, since any efficiency that the planner accrues will be directly related to the amount of unnecessary state expansion that the search algorithm can avoid while searching for the shortest path to a goal.

Our heuristic makes use of Domain Transition Graphs (DTGs) to represent the problem $\mathcal{P}$ under consideration.

Each DTG d_v may be viewed as the graphical representation of a given variable v, where v could either be a ground predicate or a ground object-fluent. The vertices $n_i \in d_v$ represent the values that v can take as defined by the objects present in the problem (the *domain* of v).

The transitions $t_i \in d_v$ reflect the ways in which v can be set to a corresponding value. A transition t_1 between two vertices $n_1, n_2 \in d_v$ is created if there exists some (ground) operator in $\mathcal{P}$ such that one of its effects e enables a change in the value of v from the value represented by n_1 to that held in n_2. The operator op that provides e is associated with the transition thus created. Each t_i may have a weight associated with choosing it – this weight is determined by the planning domain $\mathcal{D}$ – and could be of type unit cost, the cost of *op*, or the duration associated with executing *op*.

Transitions may have causal dependencies on the values of certain other variables. These causal dependencies must be satisfied before a transition may be taken, and are a direct result of the *causal cascade* that is inherent in planning (as against scheduling, for example). The particular causal dependencies in question may be determined from the *causal graph* $CG_{\mathcal{D}}$ of $\mathcal{D}$. $CG_{\mathcal{D}}$ may be built by observing the preconditions and effects of all the operators $op_i \in \mathcal{D}$. A variable P is causally dependent on another variable Q if and only if there is some op_i with P listed among its effects and Q listed as either a condition or effect.

Initially, only the top-level goals G_s are available. These goals may be used to create sub-goals on the variables $v_i \dots v_n$ that appear in G_s and the DTGs $d_i \dots d_n$ that represent those variables. A goal on d_i is the value that it must be set to from its current value. As we begin solving these *goal* DTGs, further sub-goals are created due to the causal dependencies in $CG_{\mathcal{D}}$.

Problem Formulation

IPC-2008 features a new version of the Planning Domain Definition Language (PDDL) (McDermott 1998). Some of the changes that have a significant impact on the design of our planner's components are described in detail below, along with their implications for those components. We follow this with a discussion on the formulation of a given problem $\mathcal{P}$ into a representational form that uses DTGs.

The major extension in PDDL 3.1 is the introduction of state variables which map to a finite set of values. To this end, object-fluents, which are analogous to numeric fluents in PDDL 2.1 (Fox and Long 2003), are added to the language. This enables objects in the domain to be referred to as relational entities, and is equivalent to introducing function symbols into a first-order knowledge base. With the use of these multi-valued (object) fluents, any arbitrary combination of functions may be used in the domain and problem description, as long as the resolvent is an object that exists in the problem definition.

Another modification to the existing PDDL that affects the sequential and net benefit tracks of IPC-2008 pertains to action costs. The quality of the plans returned by the planner is judged on factors that are determined by the action costs – in the sequential tracks, the plan must minimize the sum of the costs of the actions; in the net benefit tracks, the returned plan must maximize net benefit (Do, Helmert, and Refanidis 2008). This change is restricted specifically to IPC-2008. In the next section, we discuss (among other things) the importance of action costs as the heuristic's minimization metric in problems without durative actions.

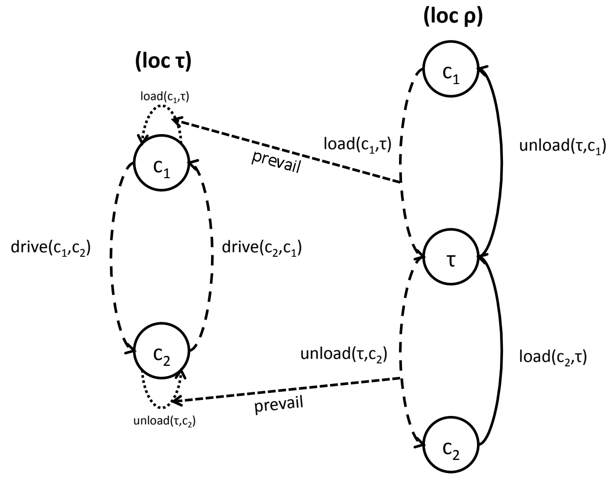

Figure 1: A simple logistics example illustrating the use of prevails. τ represents a truck; ρ a package; c_1 and c_2 are cities.

