In 9th International Symposium on AI, Robotics, and Automation in Space (iSAIRAS-08)

Automatically Generated Heuristic Guidance for Europa2

Sara Bernardini
Istituto per la Ricerca Scientifica e Tecnologica (IRST)
Via Sommarive 18
38055 Trento, Italy
bernardini@itc.it

David E. Smith
NASA Ames Research Center
Intelligent Systems Division MS 269-2
Moffet Field, CA 94035–1000
desmith@arc.nasa.gov

Abstract

In the last few years, classical planners have achieved impressive results due to the development of problem relaxation techniques for computing distance estimates. In contrast, many large temporal planning systems used for practical applications have not benefitted from these techniques. Instead, these systems rely on careful engineering of the domain knowledge, together with carefully crafted domain-dependent control information. In this paper, we explain some of the characteristics of NASA's EUROPA2 planning system that make it difficult to directly apply the heuristic techniques developed for classical planning. However, we then borrow ideas from some of these methods to develop domain-independent heuristic techniques for EUROPA2. We show some promising initial results concerning their effectiveness.

1. Introduction

Many planners used for space and robotic applications, such as EUROPA [1], ASPEN [2] and IxTeT [3] rely on careful engineering of the domain together with hand-crafted domain-dependent search control information, in order to carefully control the search for plans. Encoding this control knowledge is generally quite difficult and time consuming, and can lead to models that are not very robust to small changes in the domain or in the nature of the problems being solved.

In the last decade, the academic community has demonstrated significant improvements in the performance of domain-independent classical planning systems. Key to this improvement has been the development of domain-independent heuristic techniques for estimating the distance between states and goals. Generally, these techniques rely on automatically generating a relaxed formulation of the planning problem and using a solution of this relaxed problem as a distance estimate. One popular method for doing this is to generate a plangraph [4], extract a relaxed plan from it, and use the cost of this solution as the distance estimate [5]. It would be quite useful and desirable if these control techniques could be applied to the temporal planning systems used for space applications. Unfortunately, there are difficulties involved in doing this: the representation languages for these systems are quite different, allowing much more complex temporal and metric constraints, and the search strategies employed by these systems cannot be characterized as either simple progression or regression. As a result, it is difficult to directly map the techniques from classical planning systems to these application systems.

In this paper, we develop novel domain-independent heuristic guidance techniques for the EUROPA2 planning system. This planner has been used for several NASA mission applications including MAPGEN, the ground-based daily activity planning system for the Mars Exploration Rover mission (MER) [6], and the International Space Station, where it is currently used for power systems management at the Johnson Space Center. The EUROPA2 planner has been shown to be extremely successful in solving complex real-world problems by providing the user with a powerful modeling language as well as a highly customizable solving engine. Nevertheless, EUROPA2 suffers from having little or no effective domain-independent heuristic guidance. Our technique borrows ideas from the work of Haslum and Geffner ([7]), and Helmert ([8]). In particular, we build transition graphs for the different state variables in EUROPA2, and use these graphs to compute distance estimates for choosing and resolving flaws within EUROPA2's plan refinement mechanism.

In order to explain our technique it is necessary to have some understanding of the EUROPA2 planning paradigm and search algorithm. We give a quick overview of the essentials in the next two sections. We then describe our technique for automatically deriving domain-independent heuristic estimates. We conclude by presenting some preliminary experimental results.

2. EUROPA2: Paradigm and modeling language

For EUROPA2, planning domains and problems are described using a declarative modeling language called NDDL (New Domain Definition Language). A ***planning domain*** $\mathcal{D}$ in NDDL is represented by the following elements:

- A set of ***timelines***: $\mathrm{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n\}$, which are essentially variables capturing the evolution of a quantity or component over time

- A set of mutually exclusive ***activities*** associated with each timeline $\mathcal{T}_i$: $\mathcal{A}ct[\mathcal{T}_i] = \{a_1(\vec{x_1}, \delta_1), \ldots, a_n(\vec{x_n}, \delta_n)\}$ where $\vec{x}$ is the vector of the activity's parameters and $\delta = [\delta_{min}, \delta_{max}]$ is a mathematical interval in $\mathbb{N}$ representing the duration of the activity
- A conjunction of *temporal constraints* associated with each activity a_i: $\mathcal{C}[a_i] = c_1 \wedge \ldots \wedge c_n$, where a conjunct c_j can assume one of the following two forms:
  - $c_j = a_i$ `temporal_relation` a_k
    Such a conjunct is called a ***compatibility***. It is a qualitative (`meets`, `met_by`, etc.) or quantitative temporal constraint between the activity a_i and any other activity a_k belonging to either the same timeline $\mathcal{T}$ or to another timeline $\mathcal{T}_h$. The activity a_i is called *master* and the activity a_k is called a *slave*.
  - $c_j =$ `(case` $\gamma = 1 : \mathcal{C}_1^-$`);...;(case` $\gamma = m : \mathcal{C}_m^-$`)`
    A conjunct can also correspond to a choice between different conjunctions of compatibilities $\mathcal{C}_h^-$, where $\mathcal{C}_h^- = \{c_1 \wedge \ldots \wedge c_l\}$ and $c_i = a_i$ `temporal_relation` a_k. The choice between the conjunctions is regulated by the variable γ, which is called the ***guard*** of the case.

