Exploiting Path Refinement Abstraction in Domain Transition Graphs

Peter Gregory and **Derek Long** and **Craig McNulty** and **Susanne Murphy**
Department of Computer and Information Sciences
University of Strathclyde, Glasgow, UK
firstname.lastname@cis.strath.ac.uk

Abstract

Partial Refinement A-Star (PRA*) is an abstraction technique, based on clustering nearby nodes in graphs, useful in large path-planning problems. Abstracting the underlying graph yields a simpler problem whose solution can be used, by refinement, as a guide to a solution to the original problem. A fruitful way to view domain independent planning problems is as a collection of multi-valued variables that must perform synchronised transitions through graphs of possible values, where the edges are defined by the domain actions. Planning involves finding efficient paths through Domain Transition Graphs (DTGs). In problems where these graphs are large, planning can be prohibitively expensive. In this paper we explore two ways to exploit PRA* in DTGs.

1 Introduction

Abstraction is a powerful tool for search problems, including planning. Recent research in abstraction in planning (Helmert, Haslum, and Hoffmann 2007; Domshlak, Hoffmann, and Sabharwal 2009; Haslum et al. 2007; Edelkamp 2002; Katz and Domshlak 2008) has explored a range of techniques including value abstraction, in which the domains of one or more variables are abstracted. Value abstraction is a graph-abstraction technique, applied to Domain Transition Graphs (DTGs) (Helmert 2008). A separate strand of research is concerned with efficiently finding paths through very large graphs, using abstraction as a way to reduce and simplify the problem: Partial Refinement A-Star (PRA*) (Sturtevant and Buro 2005) is one example of this work in which the authors use abstraction to simplify the task of navigating large graphs. In this paper we consider how this idea can be adapted to help to navigate DTGs.

We demonstrate two methods for using ideas inspired by PRA* in planning, to improve search performance: *abstract-and-refine* and *abstract-and-conquer*. Abstract-and-refine begins with the creation of a hierarchy of abstracted planning problems. A solution to one of these abstract problems then determines the abstract problem that must be solved directly below it in the hierarchy. Abstract-and-conquer, on the other hand, uses the structure of a solution to an abstracted problem to define intermediate goals (which can be seen as a form of landmark) generating a sequence of planning problems whose linked solutions solve the original problem. We demonstrate, using variants of the FF (Hoffmann and Nebel 2001) and LAMA (Richter and Westphal 2010) algorithms, the effectiveness of each of these methods of abstraction for planning problems.

2 Background

Abstraction is not new to planning: Sacerdoti (1973) developed ABSTRIPS, a solver that exploits a hierarchy of layers of abstraction, each removing detail from the layer beneath it. Bacchus and Yang (1991) identified an important property of abstraction techniques that influences how effective they are in improving efficiency: the downward-refinement property, that holds when all solutions in abstract space can be refined into less abstract solutions without need to backtrack across the abstraction hierarchy.

More recently, both heuristic and optimal planning approaches using pattern databases (Edelkamp 2002; Haslum et al. 2007) exploit abstractions for heuristic guidance. The merge-and-shrink approach (Helmert, Haslum, and Hoffmann 2007) exploits a form of abstraction, in which DTGs are multiplied together (to merge them) and then shrunk by abstracting nodes that share the same relaxed distance to the goal. Katz and Domshlak (2008) have explored a third kind of abstraction they call *structural abstraction*. Domshlak *et al* (2009) also examine the ways that abstraction interacts with planning-as-SAT-solving.

Abstraction has been explored across many areas of combinatorial problem-solving, but our work is inspired by Sturtevant and Buro's Partial Refinement A-Star (PRA*) (2005) algorithm for path planning with abstractions. A crucial factor in their work is that the structure of the abstract solutions they find forms the basis of the final complete solutions. In this work, we emulate this idea in domain-independent planning: we find abstract plans, and use them to form the basis of concrete plans.

2.1 Terminology

The SAS+ planning formalism (Bäckström and Nebel 1995) has been widely adopted as a complementary formalism to the PDDL (Fox and Long 2003) planning formalism. In SAS+, the state of a planning problem is defined by a set of finite-domain variables. We denote the domain of variable V by $D(V)$. An operator in SAS+ encodes a valid transition between values of the variables, possibly requiring other variables to hold certain values to support the transition.

