Abstractions in a Domain Independent Planning Context

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

Abstract

Graph abstractions have previously been used to improve performance of search in path planning. If a path planning problem has a large underlying graph, then abstractions can help to speed up search. By clustering local vertices, a smaller graph can be obtained. A solution to this smaller problem can then be refined, giving a solution to the original problem. This technique has been shown to produce good quality solutions.

We apply this technique to domain-independent planning. Many natural planning problems have underlying graphs and these can be abstracted by a process similar to previous approaches in path-planning. In particular, the domain transition graphs of planning problems can be abstracted.

In order to make use of abstractions in domain-independent planning, we introduce two different approaches. One is based on an abstract-and-refine procedure, abstracting the problem several times, and then using the abstract solutions to filter values at lower abstraction levels. The other uses the abstract plan to create a sequence of intermediary goals in a divide-and-conquer approach.

Introduction

Abstraction is a powerful tool by which to reduce the complexity of hard combinatorial problems, such as planning. However, its exploitation has proved elusive: the challenge is always to find ways to remove complexity and detail, while retaining sufficient structure for the solution to the abstracted problem to give meaningful guidance in the solution of the original problem.

We demonstrate two methods for using abstraction in planning to improve search performance. The first of these we call *abstract and refine*, the second *abstract and conquer*. Abstract and refine is a process by which a hierarchy of abstraction planning problems are created. A solution to one of these abstract problems defines the abstract problem directly lower in the hierarchy. Abstract and conquer, on the other hand, finds a single abstract plan which defines a sequence of intermediary goals. Finding a plan to the first of these goals provides a new initial state from which to plan to the next.

We demonstrate, via modifications to the FF and LAMA algorithms, the effectiveness of using each of these methods of abstraction on planning benchmark problems. In this study, we specifically study the improvements that can be made *once* abstraction has been performed. The abstractions we use are relatively simplistic, however, we demonstrate that even without sophisticated abstraction techniques, performance is still improved.

The rest of the paper is organised as follows: the Background section discusses related work and terminology, in the next section (Abstracting Planning Problems) we discuss our definition of abstractions in planning, and in the next two sections, we analyse the abstract and refine and the abstract and conquer algorithms. We finish with some reflections and conclusions.

Background

There have been several attempts to exploit abstraction in planning. Sacerdoti (1973) developed ABSTRIPS, a solver that exploits a hierarchy of layers of abstraction, each removing detail from the model in the layer beneath it. The detail is removed by dropping preconditions from actions, leading to a relaxation of the original problem.

Yang, Tenenberg and Woods (1991) also explored abstraction in planning, identifying the importance of the "upward solution property", by which abstraction preserves solution existence. This property highlights the fact that abstraction is not merely the removal of structure from a problem: some structure represents constraints, but other structure can represent resources necessary for the problem to be solvable. Knoblock, Tenenberg and Yang (1991) identify the "monotonicity property" in which solutions to the original problem have abstracted forms in the abstracted problems and the "ordered monotonicity property", which holds when a plan is structurally unchanged at any layer of an abstraction hierarchy. All these properties are concerned with solution propagation up abstraction hierarchies; an interesting downward-looking property is the "downward refinement property" defined by Bacchus and Yang (1991) that holds when all solutions in abstract space can be refined into less abstract solutions without needing to backtrack across the abstraction hierarchy. If this does not hold, it is unlikely that the abstraction mechanism will yield results superior to those exhibited in non-abstraction systems.

More recently, both heuristic and optimal (Edelkamp 2002; Haslum et al. 2007) planning approaches using pattern databases have used abstractions as the basis for heuris-

tic guidance. The merge-and-shrink approach (Helmert, Haslum, and Hoffmann 2007) to construction of effective heuristics for planning also exploits a form of abstraction, in which domain transition graphs are multiplied together (to merge them) and then shrunk by abstracting nodes that share the same relaxed distance to the goal.

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*) (Sturtevant and Buro 2005) algorithm for path planning with abstractions. A crucial factor of their work is that the structure of the abstract solutions they find forms the basis of the solutions. In this work, we emulate this idea in domain-independent planning: we find abstract plans, and then use them to form the basis of concrete plans.