A rich set of temporal relationships is permitted in compatibilities, including: *equal*, *meets*, *contains*, *after*, *starts*, *overlaps* and all their inverse relations. These relations are similar to the thirteen temporal relations defined by Allen [9].

The above representation differs from domain descriptions expressed in PDDL [10], the standard specification language used by the academic community to describe classical planning tasks. The main differences are: 1) NDDL uses a variable/value representation (timelines/activities) rather than a propositional representation, and 2) there is no concept of state or action in NDDL, only of activities and constraints between them.

A ***planning problem*** $\mathcal{P}$ for EUROPA2 is represented by a pair $\mathcal{P} = \{\mathrm{H}, \mathcal{I}\}$ where:

- $\mathrm{H} \in \mathbb{N}$ is the end of the ***planning horizon***, meaning that we only care about the behavior of the system with respect to the temporal window $[0, \mathrm{H}]$.
- $\mathcal{I}$ is the ***initial configuration*** represented by a set of activities placed on their corresponding timelines. If we annotate an activity a by a time interval $\tau(a) = [st(a), et(a)]$ (indicating the temporal extent over which a holds), then, for each activity a_i in $\mathcal{I}$, it is possible either to specify the specific position of a_i on the timeline, which basically means fixing the start and end time of $\tau(a_i)$, or to leave a_i floating on the timeline between the origin and the horizon.

The initial configuration $\mathcal{I}$ corresponds to both the initial state and the goal state as they are defined in classical planning. The activities in $\mathcal{I}$ that are placed at the beginning of the horizon correspond to the traditional initial state, while all the others generalize the classical notion of goal since they can be placed not only at the end of the horizon, but also in any other position. The initial configuration is also the initial partial plan that is turned into a final plan by the refinement planning procedure. A final *plan* π is the configuration where all the timelines of the domain are fully covered by contiguous activities from the start to the end of the horizon. Some of the activities are those that appear in the initial configuration, the others are triggered by the applicable compatibilities associated with the initial activities and those that are incrementally added to the plan. In fact, a compatibility c that involves two activities a_i and a_k imposes that, once a_i has been chosen to be part of the plan and placed on its proper timeline $\mathcal{T}$, then an activity a_k must exist in the plan in such a way that the temporal contraint stated by c is satisfied. A plan is *complete* and *consistent* when all the timelines are fully covered by activities, all the temporal contraints involving those activities are satisfied and all the variables are instantiated.

An example. As an illustration of a simple NDDL domain model, consider a rover equipped with a set of instruments to explore a geological site. We model the following subsystems as timelines: *Battery*, *Navigator*, *Controller*, *Instrument*$_1$, ..., *Instrument*$_n$. Each subsystem can only perform certain activities. Let us assume, for example, that *Instrument*$_k$ is a microscopic imager. It can perform one of the following operations: *TakeImg(rock, 1)*, *Place(rock, 3)*, *Stow(2)*, *Unstow(2)* and *Stowed([1, +inf])*. The first activity consists of taking a microscopic picture of a rock at the site and lasts 1 time unit. The other specifications are similar. The constraints that regulate the behavior of the imager are the following: in order to take a picture of a rock, the imager must be first unstowed and then properly positioned in the vicinity of the rock. After taking the picture, the imager can be placed in another position for performing another experiment or can be stowed. Those constraints are expressed by means of the compatibilities. We show just a few of them:

- *Unstow()* `meets` *Place(rock*$_i$*)*
- *Place(rock*$_i$*)* `meets` *TakeImg(rock*$_i$*)*
- `case` $\gamma = 0$: *TakeImg(rock*$_i$*)* `meets` *Stow()*;
  `case` $\gamma = 1$: *TakeImg(rock*$_i$*)* `meets` *Place(rock*$_j$*)*

Note that in this model, we have not modeled or allowed intermediate state between *Unstow*, *Place*, and *TakeImg* operations – something that is not possible in PDDL.

Additional compatibilities for *Instrument*$_k$ govern its interactions with the activities on other timelines. For example, the instrument can take a picture of a rock only if the navigator has already reached that rock and persists in that position while the instrument is taking the picture:

Instrument$_k$*.TakeImg(rock*$_i$*)* `contained_by` *Navigator.At(rock*$_i$*)*.

An initial configuration $\mathcal{I}$ for the rover domain can, for example, specify the level of the battery, the position of the navigator and the status of the instruments at the start of the planning horizon and, furthermore, can require that an image of a particular rock be taken within a certain time interval.