Definition 1 (SAS+ **Operator).** *A* SAS+ *Operator is a pair $O = \langle \mathcal{I}, \mathcal{T} \rangle$, where:*

$\langle V, v \rangle \in \mathcal{I}$ is a constraint that requires variable V to have value v during execution of O, known as a prevail condition. Prevail(O) *is the function that returns $\mathcal{I}$.*

$\langle V, u, v \rangle \in \mathcal{T}$ is a pre/post condition, where variable V must have the value u before, and takes the value v after, application of O. PP(O) *is the function that returns $\mathcal{T}$.*

Definition 2 (SAS+ **Planning Task).** *A planning problem, π, is a tuple $\langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$, where $\mathcal{V}$ is a set of finite-domain* SAS+ *variables, $\mathcal{O}$ is a set of operators over $\mathcal{V}$, s_0 is a set of initial assignments to $\mathcal{V}$ and $s_\star$ is a set of goal assignments to a subset of $\mathcal{V}$. A solution to π is a sequence of actions $a_1, ..., a_l$ in which the final state satisfies $s_\star$.*

We assume throughout that we are solving *ground* SAS+ problems and that static preconditions of operators have been removed during grounding, along with operators with unsatisfied static preconditions.

The DTG for a SAS+ variable is a graph over nodes representing the possible values of the variable and with directed edges representing possible transitions between values induced by actions in the domain. Helmert (2009) automatically translates a significant fragment of PDDL into SAS+, including construction of DTGs.

The edges of DTGs can correspond to multiple different kinds of actions, but in many cases DTGs exhibit a strong homogeneity in which all the actions are instances of the same kind (for example, all *drive* actions). DTGs capture an underlying accessibility graph between values in the variable domains, equivalent to maps over which the variables can be considered to move (literally or metaphorically) according to the legal transitions. Fox and Long (2000) exploited this observation in Hybrid STAN to identify accessibility graphs and then use a specialised shortest-path algorithm to reduce planning costs; other researchers have exploited similar ideas. However, in contrast to the current work, none of the planners that use this structure exploit abstraction on the underlying graphs.

3 Abstracting Planning Problems

We begin by defining an abstraction of a SAS+ variable.

Definition 3 (Abstraction of a SAS+ **Variable).** *An abstraction of variable V, is an abstract variable, V' and a surjective* abstraction function, *$f_V : D(V) \rightarrow D(V')$.*

An abstraction function extends to prevail conditions: $f_V(\langle V, v \rangle) = \langle V', f_V(v) \rangle$ and $f_V(\langle W, v \rangle) = \langle W, v \rangle$ if $W \neq V$. It extends to pre/post conditions analogously and to operators by application to all their elements.

We say that two values, u and v, from the domain of V are abstracted together if $f_V(u) = f_V(v)$. An abstract planning problem is a planning problem where all of the variables are replaced by abstract variables and all values in the operators are replaced by their corresponding abstract values.

Definition 4 (Abstract SAS+ **Planning Task).** *Given a planning task, $\pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$, and a set of abstraction functions, $F = \{f_V \mid V \in \mathcal{V}\}$, the abstract planning task π' is defined to be $\langle \mathcal{V}', \mathcal{O}', s'_0, s'_\star \rangle$, where $\mathcal{V}'$ is the set of variables abstracting $\mathcal{V}$, $\mathcal{O}'$ is the image of $\mathcal{O}$ under the composition of the functions in F, $s'_0 = \{\langle V', f_V(v) \rangle | \langle V, v \rangle \in s_0\}$ and $s'_\star = \{\langle V', f_V(v) \rangle | \langle V, v \rangle \in s_\star\}$.*

One aspect of successful use of abstraction lies in appropriate choice of abstraction functions. In PRA*, the abstraction function abstracts small cliques of nodes in the accessibility graph. Abstractions can be constructed similarly in planning: values in small cliques in DTGs can be abstracted to create abstract planning problems.