We contend that our heuristic can handle the change in representation brought on by the use of multi-valued fluents naturally, since DTGs are agnostic to the type of values that a state variable may map to, whether binary or multi-valued. This ability to represent non-binary state variables is one of the main motivations behind using DTGs to extract a relaxed plan and a heuristic estimate. The second change is accommodated by initializing the weights of the transitions in each DTG to the cost of the corresponding operator.

We now describe the formulation of a specific problem $\mathcal{P}$ in terms of a set of DTGs D. As specified in the background discussion, each DTG represents a particular variable and the various transitions among the values that it can take, contingent on conditions on other values and transitions in different DTGs. These dependencies are determined by the causal graph $CG_{\mathcal{D}}$ of the domain $\mathcal{D}$.

Given $\mathcal{P}$ and $\mathcal{D}$, we create a DTG $d_i \in D$ for each variable $v_i \in \mathcal{P}$. d_i is populated with vertices $n_1 \dots n_k$ for each value $\lambda_1 \dots \lambda_k \in domain(v_i)$ respectively. A transition $t_i \in d_i$ from the vertex n_i to n_j corresponds to a fully grounded operator $op(t_i) \in \mathcal{D}$ that changes the value of v_i from λ_i to λ_j. Formally, the set of transitions $T \in d_i$ can be described as:

$$T = \{t_i \mid n_i \xrightarrow{t_i} n_j \Leftrightarrow \lambda_i \overset{op(t_i)}{\rightarrow} \lambda_j\}.$$

When d_i is constructed, there is no implicit ordering to T. However, any path $p \in d_i$ will necessarily be a list that obeys the requirement $p \subseteq T$. Each transition $t \in T$ is also assigned a *weight* that is associated with taking t as part of a path p. This weight is determined by $op(t)$.

Algorithm 1: *dequeue* implements the removal of goal DTGs to be solved. h is the overall heuristic estimate, and R is the partial plan produced.

```
  Input: DTG Set $D$, Goal DTG Set $D_g$.
1 $h := 0$;
2 $R := \phi$;
3 forall $d_g \in D_g$ do
4     $n_g$ := Vertex containing goal value of $d_g$;
5     $(h, R) := (h, R) + solve(d_g, n_g, D, R)$;
6 end
7 return $h$, $R$
```

Transitions may be associated with three kinds of causal interactions – *prevails*, *linked effects* and *conditional effects*. **Prevails** are causal links from a transition $t \in d_1$ to a vertex $n \in d_2$ (where d_1 and d_2 are DTGs) such that t is applicable if and only if the current value of d_2 is the value stored in n. As an illustration, consider a logistics-style domain where a truck τ may be in one of two locations c_1 and c_2; and a package ρ may be in any of three locations c_1, c_2 or τ (Figure 1). ρ is initially at location c_1, and the goal is to have ρ at c_2. A high-level plan for such a scenario (involving just the package ρ) is to load ρ into t from c_1, and unload ρ from t to c_2. This can be represented as a plan π, and as a path p in d_1.

$$\pi = \langle load(c_1, \tau), unload(\tau, c_2) \rangle$$
$$p = \langle t_{load-c_1}, t_{unload-c_2} \rangle$$

However, loading into τ from c_1 is dependent on the requirement that τ be *at* c_1; similarly, unloading from τ into c_2 depends on τ being at c_2. These two dependencies are *prevails* that must be respectively satisfied for the respective transitions t_{load-c_1} and $t_{unload-c_2}$ to be taken. **Conditional effects** on DTGs may be seen as a special case of prevails where the DTGs d_1 and d_2 are the same. For a detailed definition and handling of conditional effects, the reader is directed to (Pednault 1989). **Linked effects** are transitions on DTGs d_1 and d_2 such that $t_1 \in d_1$, $t_2 \in d_2$ and $op(t_1) = op(t_2)$. They are caused by operators that have effects on more than one variable $v_i \in \mathcal{D}$. Our heuristic ignores both conditional and linked effects; hence we do not discuss them in detail.