2.1 EUROPA2: search algorithm

The planning algorithm at the core of EUROPA2 can be thought of as an instance of *plan refinement search* [11]; given a domain $\mathcal{D}$ and a problem $\mathcal{P}$, the algorithm starts from the initial configuration $\mathcal{I}$ and incrementally refines it by adding activities to the timelines, ordering those activities and binding variables until a final consistent configuration is found. This algorithm can also be seen as a search in the *space of partial plans* [12], where a ***partial plan*** Π consists of the following elements:

- For each timeline $\mathcal{T} \in \mathcal{D}$, a set of activities $\mathcal{A}ct_\Pi = \{t_1, t_2, \ldots, t_n\}$, which are not necessarily contiguous on time.
- A ***temporal network*** $\mathcal{N}_\Pi$ representing all the start and end times of the activities in the plan and the constraints between them.
- A set of ***flaws*** $\mathcal{F}_\Pi = \{f_1, f_2, \ldots, f_m\}$, where a flaw is an indication of a potential inconsistency in the partial plan. There are three types of flaws:
  - ***Open condition flaws***: They arise when applicable compatibilities are applied, triggering activities as slaves of masters that are already in the plan Π. Those slave activities are enforced to be part of the plan, but they are not yet associated with any timeline. We call them *free activities*.
  - ***Ordering flaws***: They arise anytime an activity is placed on a timeline and an ordering is required for the activity with respect to the other activities already on that timeline.
  - ***Unbound variable flaws***: They arise when variables that have not yet been instantiated appear in the plan Π. Those variables are said to be *unbound*. There are two kinds of unbound variables: parameters of activities that are already in the plan and guards of applicable temporal constraints.

Refining a partial plan means to pick a flaw and resolve it. The process terminates when the set of flaws is empty. Each kind of flaw is resolved in a different way.

- *Resolvers for open condition flaws*

  Flaws corresponding to free activities can be resolved in two ways:
  - ***Merging*** A free activity is merged with a matching activity already in the plan The operation of merging does not result in the addition of any new flaws to the current plan. An activity a is said to match an activity a' if a and a' unify and the temporal constraints involving a are satisfied by a'. Thus, a and a' can be considered the same activity and we do not need to introduce a in the plan. Consequently, the compatibilities associated with a are not fired, because they have been already triggered when a' was introduced in the plan.
  - ***Activation*** We introduce a new activity a in the current plan associating it with the proper timeline, but without choosing a specific time slot for it.The compatibilities associated with a are applied and the subgoal activities resulting from those compatibilities are introduced as free activities. This results in both an ordering flaw, corresponding to the just activated activity, and a number of open condition flaws, corresponding to the new subgoal activities.
- *Resolvers for ordering flaws*

  Once we have decided to place a new activity on a timeline, we need to choose where to put it with respect to the other activities already on that timeline. For this purpose, the temporal constraints involving the new activity are checked against the current temporal network. An ordering flaw is resolved by imposing ordering constraints among activities in such a way that the temporal network remains consistent and all the constraints are satisfied.
- *Resolver for unbound variable flaws*

  Unbound variable flaws are resolved by specifying a value in the domain of the variable. If the variable is a guard, the binding causes the introduction in the current plan of the activities associated with the chosen value.

The basic algorithm in EUROPA2 is a *depth-first search* characterized by ***flaw selection***, ***flaw resolution*** and ***constraint propagation*** steps. Flaw selection identifies which flaw to resolve next. This is not a backtracking point, but, like variable ordering in constraint satisfaction, has a significant impact on the amount of search and backtracking required to find a solution. Flaw resolution deals with resolving a flaw by subsequently trying all the resolution options (activation and merging for open condition flaws, all possible activity orderings for ordering flaws and possible variable bindings for unbound variable flaws). This is a backtracking point because if a resolution option does not work, the algorithm tries another option until all options are exhausted. Operations of plan refinement are interleaved with *constraint propagation* on the constraint network underlying the current partial plan. Contraint propagation is mainly used to test partial plans for consistency, and discovers dead ends, which are either inconsistent partial plans or partial plans with flaws that cannot be resolved. However, it also plays another major role: it provides the algorithm with a *look-ahead capability* that allows it to filter away infeasible flaw resolvers before the algorithm actually commits to them.

2.2 The search control problem

Through a combination of careful domain engineering and the crafting of domain-dependent search control information, a user can customize and control search, flaw selection and flaw resolution in EUROPA2. However, this process is painful, time consuming, and often leads to models that are not robust to further enhancements or changes. If EUROPA2 is run in the absence of domain-dependent heuristics, it inevitably experiences serious control problems. Plans are not found within a reasonable amount of time even for problems that are trivial for other domain-independent planners.