3.1 DTG Abstractions

In PRA*, abstract nodes are constructed from strongly connected cliques in the graph. Using cliques has the desirable property that all concrete points within the abstract node are directly reachable from each other. In the work that follows, the abstraction that we use is always clique-based abstraction, using cliques of size two. We abstract together two values, u and v, only if there are two actions containing the pre/post condition $\langle V, u, v \rangle$ and $\langle V, v, u \rangle$. Our basic approach to abstraction of DTGs is to iteratively select an arbitrary pair of mutually linked values and abstract them into a single value. At successive iterations we choose pairs of values that have not yet been abstracted and repeat the process. The abstraction process itself can be iterated to achieve successive levels of abstraction, as we discuss in section 4.

Whether an abstraction preserves the downward-refinement property (DRP) depends, in part, on the dependency relations between abstracted variables and other variables in the problem. A causal graph (Helmert 2008) is a directed graph recording the dependencies between variables in a planning domain. We say that a variable, V, lies on a causal chain if the subgraph in the causal graph containing V and those variables with a path to V is acyclic and none of the variables in this subgraph (other than V itself) has an edge to a variable not in the subgraph. We say that the causal chain is *accessible* if the values of the variables in the chain all form strongly connected components. There are additional constraints on DTGs that can contribute to preservation of the DRP:

Homogeneous DTG (HD) A DTG is homogeneous if all its transitions correspond to the same action type.

Unary Effect (UE) A DTG is unary effect if every edge is an action that has only one pre/post condition.

Uniform Prevail Conditions (UPC) A DTG has uniform prevail conditions if, for each transition in the DTG, A, the DTG is strongly connected when restricted to edges with the same prevail condition as A.

Many domains in which abstraction proves useful satisfy HD and it can make the process of finding DTGs that are appropriate to abstract faster if we restrict attention to those with HD. DTGs with UPC and HD represent a class of DTGs that can be easily tested to determine the accessibility of their causal chains.

Theorem 1. *Abstractions of a UE DTG on an accessible causal chain have the DRP.*

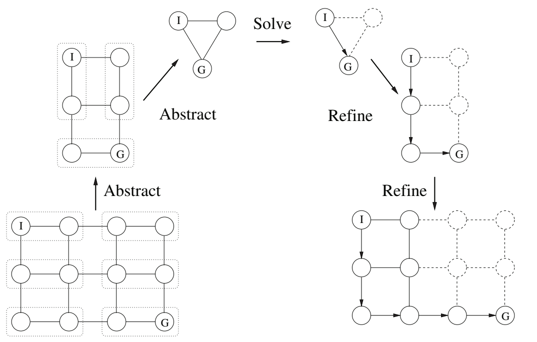

Figure 1: An example of an abstract-and-refine process. At left, nodes bordered by dotted lines are abstracted in the next level. At right, successive abstract problems are solved. Dotted structure is removed in refinement to lower levels.

Proof (sketch): This result appears to be related to a theorem of Haslum's (2007) and a result of Helmert's (2008), showing that abstraction of strongly connected components in free DTGs preserves DRP. However, it should be noted that those results are concerned with abstraction by removal of a variable, where this result is concerned with abstraction by reduction of the value set in a DTG.

The proof depends on showing that DRP holds for a single variable abstraction, f_V, which can be generalised to multiple abstractions by induction. Suppose that $a'_1, ..., a'_n$ is a plan for the abstracted planning task. Each action, a'_i is the abstraction of a corresponding action, a_i, in the original task. Consider the process of applying the actions $a_1, ..., a_n$ in sequence. Suppose some action, a_i, is not executable in the state, s_{i-1}, resulting from application of the preceding actions. Then there must be distinct values, x and y, such that $V = x$ in s_{i-1}, $V = y$ is a precondition of a_i and $f_V(x) = f_V(y)$. To demonstrate DRP it is sufficient to show that there is a sequence of actions, B, applicable to s_{i-1} such that the state following execution of B is $s_{i-1}[V = y]$. In this state a_i can execute and an inductive argument confirms that the complete plan can be similarly repaired.