Heuristic

We begin by describing h_t, the version of the heuristic that deals with action costs, but not durative actions. The DTG set D is set up using the domain description $\mathcal{D}$ and the problem description $\mathcal{P}$, as discussed in the previous section. The transitions $t \in d$ in each DTG $d \in D$ have the respective operator's costs associated with their weights; that is, $weight_t = cost(op(t))$. All DTGs that correspond to variables that appear in the goal description $v_g \in G$ are grouped in a set $D_g \subseteq D$. During this process, the vertex n_g bearing the *goal-value* in each $d_g \in D_g$ is marked as well. The goal-value is the value assigned to the variable v_g in G.

h_t provides guidance to the search in two ways – by providing a heuristic estimate h, which is a numeric estimate on the cost to reach G from the initial specification I of $\mathcal{P}$; and by providing a relaxed-plan R that converts I to some state that is a subset of G. Formally,

$$R = \{op(t_1) \ldots op(t_n) \mid I \rightarrow op(t_1) \ldots op(t_n) \rightarrow G_s \subseteq G\}.$$

Initially, as shown in the description of *dequeue* (Algorithm 1), $h = 0$ and $R = \phi$. *dequeue* iterates through D_g and calls *solve* (Algorithm 2) on each $d_g \in D_g$. Currently, we impose no ordering on the dequeuing of DTGs from D_g; however, it may be possible to obtain better performance by making the causal graph of the domain $CG_{\mathcal{D}}$ acyclic through a cycle-breaking process, and then dequeuing the DTGs in decreasing order of causal dependency, as described in (Helmert 2006).

The *solve* routine tries to find the shortest path p_g in a given DTG d_s from its initial vertex n_c to the goal vertex n_g. To find p_g, Dijkstra's all-pairs shortest path algorithm is used, with the vertex pair $\langle n_c, n_g \rangle$ and the graph d_s as input. If there is *no* path from n_c to n_g, we update the the heuristic estimate h_{est} with an arbitrary value. This number psi_{est} denotes the cost of a *phantom path* between n_c and n_g and is designed such that all legitimate paths $p_g : n_c \rightsquigarrow n_g$ have cost strictly less than it. That is,

$$\forall p \mid n_c \overset{p}{\rightsquigarrow} n_g, cost(p) < \psi_{est}.$$

Algorithm 2: *solve* updates the heuristic and plan estimates by solving a given DTG. ψ_{est} is an arbitrary value that provides the cost of a phantom path between 2 vertices.

```
   Input: DTG $d_s$, Goal Vertex $n_g$, DTG Set $D$, Partial
          Plan $R$.
1  $n_c$ := Vertex containing current value of $d_s$;
2  $p_g := dijkstra(n_c, n_g, d_s)$;
3  if $p_g$ is $\phi$ then
4      $h_{est} := h_{est} + \psi_{est}$;
5  else
6      $T$ := {Transition $t_i \in p_g$ such that $t_i < t_j$ if $i < j$};
7      $h_{est} := h_{est} + length(p_g)$;
8      for $i := 1 \ldots |T|$ do
9          $resolve(t_i, D, R)$;
10         $R := R + op(t_i)$;
11     end
12 end
13 Set current vertex of $d_s$ to $n_g$;
14 return $h_{est}, R$
```

Algorithm 3: *resolve* solves all the prevails l_i on a Transition t.

```
  Input: Transition $t$, DTG Set $D$, Partial Plan $R$.
1 $L := \{l \mid l$ is a prevail on $t\}$;
2 forall $l \in L$ do
3     $d_c$ := DTG associated with $l$;
4     $n_{dg}$ := Vertex containing goal value of $d_c$;
5     $solve(d_c, n_{dg}, D, R)$;
6 end
7 return
```

Phantom path refers to the fact that such a path does not exist in actuality, but is merely being added to ensure that a heuristic value is returned. The addition of this phantom path also ensures that the heuristic does not indulge in expensive backtracking at any point during its search. It is important to note that this addition is possible only due to the fact that we are estimating a heuristic value, and unlike (Chen, Huang, and Zhang 2008) are not doing complete planning on DTGs. The constraint that ψ_{est} – the cost of taking the phantom path – be greater than the cost of any legal path between n_c and n_g is required lest this phantom path be preferred to a *real* path, if there exists one.