As mentioned in the introduction, there has been considerable work in the academic community on devising domain-independent heuristics for controlling planning search. Generally, these techniques involve solving some relaxed form of the planning problem in order to obtain heuristic distance estimates, which are then used to guide search. Simple but effective ways to obtain relaxed problems are, for instance, ignoring PDDL operator delete lists or decomposing the goal set of atoms into smaller subsets [13, 7]. The computation of some of those heuristics rely on the explicit construction of a *reachability graph* [4, 5], while other methods perform *shortest path* calculations on data structures that implicitly correspond to relaxations of the search space [7]. In addition to "distance-based" heuristics, other techniques have been proposed that work on multi-valued representations of planning problems. Fast Downward [8] extracts a heuristic function by constructing a *causal graph* of the domain and a *domain transition graph* for each state variable in the domain. The first graph represents the critical interactions between state-variables, while the second graph describes the dependancies between the values of a single state variable.

Although plangraph distance estimates have been used effectively to guide POCL planners like RePop [14] and VHPOP [15], to date, EUROPA2 has not benefited from any of these techniques. There are several reasons for this, including: the variable/value representation, the lack of distinction between state and action, the lack of distinction between fact and goal, the lack of causality in the compatibilities, the large number of exogenous events and time constraints in many practical problems, and the bidirectional nature of the search strategy (which appears essential for domains involving many time constraints and exogenous events). All these factors make it difficult to directly map existing domain-independent search control strategies to EUROPA2. (Similar issues exist for other temporal planners like ASPEN [2] and IxTeT [3].)

In the next section, we develop a domain-independent control strategy for EUROPA2 that builds on the idea of constructing transition graphs [8] to compute distance-based cost estimations [7].

3. A search control strategy for EUROPA2

In order to effectively guide search in EUROPA2, we need a method of assessing the impact of each possible flaw resolution on the cost of completing a partial plan. To do this, we build a set of transition graphs and use these graphs during planning to do distance estimation. More specifically, we construct a graph for each timeline in the domain, describing the possible *transitions* between the activities on that timeline. The nodes in the graph represent activities and the transitions are induced by the information available in the compatibilities for the activities participating in the transition. Constructing a useful transition diagram for a timeline essentially requires that we reconstruct the causality hidden in the compatibilities for the different activities. A cost is associated with a transition that identifies the *temporal distance* between the activities involved in the transition. We calculate the cheapest path from any activity to any other activity by running an *all-pairs shortest path algorithm* on each graph.

Given a domain $\mathcal{D}$ and a timeline $\mathcal{T} \in \mathcal{D}$, the ***Activity Transition Graph*** for $\mathcal{T}$ is a *directed weighted graph* $\mathcal{G}[\mathcal{T}] = \{\mathrm{V}, \mathrm{E}, \mathcal{L}_{\mathrm{E}}\}$, where V is the set of vertexes, E the set of edges and $\mathcal{L}_{\mathrm{E}}$ is a weight function that assigns a numeric weight to each edge in the graph. The graph is developed as follows:

- We create a node $v \in \mathrm{V}$ for each grounded activity a that can appear on $\mathcal{T}$.
- For each activity a that belongs to $\mathcal{T}$, we split its compatibilities into two groups: *internal* compatibilities, indicated as $\mathcal{C}^{\mathrm{I}}[a_i]$, and *external* compatibilities, indicated as $\mathcal{C}^{\mathrm{E}}[a_i]$. An internal compatibility expresses a temporal constraint between the activity a and any other activity a_i belonging to the same timeline $\mathcal{T}$, while an external compatibility expresses a temporal constraint between the activity a and any other activity a_i belonging to a timeline different from $\mathcal{T}$. We examine the sets of compatibilites $\mathcal{C}^{\mathrm{I}}[a]$ and $\mathcal{C}^{\mathrm{E}}[a]$ with the purpose of defining the transitions in the graph. In particular, the set $\mathcal{C}^{\mathrm{I}}[a]$ will specify the edges appearing in the transition diagram, while the set $\mathcal{C}^{\mathrm{E}}[a]$ will dictate additional *conditions* on those edges.
- Consider the set $\mathcal{C}^{\mathrm{I}}[a]$ of internal compatibilities for a. All of these compatibilities must be either generalized `meets` or `met_by`, because activities on the same timeline cannot overlap. We define the transitions into a and out of a as follows:
  - The possible transitions out of a are described by `meets` compatibilities: for each $c \in \mathcal{C}^{\mathrm{I}}[a]$ such that $c = a$ `meets` a', we add a directed edge $e \in \mathrm{E}$ between the node corresponding to a and the node corresponding to a'. The edge e is labelled with the lower bound of the duration δ of the activity a.

– The possible transitions into a are described by `met_by` compatibilities: for each $c \in \mathcal{C}^{\mathrm{I}}[a]$ such that $c = a$ `met_by` a', we add a directed edge $e \in \mathrm{E}$ between the node corresponding to a' and the node corresponding to a. The edge e is labelled with the lower bound of the duration δ of the activity a'.

Note that there may be more than one edge into or out of a because of the presence of unbound guards in the specification of the temporal constraints involving a.