The existence of the sequence B follows from the fact that V lies on an accessible causal chain, so each variable in the chain can be set to allow transitions for the variables below it in the chain, and their values can be restored after V has been switched from x to y. UE is required to ensure that the actions that change V from x to y do not have any further effects on other variables in the problem. □

4 Abstract-and-Refine

Planning problems can scale in two ways: the number of DTGs can grow and the size of DTGs can grow. In the former case, planning problems become harder because of the need to coordinate the values of many interacting variables, while in the latter they become harder because of the need to navigate variables through larger DTGs. As the number of values in DTGs increases, the performance of many planners can be observed (empirically) to degrade exponentially — this is true, for example, of those planners based on FF (see, for instance, figures 2 and 4).

We hypothesise that an abstract-and-refine approach can be used to improve scaling performance on problems with large DTGs, such as the Driverlog-1,1,2 problem family, a set of Driverlog problems in which there is one truck, one driver and two packages to deliver, but a varying number of locations to traverse.

4.1 Abstract-and-Refine Algorithm

In order to define the abstract-and-refine (A&R) algorithm, we introduce a *hierarchy of abstractions* and *refinement*.

Definition 5 (Abstraction Hierarchy). *Given a planning task π, an abstraction hierarchy is a sequence of planning tasks, $\pi, \pi_1, ..., \pi_n$, where π_1 is an abstraction of π, and π_i is an abstraction of π_{i-1}.*

Definition 6 (Refinement). *Given a planning task $\pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$, an abstract planning task π' and the set of facts, $F(P)$, used in a plan P solving π', a refinement is the planning task $\pi^R = \langle \mathcal{V}, \mathcal{O}_R, s_0, s_\star \rangle$ where*

$$\mathcal{O}_R = \{\langle \mathcal{I}, \mathcal{T}, c \rangle \in \mathcal{O} | \forall \langle V, v \rangle \in \mathcal{I}\ f_V(v) \in F(P) \textbf{ and } \forall \langle V, u, v \rangle \in \mathcal{T}\ f_V(u), f_V(v) \in F(P)\}$$

Definition 6 is the key to the A&R approach: the refinement, π^R, of a problem, π, with respect to its abstraction, π', and a solution, P', of π' is a *restriction* of the original problem. The restriction removes the values from the domains of abstracted variables if those values are not used in P'. The process also removes ground operators referring to these values. π^R is (usually) easier to solve than π because it is smaller, containing fewer irrelevant values (and, correspondingly, fewer grounded operators).

A hierarchy of abstractions is constructed using a simple process to select the values to abstract together in each successive level of abstraction. In our implementation we restrict attention to DTGs satisfying HD and UE. This process ends when it reaches a fixpoint. To select the starting point for search within the abstraction hierarchy we follow the proposal made by Sturtevant and Buro (2005) and begin at the middle layer in the hierarchy.

Once an abstracted problem is solved, the plan generated at that level is used to guide the search for a plan at the next level in the hierarchy. This is achieved by removing from the refinement all the values that do not correspond to abstract values appearing in the abstract plan. The process is illustrated in Figure 1. The A&R algorithm is as follows:

```
Abstract and Refine Algorithm
  Given a planning task $\pi$:
  1. Let $\pi, \pi_1, \ldots, \pi_n$ be the abstraction hierarchy for $\pi$.
  2. Let $c = \lfloor n/2 \rfloor$
  3. Repeat, while $c \geq 0$:
     (a) Let $P$ be a solution to $\pi_c$
     (b) Decrement c
     (c) Let $\pi_c$ be the refinement of $P$ and $\pi_{c+1}$
  4. Return $P$
```

4.2 Algorithm Properties

A&R does not backtrack over bad decisions; if an abstraction that does not preserve DRP refines into an unsolvable problem then no solution is returned. In the general case abstractions do not preserve DRP, therefore A&R is an incomplete algorithm. In specific cases (for example, when

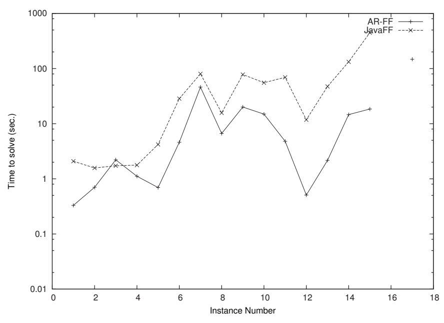

Figure 2: Time to plan in the Roadlog domain.