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. Given a variable V, we denote the domain of V by $D(V)$. An operator in SAS+ encodes a valid transition between values of the variables, possibly dependent on separate values.

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

$\langle V, v \rangle \in \mathcal{I}$ is an assignment that is true before and after application of O. This is known as a prevail condition.

$\langle V, u, v \rangle \in \mathcal{T}$ is defined in the following way: V is a variable which has the value u before application of O, and value v after application of O. This is known as a pre/post condition.

c is the cost of applying O.

Recent work by Helmert (2009) has led to the development of a translation from a significant fragment of PDDL into SAS+, together with the construction of domain transition graphs, which are important relational structures over the domains of the SAS+ variables.

Definition 2 (SAS$^+$ Planning Task)**.** *A planning problem is defined as 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 $\mathcal{V}$,

$s_\star$ is a set of goal assignments to a subset of $\mathcal{V}$.

A solution is a sequence of actions $a_0, ..., a_l$ in which the final state satisfies $s_\star$.

The domain transition graph (DTG) for any given SAS+ variable is a graph containing nodes that represent the possible values of the variable and directed edges that represent the possible transitions between values induced by legal actions in the domain. In general, 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 action type. In these cases, the DTGs capture an underlying accessibility graph between values in the variable domains, equivalent to maps over which the variables can be conceived as moving according to the legal transitions. This observation is equivalent to the view argued by Long and Fox (2000) who identified common generic types of behaviour across multiple planning domains.

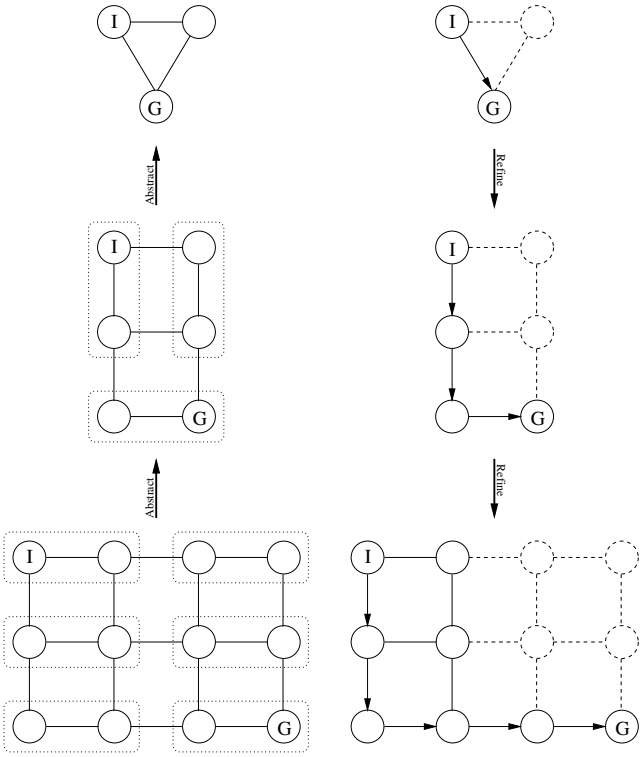

Figure 1: An example of an abstract and refine process. On the left, nodes bordered by dotted lines are abstracted in the next level. On the right, each abstract problem is solved, from top to bottom. Dotted nodes and edges are removed from lower levels.

Abstracting Planning Problems

In order to define an abstract planning problem, we first define an abstract SAS$^+$ variable.

Definition 3 (Abstract SAS$^+$ Variable)**.** *Given a SAS$^+$ variable V, an abstract variable, V', is defined as a SAS$^+$ variable, and a function f_V, where:*

$|D(V)| \leq |D(V')|$

$f_V : D(V) \rightarrow D(V')$ is a surjective function

We call f_V an abstraction function. We say that two values, u and v, from the domain of V have been abstracted together if $f_V(u) = f_V(v)$. Following this definition, an abstract planning problem can be defined as a planning problem, where all of the variables are replaced by abstract variables, and all values in the operators are replaced by the abstract values.