- We now consider the external compatibilities $\mathcal{C}^{\mathrm{E}}[a]$ for a and divide them into two further categories:
  – `meets`, `starts` and `contains` compatibilities, which specify that the activity a must start at a particular time at or before the start of another activity a'. We will assume that these compatibilities are describing "side effects" of the activity a and we will ignore the compatibilities in this category. (These side effects might cause interference with the behavior of other timelines, but we neglect this point here.)
  – `met_by`, `ends` and `contained_by` compatibilities, which specify that the activity a must start after the start of another activity a', or that only specify that a must start after some particular time. We will assume that these compatibilities describe "requirements" for the activity a.[1] For such compatibilities, we do not add any edge in the graph, but we keep track of them by associating a set of *conditions* with the appropriate incoming edge for the node representing a (we call the set $\mathcal{C}ond(a)$).

For the rover example, Fig. 1 shows the activity transition graph for the *Instrument*$_i$ timeline, assuming there are only two rocks in the domain.

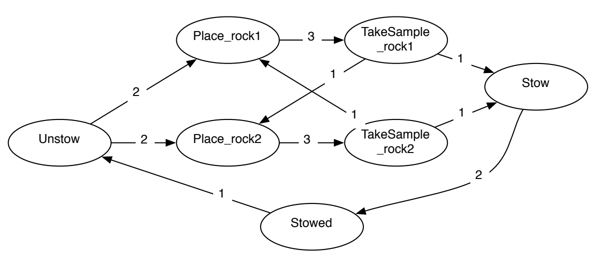

Figure 1: Activity Transition Graph for *Instrument*$_i$

Given an activity transition graph $\mathcal{G}[\mathcal{T}]$ for a timeline $\mathcal{T}$, we define $\mathcal{C}ost_{\mathrm{SP}}(a_1, a_2)$ to be the cost of the shortest path between a_1 and a_2 in the graph. Using an ***all pairs shortest-path*** algorithm we can precompute and store this information for each timeline prior to beginning planning.

We now consider how to make use of this information to do flaw resolution. Consider a partial plan Π with an open condition or ordering flaw f, and suppose that f has a possible resolution r. We define the cost of the resolution, $\mathcal{C}_{res}(r)$, as follows for merging and placement:

- *Merging* the activity a with some existing activity a' on the timeline $\mathcal{T}$:

$$\mathcal{C}_{res}(r) \equiv 0$$

- *Placing* the activity a in an empty slot s on the timeline $\mathcal{T}$. The activity a can be compatible with more than one empty slot on $\mathcal{T}$. Given one of those empty slots s, the activity a_i preceding the slot s, and the activity a_{i+1} following the slot s (see Fig. 2):

$$\mathcal{C}_{res}(r) \equiv$$
$$\mathcal{C}ost_{\mathrm{SP}}(a_i, a) + \mathcal{C}ost_{\mathrm{SP}}(a, a_{i+1}) - \mathcal{C}ost_{\mathrm{SP}}(a_i, a_{i+1})$$

The first definition corresponds to the intuition that the operation of merging has little cost, since it does not modify the partial plan except for adding new temporal constraints. Moreover, it narrows the current set of flaws while not adding any new flaws. The second definition estimates how well the activity a fits in the empty slot s on $\mathcal{T}$. Without a, there is a cost $\mathcal{C}ost_{\mathrm{SP}}(a_i, a_{i+1})$ of going from the activity a_i preceding s to the activity a_{i+1} following s. By inserting a in the slot s, we instead incur the cost $\mathcal{C}ost_{\mathrm{SP}}(a_i, a)$ of getting from a_i to a, plus the cost $\mathcal{C}ost_{\mathrm{SP}}(a, a_{i+1})$ of getting from a to a_{i+1}. The difference of these costs is an indication of the penalty incurred by placing a in the slot s. It represents the difference between the shortest path to go from a_i to a_{i+1} going through a and the direct shortest path from a_i to a_{i+1}. Clearly, if a is part of the direct shortest path, the measure is zero.

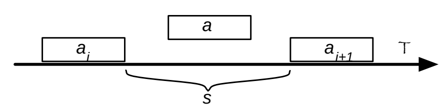

Figure 2: Placing the activity a on the timeline $\mathcal{T}$

If $\mathcal{R}[f] = \{r_1, \ldots, r_k\}$ is the set of possible resolutions for a flaw f, we define the ***Cheapest Local Resolution*** as:

$$\mathrm{CLR}(f) \equiv min_{r_i \in \mathcal{R}[f]} \mathcal{C}_{res}(r_i)$$

By using the $\mathrm{CLR}(f)$ for a placement flaw f, we prefer to place the activity in a slot where it causes the smallest increase in the net cost for the timeline $\mathcal{T}$. This provides an initial good estimate of cost since it generally prefers merging (cost zero) to other possibilities, prefers slots with low cost paths to higher cost paths, and avoids slots where no transition is possible (infinite cost).