Theorem 1 holds) completeness can be guaranteed. However, we do not restrict ourselves to these cases, we insist only on the UE property. We will demonstrate the practical implications of this decision.

The Grid domain is an example affected by loss of DRP and consequent incompleteness. In this domain, a single variable encodes the position of a robot in a grid. Some of the locations begin locked, requiring a key to unlock them. If the abstraction algorithm abstracts two values together that represent a locked and an unlocked location, then the locations will be considered both locked and unlocked during planning. Abstract plans will, therefore, not necessarily use paths that visit the locations of keys that are required to open locked locations on the path once it is refined. This problem demonstrates that the abstraction does not have DRP in general: backtracking across abstraction layers is necessary if a solution is to be found in this case.

Theorem 1 shows how DRP can be maintained, at the price of reduced applicability of the approach. The Grid domain also violates UPC since the transition from a to b has a prevail condition that b be unlocked, but the same prevail condition is not sufficient to allow transition from b to a.

4.3 Empirical Evaluation

We compare the performance of A&R FF (AR-FF) with JavaFF (Coles et al. 2008) (an easily modifiable Java implementation of FF). AR-FF is built on JavaFF so the comparison evaluates the effect of the A&R process. Experiments used an Intel 3.16GHz Dual Core CPU with limits of 2GB memory and 15 minutes.

Choice of Benchmarks We present results for five benchmark domains: Roadlog, Driverlog, Rovers, Goldminer and Grid. AR-FF abstracts (according to the process described in section 3.1), but only DTGs satisfying the UE restrictions. Any problem which has no DTGs meeting these criteria is not abstracted and is therefore solved in the same time and memory as standard JavaFF. We therefore only show results on benchmarks containing UE DTGs. Roadlog is a variant of Driverlog in which the drivers are removed and the maps are planar graphs: we explore this version because the geometric structure of the maps is more realistic and also, possibly, more susceptible to the PRA* approach. The first five Grid problems are the IPC instances, the remainder were generated by the authors. The Goldminer instances are the problems used in the IPC learning track for evaluation.

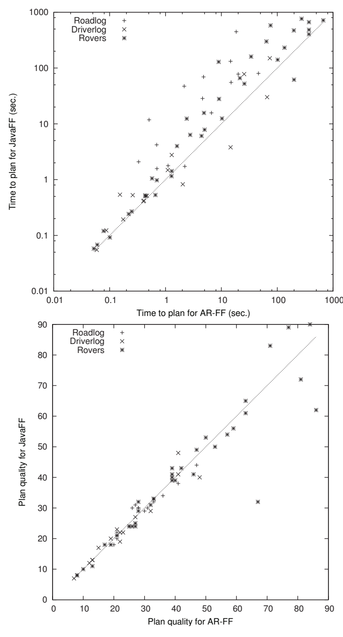

Figure 3: Time to plan for A&R (top), Quality of plans for A&R (bottom)

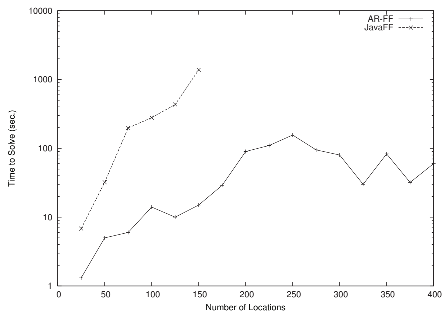

Figure 4: Performance of JavaFF and AR-FF in Driverlog-1,1,2, with increasing numbers of locations (x-axis).

AR-FF is influenced by two opposing factors: solving multiple abstract problems slows search, while refining planning problems reduces their size and hence the time to solve them. The number of layers scales approximately logarithmically with the number of locations, as expected.

Refinement removes choices for the planner at each level of the abstraction. This creates a tradeoff: the abstraction-based search can speed the discovery of a solution, but the pruned search space can prune high quality solutions, so that the final result is a lower quality solution than is optimal in the original search space. Of course, JavaFF is not an optimal planner, so there is no guarantee that either planner finds optimal plans in any of the problems.