Definition 4 (Abstract SAS$^+$ Planning Task)**.** *Given a planning task, $P = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$, and a set of abstraction functions, f_V, for all $V \in \mathcal{V}$, an abstract planning task is defined*

as $P' = \langle \mathcal{V}', \mathcal{O}', s'_0, s'_\star \rangle$, where:

$\mathcal{V}'$ is a set of finite-domain SAS$^+$ variables abstracting $\mathcal{V}$,

For all operators in $O \in \mathcal{O}$, there exists an operator $O' \in \mathcal{O}'$ such that for all prevails $\langle V, v \rangle$ in O, there is a corresponding prevail $\langle V', f_V(v) \rangle$ in O' and for all pre/post conditions $\langle V, u, v \rangle$ in O, there is a corresponding pre/post condition $\langle V', f_V(u), f_V(v) \rangle$ in O'.

For all $\langle V, v \rangle \in s_0$ ($\langle V, v \rangle \in s_\star$), there is a corresponding $\langle V', f_V(v) \rangle \in s'_0$ ($\langle V', f_V(v) \rangle \in s'_\star$), respectively.

In the example of path planning, adjacent locations are abstracted into the same abstract node. Clearly, abstracting disconnected nodes would create a poor abstraction, as an abstract solution would almost certainly not retain the downward-refinement property. This is also true of abstractions in planning: a property that we desire is that abstracted values should be local to each other. One way in which we can achieve this goal is to base all of our abstractions on the domain transition graphs of the variables.

Different Levels of DTG Abstractions

There are many different restrictions we can place on what forms a valid abstraction. How restrictive the abstraction forms a trade-off: too restrictive and there will be only limited scope for performing abstraction, too liberal and the downward-refinement property will be lost.

The least restrictive abstractions will allow any values to be abstracted together. This is almost certainly of no use, for reasons discussed previously. We now identify several restrictions that can be placed on which values can be abstracted in a planning problem. The list is not intended to be exhaustive, simply indicative of the types of choices that are needed in order to perform abstraction.

Homogeneous DTG Restriction Only values from DTGs containing a single action-type can be abstracted together. This is a very restrictive definition, but also highly-likely to retain the downward refinement property.

Single Value Restriction Given a variable, V, a pair of values, u and v, can be abstracted together if there exists a SAS$^+$ operator which has exactly one pre/post condition, $\langle V, u, v \rangle$, with no restrictions on its prevail conditions.

Consistent Multi-Value Restriction Given a variable, V, a pair of values, u and v, can be abstracted together if there exists a SAS$^+$ operator which has a pre/post conditions $\langle V, u, v \rangle$ with no restrictions on prevail conditions. For all other pre/post conditions $\langle V', u', v' \rangle$, u' and v' *must* be abstracted together.

Inconsistent Multi-Value Restriction Given a variable, V, a pair of values, u and v, can be abstracted together given the following condition: there exists a SAS$^+$ operator which has a pre/post conditions $\langle V, u, v \rangle$ with no restrictions on prevail conditions.

As well as deciding which values can be pairwise abstracted together, it is important to decide what kinds of structures can be abstracted together into abstract nodes. In PRA$\star$, cliques in the graph are are the structures used to form abstract nodes. Using cliques has the desirable property that all concrete points within the abstract node are reachable directly from all other points in the abstract node. Other methods for abstracting values could be based on properties (articulation points, for example) of the DTGs, or clustering techniques. In the work that follows, the abstraction that we use is always clique-based abstraction, with maximum size clique of size two. Meaning, we abstract two values, u and v, together only if there are two actions containing the pre/post condition $\langle V, u, v \rangle$ and $\langle V, v, u \rangle$. We also only consider the Single Value Restriction, as described above.

Once a planning problem is abstracted, there are several ways in which to exploit the result. We explore two of these in the following sections. The first we call abstract and refine, the second we call abstract and conquer. The first method parallels the PRA* algorithm already discussed. The second is a divide and conquer approach, where a single abstract plan is used to create a sequence of planning problems.