The above scheme is fairly simple because it does not consider the interactions between an activity a and activities on other timelines. In particular it neglects the

[1] The causality for contains and contained-by compatibilities in NDDL is not always clear. The contained interval could be a temporary *effect* of the containing activity, or it could be a *condition* that must hold in order for the containing activity to function as desired. For present purposes we will assume that the contained interval is an effect rather than a condition.

requirements that must be satisfied on other timelines when placing a in a slot s. It also does not consider the *side-effects* that might result on other timelines by placing a in slot s. Omitting the side-effects is similar to "ignoring delete lists" used in many current planning systems, and we do not consider it further here. However, if we want to compute a better estimation of the cost of placing an activity on a timeline, we should consider the costs of the *conditions* that must be satisfied on the other timelines in order to make the placement possible. The information regarding conditions is available in the transition graph for the activity, since each edge is annotated with a set of conditions involving activities on other timelines (and hence appearing in other transition graphs). There are a number of possibilites for estimating the costs of satisfying these conditions. All those options basically try to estimate the cost of achieving a condition a' on a timeline $\mathcal{T}'$ by analyzing the transition graph for $\mathcal{T}'$ and calculating the $\mathrm{CLR}(f')$, where the flaw f' corresponds to the placement of a' on $\mathcal{T}'$. Two issues must be addressed when following conditions back to their transition graphs:

- *Duplication:* Conditions may be repeated for several edges along a shortest path, so we must avoid including the cost of a single condition more than once.
- *Recursion*: We could continue chasing back the conditions along the shortest path for each condition c_i, trying to get a better estimate of the cost of obtaining it. This process might never end, because conditions for achieving c_i might belong to the original timeline.

We have developed an algorithm for calculating costs of conditions that gets around those problems by first recursively collecting all the conditions into a set, and then adding up the CLRs of the conditions. This approach avoids double counting and recursion because each condition can appear at most once in the set. Space limitations prevent us from presenting the details of the algorithm so we give only a sketch here. Given a flaw f for placing an activity a on a timeline $\mathcal{T}$, the process aims at collecting the set of all the conditions on all the timelines that should be satisfied in order to perform that placement. The final cost of the placement is then taken as the sum of the CLR of the flaw f (as before) plus the costs for this set of conditions. The set is developed by recursively going backward to the graphs of the conditions for a, finding the paths to achieve them, and unioning their conditions to the set, while taking particular care that no duplicates are added. Since there are a finite number of nodes and edges in the transition graphs of the domain, this process will terminate. By doing this, we are in essence collecting the entire set of steps (over all timelines) that are necessary in order to place a on $\mathcal{T}$. This set can be seen as a relaxed plan for a.

So far, we have presented the algorithm used by the flaw resolution procedure when it has to estimate the cost of resolvers for open condition and ordering flaws. The algorithm to treat *unbound variable flaws* is based on the same concepts and mechanisms that we have just described. Choosing a resolver for an unbound variable flaw means choosing a value for a guard variable, which in turn corresponds to enforcing one set of compatibilities instead of another. In order to rank the different possible choices for a guard variable, we need to evaluate how difficult it is to achieve the compatibilities associated with that choice. Each compatibility will raise an open condition flaw or an ordering flaw and we have shown how to estimate the cost of resolving these kinds of flaws. Once we have the cost of each flaw triggered by binding the unbound variable with a certain value, we pick the value associated with the lowest cost and assign it to the guarded variable.

For *flaw selection*, we adopt a kind of *fail first* strategy by considering the most costly flaw to attack next according to the CLR metric defined above. We have also considered other more traditional heuristics for flaw selection, such as choosing the variable with "Minimum Domain Size", but these heuristics resulted in very poor performance.

4. Implementation and Experimental Results

We have some preliminary experimental results for the proposed heuristics within EUROPA2. The current implementation is in C++ and the results were obtained using a Pentium IV machine running at 1.8GHz with 1GB of RAM. Our current implementation includes pre-processing to construct the activity transition graphs and shortest-path tables, and the simple versions of the flaw resolution and flaw selection procedures. We have not yet fully implemented the more complex heuristics that recursively chain back through transition graphs to account for the conditions on graph edges. The unavailability of a benchmark set of domains written in NDDL makes performing experimentations within EUROPA2 very laborious, since domains and problems have to be manually provided. In order to carry out a more comprehensive evaluation of the performance of EUROPA2, we are developing an automatic translator from PDDL2.1 [10] to NDDL, building on the translator from PDDL2.1 to SAS^+ tasks (a variable/value representation similar in concept to NDDL) presented in [16] and [17]. The translator will provide us with the opportunity to use the benchmark problem sets developed for the International Planning Competition, a competition that is held every other year in conjunction with the International Conference on Planning and Scheduling (ICAPS). Although these results are preliminary, we present them as an indication of the fact that it is possible to successfully export key techniques developed by the classical planning community into a very different framework such as EUROPA2. In particular, we aim at showing that, if we introduce automatically derived heuristics into EUROPA2, it can work reasonably well on domains that are not specifically tailored to fit its features, without the use of hand written control rules.