In the Roadlog domain, Figure 2 shows that time performance is improved. In the Rovers domain (Figure 3) plans are typically found more quickly, especially for larger instances. Figure 3 shows the distribution of relative plan qualities across the domains, confirming that the use of abstractions does not lead to a significant change in plan quality

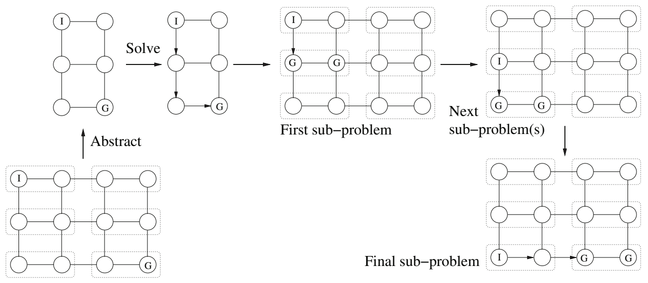

Figure 5: A&C. The problem is abstracted (left) and the resulting problem solved (top). Each abstract action defines a new problem (right), solved in sequence.

across the entire problem set.

Due to the incompleteness issues discussed in the previous section, AR-FF fails to solve some of the Goldminer and Grid instances: AR-FF fails to solve 4/30 Goldminer instances, but solves 25 more instances overall than JavaFF. AR-FF solves 16/20 Grid problems compared with 11/20 for JavaFF. In these domains at least, the advantage gained in scalability improvements outweighs the disadvantage presented by the incompleteness.

Figure 4 shows results for Driverlog-1,1,2 problem: AR-FF solves instances with 400 locations relatively easily, while JavaFF fails at 150-location problems, demonstrating that the A&R approach can improve the scaling performance of planners faced with large DTGs.

5 Abstract-and-Conquer

We now consider a different way to exploit the same abstraction, using an *abstract-and-conquer* approach (A&C). The name references the standard divide-and-conquer recursive algorithm design strategy. In this approach, an abstract plan is found and the abstract effects of each action are used as intermediate goals. This is similar to the use of landmarks in STeLLa (Sebastia, Onaindia, and Marzal 2002) to create a sequence of planning problems.

Figure 5 illustrates the process. The abstract solution is used as a skeleton for the concrete plan. The sequential search is shown on the right side of the diagram. For each action in the abstract plan a corresponding sub-problem is created with an initial state equal to the final state of the plan so far and a goal state equal to the *abstract* effect of the action. This abstract goal corresponds to a disjunctive concrete goal formed from the abstracted nodes in the graph. More generally, the process can apply to a hierarchy of abstractions (in direct analogy with the recursive use of divide-and-conquer), but we have found it best to apply only one or two levels of abstraction before solving the problem directly.

A formal description of the algorithm relies on the following definition of an *augmented* planning task.

Definition 7 (Augmented SAS+ Planning Task). *Given a planning task, $\pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$, and an abstract planning task $\pi' = \langle \mathcal{V}', \mathcal{O}', s_0', s_\star' \rangle$, an augmented planning task is defined as $\pi^+ = \langle \mathcal{V}^+, \mathcal{O}^+, s_0^+, s_\star^+ \rangle$, where $\mathcal{V}^+ = \mathcal{V} \cup \mathcal{V}'$,*

$$\mathcal{O}^+ = \{ \langle \mathcal{I} \cup \mathcal{I}', \mathcal{T} \cup \mathcal{T}' \rangle | \, O = \langle \mathcal{I}, \mathcal{T} \rangle \in \mathcal{O} \text{ and } \langle \mathcal{I}', \mathcal{T}' \rangle \in \mathcal{O}' \text{ is the abstraction of } O \}$$

$s_0^+ = s_0 \cup s_0'$ and $s_\star^+ = s_\star$.