If there *is* a path p_g, we make an ordered list T of the transitions $t_i \in p_g$. We also add the length of the path to h_{est}. The $length$ of a path p is defined as the sum of the costs of all the operators associated with the transitions along that path.

$$length(p) = \sum_{i=1}^{|T|} cost(op(t_i));\ t_i \in T.$$

It has been mentioned earlier that transitions have causal dependencies known as prevails associated with them. After compiling an ordered list T of the transitions $t_i \in p_g$, $resolve$ (Algorithm 3) is called on each $t_i \in T$. Given a transition t, resolve compiles a set L of the prevails on t. For each prevail condition $l \in L$, the DTG d_c associated with l and the corresponding goal-vertex n_{dg} of that DTG are determined. In order to satisfy l, $solve$ is then called on d_c and n_{dg}. This call to $solve$ introduces recursion into the heuristic algorithm; to completely satisfy all the transitions $t \in T$ along a path p_g that satisfies a goal DTG d_g, we must satisfy (in turn) all DTGs associated with prevails $l \in L$ on each of the transitions t.

Once all the prevails $l \in L$ on t are satisfied (solved), $resolve$ returns control to $solve$, where the operator associated with t $op(t)$ is added to the partial plan R. This addition signifies that the transition t has been *taken*; that is, all prevail conditions on t have been satisfied by previous additions to R and t is hence satisfied. When all the transitions $t \in T$ are thus satisfied, the value of the current vertex of d_s is modified to point to n_g, the goal vertex. Following this, the value of the heuristic estimate h_{est} and the existing partial plan R are returned to $dequeue$, where the current estimates h and R are updated. The combination of Algorithms 1, 2 and 3 outlined above is then repeated, until all the goal DTGs $d_g \in D_g$ have had $solve$ called on them. Succeeding that, the heuristic value estimate h and the partial plan R to achieve that estimate are returned.

Using a Lookahead Strategy

We use the partial plan R generated during the computation of the heuristic to augment our search with a lookahead analysis as described in (Vidal 2004). Succinctly, our search is altered as follows: we run the heuristic h_t on the initial configuration I specified in $\mathcal{P}$. We then apply the partial (relaxed) plan R that is returned as a result of the heuristic computation to I, resulting in a state S_1. This state is known as a *lookahead* state, and it is guaranteed to have a heuristic value that is better than that of I. In fact, if the relaxed plan R were a true plan (i.e., actions in R had no deletes), S_1 *would* be the actual state resulting from the application of R in I. To bias the search towards picking the lookahead states as often as applicable, we use an enforced hill-climbing approach.

We repeat this strategy until we reach some state S_g such that $S_g \subseteq G$. There is a possibility that this strategy may never reach such a state; hence we fix a cut-off time of 10 minutes for this lookahead strategy. If S_g is not found within 10 minutes, we switch to a traditional best-first search guided by the heuristic values h_{est} for the remaining 20 minutes. Even if a state S_g *is* found, we spend the remaining time trying to improve the existing plan R_p, where

$$R_p = R_1 \dots R_n \mid I \rightarrow R_1 \dots R_n \rightarrow S_g \subseteq G,$$

with respect to the cost of the entire plan.

Support for Soft Deadline Goals

Deadline goals are goals that must be achieved before a certain *deadline* t_d; else the plan π fails. **Soft deadline goals** are goals with deadlines that may be violated. If a soft deadline goal g_d is achieved before its deadline t_d, the full utility r_d associated with achieving g_d is awarded to the planner. If t_d is violated, the plan π still holds, but none of the utility associated with achieving g_d is accrued. g_d is a **soft deadline goal with diminishing utility** if the reward for achieving g_d exhibits the following behavior: if g_d is achieved by time t_d, the full utility r_d is awarded to the planner; however, even if t_d is violated, some reward $r_d^\star < r_d$ still accrues. $r_d^\star$ diminishes to 0 as a function of the time t_p that has passed since the violation of t_d; that is,

$$r_d^\star = min(0,\ r_d - k(t_c - t_d)),$$

where t_c is the current time and k is an arbitrary domain or problem-dependent constant.

In order to support soft deadline goals with diminishing utility, it is imperative that the makespan estimator h_{dur} be *accurate*. Accuracy is required so that we may evaluate how feasible a given deadline goal g_d is, given the current estimate on the makespan h_m. At any given instant, h_{dur} should return an accurate h_m such that we can estimate within reasonable bounds the remaining utility $r_d^\star$. In domains where it may be possible to interleave actions and exhibit concurrency, h_{dur}'s accuracy improves if it also takes the action interleaving into account. In the following section, we describe an extension to our representational scheme that enables the handling of interleaved actions.