We discuss the tests of the proposed heuristic on two standard domains: TOWER-n and LOGISTICS. Both these domains are particularly difficult for the standard version of EUROPA2, because they involve many causal disjunctive constraints and just a few simple temporal constraints. In our translation, an activity can correspond to either an action or a proposition. The constraints involving actions describe the conditions and the effects of actions. The constraints on propositions explain under which circumstances those propositions can be started and terminated (explanatory and regular frame axioms). The second type of constraints are particularly critical for EUROPA2 for two reasons. First, they introduce many disjunctions in the domain specification. Second, since EUROPA2 works bi-directionally, it can happen that a constraint explaining how an activity can be terminated is prematurely applied. That results in an early action commitment that is completely unmotivated with respect to the achievement of the goal. The proposed heuristic overcomes the two problems by postponing those kinds of constraints and binding disjunctive guards in an effective way.

We compare EUROPA2 with a closely related planner, CPT [18], which was awarded distinguished performance in optimal planning for temporal domains at the 2006 International Planning Competition. CPT is based on a simple extension of the STRIPS language where concurrent actions with an integer duration are allowed. A constraint programming formulation is extracted from the initial problem specification. The domain theory is hence expressed in terms of variables, their domains and constraints corresponding to disjunctions, rules and temporal restrictions. The inference machinery over this CP formulation provides a powerful pruning mechanism for discarding partial solutions generated by a classical POCL branching schema. The novelty of CPT is the ability to perform inference not only on the actions already in the partial plan, but on all the actions in the domain. Constraint propagation in EUROPA2 offers some look-ahead capability, but not the full reachability analysis provided in CPT.

The TOWER-n domain deals with the construction of a tower made of n blocks $b_1, \ldots, b_n$. Eventually, the block b_1 should be on top and b_n on the table. We consider two different initial configurations: (1) all the blocks are on the table; (2) all the blocks are on the table, except for b_n which is on top of b_1. If the original planner is run on those problems without the use of hand written control rules, it does not manage to find a plan within a time bound of hours, even for instances with only three blocks. On the other hand, when we introduce the heuristic estimators, EUROPA2 performs extremely well. For the initial configuration (1), a solution is found by pure inference and no search. In Figure 3, we show the performance of EUROPA2 on this problem considering instances from two to fifty blocks. We compare EUROPA2 with CPT. This planner, like EUROPA2, does not rely on search to solve this problem and so outperforms other state-of-the-art planners. For the initial configuration (2), EUROPA2 finds a plan with only shallow backtracking and the performance is comparable with that shown in Figure 3. Although this problem appears trivial for classical planners, powerful systems such as FF [5] cannot solve it.

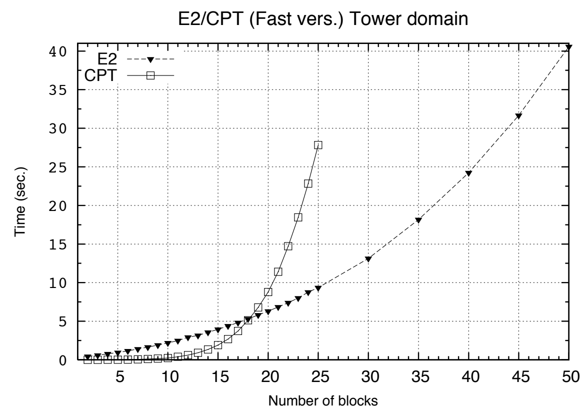

Figure 3: Results for the Tower domain

LOGISTICS is the classical problem of moving packages between different cities using trucks and planes. We analyze several problems, from small ones (e.g. "log01" has six packages, two trucks, one airplane, two cities, two locations and two airports) to large ones (e.g. "log39" has twenty one packages, seven trucks, two airplanes, seven cities, seven locations and seven airports). This domain differs from TOWER since it presents many independent subgoals, while TOWER is characterized by many dependent subgoals. Figure 4 shows that EUROPA2 scales well and manges to solve all the problems (sometimes even without search). This is not the case for CPT, which fails to solve the largest instances. Without the new control heuristics, EUROPA2 is not able to find a plan within a time bound of hours for trivial instances, such as those involving three packages and two cities.

5. Conclusions and Future Work

We have developed novel domain-independent search control techniques for the EUROPA2 planning system. These techniques construct transition graphs for each timeline in the domain model and use these graphs to estimate the cost of resolving flaws in different ways. This information is used to guide both flaw selection, and flaw resolution. Although our experimental results are preliminary, they suggest that EUROPA2 can get by with far less domain-dependent guidance, and can successfully function as a general purpose engine if it makes use of these powerful domain-independent heuristics.