The A&C algorithm is as follows:

Abstract-and-Conquer Algorithm

```
Given a planning task, $\pi$:
1. Let $\pi'$ be an abstraction of $\pi$.
2. Let $\langle \mathcal{V}^+, \mathcal{O}^+, s_0^+, s_\star \rangle$ be the augmentation of $\pi$ and $\pi'$.
3. Let $P'$ be a plan for $\pi'$, let $P_0$ be the empty plan, let $s_c = s_0^+$.
4. For all $O' \in P'$:
   (a) Let $\mathbf{effs}(O')$ be the effects of $O'$
   (b) Let $P^+$ be a plan for $\langle \mathcal{V}^+, \mathcal{O}^+, s_c, \mathbf{effs}(O') \rangle$
   (c) Let $P_c = P_c + P^+$ and $s_c$ be the final state induced by $P^+$
5. Let $P^+$ be a plan for $\langle \mathcal{V}^+, \mathcal{O}^+, s_c, s_\star \rangle$.
6. Return $P_c + P^+$
```

This algorithm generates a plan for the original problem by linking together subplans connecting the last state of the preceding subplan to a goal created by taking the effects of the next abstract action in a solution to the abstracted problem. Since achieving the effects of the last abstract operator might not achieve the original goal, a last subplan has to be added to navigate to the final goal state. In the best case, there will be nothing to do in this step; in the worst-case, too much information is lost in the abstraction and significant extra work is required. In A&C we lift the restriction that abstraction be applied only to DTGs satisfying HD and UE.

5.1 Algorithm Properties

One clear advantage of A&C over the abstract-and-refine approach is that the problems associated with DRP are reduced. This is because each augmented problem has access to the entire set of operators, since no refinement-based filtering takes place. The improvement in planning performance is gained from the planner having to find relatively short plans to achieve the effects of each abstract step. However, there is still a potential problem with incompleteness that could be resolved by backtracking, where, in achieving an intermediate abstract goal, the plan moves into a dead-end. This can happen if the problem involves tightly constrained resources, where the abstract plan can mislead the planner into pursuing a path that cannot be refined within the constraints of the resource availability. We have not experienced this problem in practice.

5.2 Empirical Analysis

We implemented A&C as a simple harness around LAMA (Richter, Helmert, and Westphal 2008), creating a SAS+ instance for each of the individual steps, each passed to a new invocation of LAMA. A disadvantage of this simple implementation is that the majority of execution time is spent writing subproblem instances to file and in LAMA rereading them. For this reason, search nodes provide a better reflection of performance than run-times.

Figure 6 (left) compares the numbers of nodes expanded when using A&C or not. The graph is log-scaled, so the benefit favours A&C with exponential improvements, particularly in Rovers, Logistics, Satellite and Driverlog. The behaviour in Grid and Goldminer is mixed, although the pattern is similar. These are both domains in which the abstract solutions can require significant work to refine, due to the resource constraints in each of the domains.

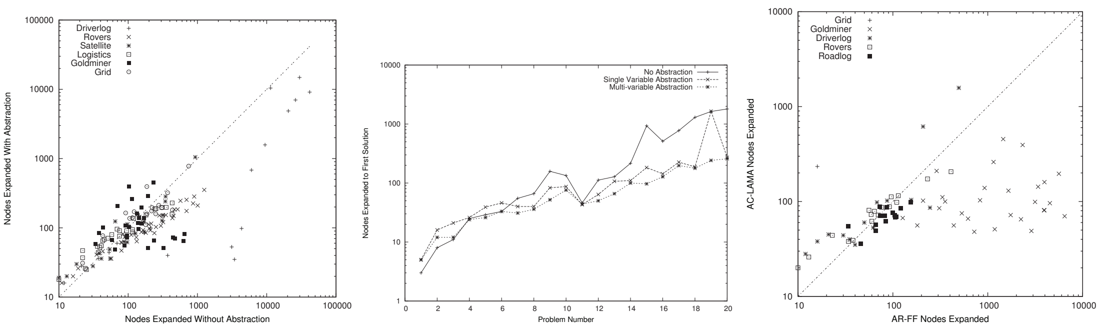

Figure 6: Nodes expanded by LAMA to find first solution: (left) with A&C (log-log-scaled); (centre) using no abstraction, single-variable-abstraction and multi-variable-abstraction (log-scaled) in the Zeno domain. Right: Comparison of A&R and A&C.

The performance of A&C in the Zeno domain (Figure 6) shows the potential benefit of lifting the UE constraint. With the UE restriction, only the passenger variables abstract.