Handling Durative Actions

The heuristic described thus far may be extended to return makespan estimates for temporal planning problems, where operators may execute concurrently in an interleaved fashion. Some changes are needed to the setup of the DTGs $d \in D$ – primary among these is that the transitions $t \in d$ no longer carry the costs of the operators $op(t)$ associated with them, but the durations of the same operators as their weights $weight_t$. Hence the length of a path p in a DTG d

may now be considered the sum of the durations of all transitions $t \in p$;

$$length(p) = \sum_{i=1}^{|T|} dur(op(t_i));\ t_i \in T.$$

Since the temporal track of IPC-2008 only considers the minimization of the makespan as its objective, the search is better guided by a heuristic that returns estimates on the makespan.

Durative actions introduce the possibility of plans that may have two or more actions executing concurrency at a given instant. In previous sections, we introduced the interleaving of actions as an essential property that can lead to better makespan estimates as well as more executable relaxed plans, and motivated the need for these properties. However, there remains an important change that must be made to the heuristic before it can handle concurrently executing durative actions. h_t, the version of the heuristic previously described, works with DTGs that admit only *contiguous transitions* between two connected vertices n_1 and n_2 in a DTG d. A **contiguous transition** t_c from n_1 to n_2 is a transition that, once taken, instantly changes the current value of d to the value stored in n_2.

However, given the definition of durative actions in PDDL (Fox and Long 2003), effects that change the value of a variable may take place either at the beginning, throughout the execution, or the end of an action a. An effect e_a that take place *over all* may be considered a special case of an *at end* effect, since the result of e_a is considered unknown until the end of a. It is possible to interleave actions only if the start and end points of some action may be separated by other actions' starts and ends. The representation of the problem $\mathcal{P}$ as a set of DTGs D needs to be extended to take this requirement into account.

We describe this extension in terms of the existing structure of the DTGs, as outlined earlier. Each DTG $d \in D$ is augmented with an additional transition t_e for every existing transition t – the first transition in the sequence, now called t_s, describes the *at start* portion of the effect; the other, t_e, describes the *at end* part. The duration of the operator associated with the erstwhile t $dur(op(t))$ is attached entirely to the *at start* transition t_s. This is done to deal with cases where there may be more than one set of transitions $\langle t_s, t_e \rangle$ between two connected vertices $n_1, n_2 \in d$ — in such a scenario, the heuristic must be allowed to choose the transition with the lowest makespan and avoid costly backtracking.

Once the heuristic chooses a list of transitions

$$T = \{t_1 \dots t_n \mid n_c \rightarrow t_1 \dots t_n \rightarrow n_g\},$$

where n_c and n_g are the initial and goal vertices on the DTG d, we follow the h_t algorithm to resolve prevails as described in the previous section. The only change is the requirement that a start transition t_s be already active (taken) for an end transition t_e to be availed. This requirement ensures that there are no *orphaned transitions*. An **orphaned transition** is an *at end* transition t_{1e} such that there is no corresponding *at start* transition t_{1s} in the partial-plan R.

To alleviate this issue, we maintain a counter Θ for each (ground) operator. An operator o's counter Θ_o increments when a transition t^s_o is added to R, such that $op(t^s_o) = o$ *and* t^s_o is an *at start* transition. Conversely, Θ_o *decrements* when a transition t^e_o is added to R, such that $op(t^e_o) = o$ and t^e_o is an *at end* transition.

Conclusion

In this paper, we have described a planning system called *TADA* that utilizes a novel heuristic based on domain transitions graphs in order to search for a plan that minimizes either a cost or a makespan measure. We have motivated the use of domain transition graphs as an analytical tool in domains where variables may be multi-valued, and contended that this representation may help in the extraction of better heuristic estimates.

We have shown that the actions chosen by the heuristic in order to arrive at a value estimate may be stored as a relaxed plan, and used to guide a search lookahead strategy. This strategy is discussed on the premise that it may lead to faster synthesis of satisficing plans; the rest of the planning time can then be spent on improving the metric specified by the problem.

The achievement of soft deadline goals of diminishing utility has been discussed, and we have shown how our heuristic may be extended to deal with temporal actions such that the heuristic returns accurate estimates on the makespan. Good makespan estimates are required in order to deduce a deadline goal's utility at any given point during the plan currently created.