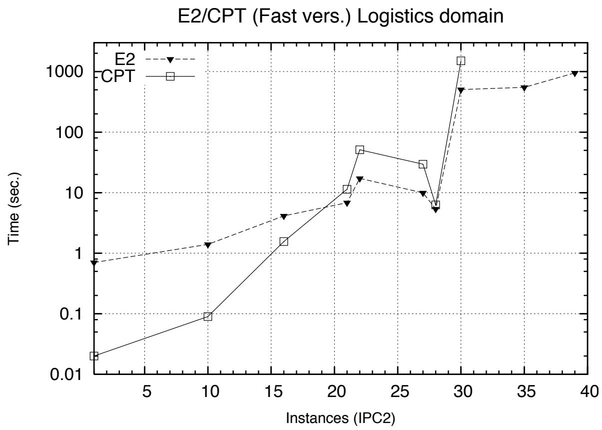

Figure 4: Results for the Logistic domain

In future work, we plan to carry out a more comprehensive performance evaluation by considering not only classical problems developed for the International Planning Competition, but also real-world problems. In particular, we intend to take domains that were originally developed for EUROPA2 and traditionally solved by using domain-dependent control rules, remove these rules, and run the new algorithm against the resulting domains. Those domains are usually big in size and contain many complex temporal constraints. More empirical work is needed to test whether the automatically generated heuristics described in this paper can extract sufficient knowledge from the domain analysis, and obtain acceptable performance comparable to that produced by hand-crafted heuristics.

6. Acknowledgements

We thank Jeremy Frank, Javier Barreiro, Nicola Muscettola, Michael Iatauro, David Rijsman, Peter Jarvis and Patrick Haslum for their comments on the work presented in the paper. We thank Malte Helmert for making his translator from PDDL to SAS^+ available. This work was supported by the Association for International Practical Training (AIPT), and by the Automation for Operations project of the NASA ETDP program.

7. References

[1] J. Frank and A. Jonsson, “Constraint-based Attribute and Interval Planning,” *Journal of Constraints*, vol. 8, no. 4, 2003, pp. 339–364.

[2] S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B.Smith, F. Fisher, T. Barret, G. Stebbins, and D. Tran, “ASPEN - Automated Planning and Scheduling for Space Missions Operations,” in *International Conference on Space Operations (SpaceOps 2000)*, Toulouse, France, 2000.

[3] M. Ghallab and H. Laruelle, “Representation and Control in IxTeT, a Temporal Planner,” in *Proc. of the Second International Conference on Artificial Intelligence Planning Systems (AIPS-94)*. AAAI Press, 1994, pp. 61–67.

[4] A. Blum and M. Furst, “Fast planning through planning graph analysis,” *Artificial Intelligence*, vol. 90, 1997, pp. 281–300.

[5] J. Hoffmann and B. Nebel, “The FF Planning System: Fast Plan Generation through Heuristic Search,” *Journal of Artificial Intelligence Research*, vol. 14, 2001, pp. 253–302.

[6] J. Bresina, A. Jonsson, P. Morris, and K. Rajan, “Activity Planning for the Mars Exploration Rovers,” in *Proc. of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS-05)*, 2005, pp. 40–49.

[7] P. Haslum and H. Geffner, “Admissible Heuristics for Optimal Planning,” in *Proc. of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS-00)*, 2000, pp. 140–149.

[8] M. Helmert, “The Fast Downward Planning System,” *Journal of Artificial Intelligence Research*, vol. 26, 2006, pp. 191–246.

[9] J. Allen, “Maintaining Knowledge about Temporal Intervals,” *Communications of the ACM*, vol. 26, no. 11, 1983, pp. 832–843.

[10] M. Fox and D. Long, “PDDL 2.1: An extension to PDDL for expressing temporal planning domains,” *Journal of Artificial Intelligence Research*, vol. 20, 2003, pp. 61–124.

[11] S. Kambhampati, C. A. Knoblock, and Q. Yang, “Planning as refinement search: a unified framework for evaluating design tradeoffs in partial-order planning,” *Artificial Intelligence*, vol. 76, no. 1-2, 1995, pp. 167–238.

[12] D. McAllister and D. Rosenblitt, “Systematic nonlinear planning,” in *Proc. of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, vol. 2. AAAI Press, 1991, pp. 634–639.

[13] B. Bonet and H. Geffner, “Planning as heuristic search,” *Artificial Intelligence*, vol. 129, no. (1-2), 2001.

[14] X. Nguyen and S. Kambhampati, “Reviving Partial Order Planning,” in *Proc. of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, 2001, pp. 459–466.

[15] H. Younes and R. Simmons, “VHPOP: Versatile Heuristic Partial Order Planner,” *Journal of Artificial Intelligence Research*, vol. 20, 2003, pp. 405–430.

[16] S. Edelkamp and M. Helmert, “Exhibiting Knowledge in Planning Problems to Minimize State Encoding Length,” in *Proc. of the Fifth European Conference on Planning (ECP’99)*, 1999, pp. 135–147.

[17] M. Helmert, “A Planning Heuristic based on Causal Graph Analysis,” in *Proc. of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS-04)*, 2004, pp. 161–170.

[18] V. Vidal and H. Geffner, “Branching and pruning: An optimal temporal POCL planner based on constraint programming,” *Artificial Intelligence*, vol. 170, no. 3, 2006, pp. 298–335.