In terms of quality, abstract-and-conquer generally produces plans that are up to 10% poorer, although in a few problems in Grid, Goldminer and Driverlog domains the quality deteriorates by as much as 50-100%. It is unsurprising that quality is adversely affected by the approach, since the subplans are combined naively.

Figure 6 (right) shows a comparison of numbers of nodes expanded using AR-FF and A&C. This comparison is not entirely equal, since the work performed at nodes is different between LAMA and FF. Nevertheless, the trend appears to favour the A&C approach.

6 Conclusions

The ability to abstract is the ability to focus on relevant information, while discarding irrelevant detail. We have presented Abstract-and-Refine-FF, a planner that uses graph abstraction and refinement in order to improve planning performance, and a second approach to exploitation of abstraction, based on abstract-and-conquer.

In our experiments AR-FF is shown to solve more instances and solve larger instances faster than JavaFF while not significantly reducing plan quality. Our second approach offers an exponential improvement in the number of nodes searched by LAMA. We anticipate that this will translate into a similar time-performance benefit in an integrated implementation. Nevertheless, one of the attractions of the abstract-and-conquer approach is that it can be used easily with any planning system. Abstraction approaches provide fertile ground both for new planning research and for further increasing the scalability of planning algorithms.

References

Bacchus, F., and Yang, Q. 1991. The Downward Refinement Property. In *Proc. Int. Joint Conf. on AI (IJCAI)*, 286–292.

Bäckström, C., and Nebel, B. 1995. Complexity Results for SAS+ Planning. *Computational Intelligence* 11:625–656.

Coles, A. I.; Fox, M.; Long, D.; and Smith, A. J. 2008. Teaching Forward-Chaining Planning with JavaFF. In *Colloquium on AI Education, 23rd AAAI*.

Domshlak, C.; Hoffmann, J.; and Sabharwal, A. 2009. Friends or Foes? On Planning as Satisfiability and Abstract CNF Encodings. *J. Art. Intel. Res. (JAIR)* 36:415–469.

Edelkamp, S. 2002. Symbolic pattern databases in heuristic search planning. In *Proc. Int. Conf. on AI Planning and Scheduling (AIPS)*, 274–283.

Fox, M., and Long, D. 2000. Hybrid STAN: Identifying and Managing Combinatorial Optimisation Sub-problems in Planning. In *Proc. Int. Joint Conf. on AI (IJCAI)*, 445–452.

Fox, M., and Long, D. 2003. PDDL2.1: An extension of PDDL for expressing temporal planning domains. *J. Art. Intel. Res. (JAIR)* 20:61–124.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proc. 22nd Conf. Ass. Adv. AI (AAAI)*, 1007–1012.

Haslum, P. 2007. Reducing accidental complexity in planning problems. In *Proc. Int. Joint Conf. on AI (IJCAI)*, 1898–1903.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible Abstraction Heuristics for Optimal Sequential Planning. In *Proc. Int. Conf. AI Planning and Scheduling (ICAPS)*, 176–183.

Helmert, M. 2008. *Understanding Planning Tasks: Domain Complexity and Heuristic Decomposition*, volume 4929 of *Lecture Notes in Computer Science*. Springer.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Art. Int.* 173(5-6):503–535.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *J. Art. Intel. Res. (JAIR)* 14:253–302.

Katz, M., and Domshlak, C. 2008. Structural Patterns Heuristics via Fork Decomposition. In *Proc. Int. Conf. AI Planning and Scheduling (ICAPS)*, 182–189.

Richter, S., and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Art. Intel. Res. (JAIR)* 39:127–177.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks Revisited. In *Proc. Conf. Ass. Adv. of AI (AAAI)*, 975–982.

Sacerdoti, E. D. 1973. Planning in a hierarchy of abstraction spaces. In *Proc. Int. Joint Conf. on AI (IJCAI)*, 412–422.

Sebastia, L.; Onaindia, E.; and Marzal, E. 2002. STeLLa v2.0 : Planning with Intermediate Goals. In *Proc. IBERAMIA*, 805–814.

Sturtevant, N. R., and Buro, M. 2005. Partial Pathfinding Using Map Abstraction and Refinement. In *Proc. Nat. Conf. on AI (AAAI)*, 1392–1397.