We hope to be able to complete most of the proposed ideas to enable the planner to enter into the sequential, net-benefit and temporal tracks of IPC-2008. We also plan to add extensions such as a causal graph analysis for goal ordering and a problem decomposition approach for faster analysis of problems and domains.

References

Benton, J.; van den Briel, M. H. L.; and Kambhampati, S. 2007. A hybrid linear programming and relaxed plan heuristic for partial satisfaction planning problems. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS-2007).*

Chen, Y.; Huang, R.; and Zhang, W. 2008. Fast Planning by Search in Domain Transition Graphs. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-2008).*

Do, M.; Helmert, M.; and Refanidis, I. 2008. The Sixth International Planning Competition.

Fox, M., and Long, D. 2003. PDDL2.1: An extension of PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible Abstraction Heuristics for Optimal Sequential Planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS-2007).*

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artifical Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

McDermott, D. 1998. PDDL — The Planning Domain Definition Language.

Pednault, E. P. D. 1989. ADL: exploring the middle ground between strips and the situation calculus. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, 324–332. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

van den Briel, M.; Benton, J.; Kambhampati, S.; and Vossen, T. 2007. An LP-based heuristic for optimal planning. In *Proceedings of the International Conference of Principles and Practice of Constraint Programming (CP-2007)*.

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-2004)*.

Appendix

This section contains additional information related to CSE574 (Planning & Learning) that may not form the body of a report on the *TADA* planning system and its associated heuristic. This section is **specific to the class report**.

Progress and Status

At the beginning of the semester, it was hoped that this work would culminate in the submission of a technical paper to the main track of ICAPS-2008 describing the planning system and results generated. However, this was rendered infeasible due to the fact that the planner was incomplete as per the ICAPS-2008 abstract submission deadline. In this sense, the overall initial goal of the project may be seen as unachieved. However, in keeping with the true spirit of this planner in producing solutions to problems that are oversubscribed, we describe the current status of the implementation of this system and a brief timeline.

At present, a parser that reads in the new version of PDDL (v. 3.1) and converts it to the representation required by our heuristic is complete. This parser even handles object-fluents, arguably the hardest addition to the language for this year's competition. The grounding method for the information that is parsed is complete as well.

The heuristic algorithm as described above is nearly complete; however, the exact details of resolving the prevail conditions need to be implemented into the system. Our search strategy will be a combination of lookahead, as described in the paper, and a (default) best-first search in the state-space using the values returned by the heuristic as guidance. The search component is still pending implementation.

We plan to submit the planner to three tracks in IPC-2008. The date for submission of the planner is May 31, 2008. Before that, a four page paper describing the planning system is due on May 15, 2008. We will use most of the information in this report as part of that paper. Additionally, we plan to submit our heuristic idea to the ICAPS 2008 workshop on Oversubscribed Planning & Scheduling.The deadline for this submission is June 18, 2008.

Future Work

The heuristic described above may be extended in numerous ways to improve performance, as also the accuracy of the returned estimate. We describe two extensions here – goal-ordering, and problem decomposition.

Currently, the dequeuing of goal DTGs from the set D_g is performed in no specific order (non-deterministically). It is feasible to incorporate an ordering based on the causal graph of the domain $\mathcal{D}$, as described in (Helmert 2006). This kind of ordering may help in two ways:

1. Since most existing planning domains offer themselves readily to a hierarchical decomposition in the causal direction, the achievement of goals may be made easier if the heuristic searches in a fashion dictated by the causal graph.
2. There may be fewer conflicts between actions taken to satisfy goals or sub-goals, and hence a relaxed-plan might be very close to the actual plan π for the problem. This will greatly improve the performance of the system, especially if a lookahead strategy is used in search.

Problem decomposition is another extension that may be provided. Most planning problems $\mathcal{P}_i \in \mathcal{D}$ are similar in nature for a given $\mathcal{D}$. If an analysis of problems can be performed, literals or conditions that are hard to achieve or satisfy may be identified as *bottlenecks*. Special attention may then be paid to these bottleneck conditions. An analysis of $\mathcal{D}$ and $\mathcal{P}_i \in \mathcal{D}$ may also facilitate the inference of axioms regarding the domain that may greatly reduce the state-space that needs to be considered by the heuristic and the search.

We do not make any claims about the applicability of these two extensions at this point, but it is conceivable that the planning system may be augmented with them and the ensuing results examined for further proof of their effectiveness.