Abstract and Refine

As the number of locations is increased, the performance of many planners degrades exponentially — this is true, for example, of those planners based on FF. To a human problem-solver, the increased number of locations does not make the problem of delivering two packages significantly more difficult. This is because the human problem-solver can abstract the problem, and focus on the core elements.

We hypothesise that an abstraction and refinement approach can be used to improve the scaling performance of a planner on problems with large underlying graphs, such as the Driverlog-1,1,2 problem family. We now briefly describe the algorithm we use to plan using abstraction and refinement and then follow this description with a detailed discussion of each stage.

Abstract and Refine Algorithm

In order to define the abstract and refine algorithm, we introduce the natural concepts of *level of abstraction* and *refinement*.

Definition 5 (Abstraction Hierarchy)**.** *Given a planning task P, an abstraction hierarchy is a sequence of planning tasks, $P, P'_1, ..., P'_n$, where P'_1 is an abstraction of P, and P'_i is an abstraction of P'_{i-1}. P'_n is a fixpoint, in that the result of abstracting P'_n equals P'_n.*

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

$\mathcal{O}_R$ contains $\langle \mathcal{I}, \mathcal{T}, c \rangle \in \mathcal{O}$ if:

for all $\langle V, v \rangle \in \mathcal{I}$, $f_V(v)$ occurs in $F(\pi')$

for all $\langle V, u, v \rangle \in \mathcal{T}$, $f_V(u)$ and $f_V(v)$ occur in $F(\pi')$

The way in which we determine the search starting point within the abstraction hierarchy is important. The effort of abstraction is wasted if search is initiated at too low a level but, conversely, if search begins at too high a level, the time taken to propagate through layers will negates the abstraction benefit. We follow the proposal made by Sturtevant and

Buro (2005) to begin search at the middle layer in the abstraction hierarchy.

Once the problem is solved at an abstract level, the plan generated at that level is used to guide the search for a plan at the next level down the hierarchy. This is achieved by considering the values in the actions of the high level plan and removing the values in the problem at the new level of abstraction that appear in the abstracted problem but do not correspond to values used in the solution to the abstracted problem.

We can now formally define the abstract and refine algorithm.

Abstract and Refine Algorithm

Given a planning task P:

1. Let $P, P'_1, ..., P'_n$ be the abstraction hierarchy for P.
2. Let $c = \lfloor n/2 \rfloor$
3. Repeat, until $c = 0$:

(a) Let π' be a solution to P'_c,

(b) Let π'_c be the refinement of π' and P'_{c-1}

4. Return π_c

Algorithm Properties

Often, SAS+ variables encode the locatedness of some object in a planning problem. Without loss of generality, our discussion of algorithm properties focusses on this type of variable. When locations are merged in the abstraction process, their corresponding properties are merged. For example, vehicles at concrete locations now appear at the corresponding abstract location. However, these locations that are abstracted together could have properties that are mutually exclusive in the original problem.

An example is the Grid domain, in which some locations are locked and accessible only with a correct key. If these locations are merged with others that are not locked then then the abstract location will be both locked and unlocked. At the abstract level, these locations will be considered unlocked for the purposes of planning a path and plans will not

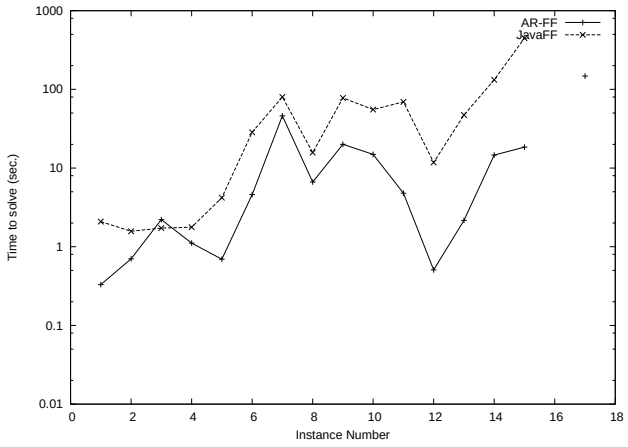

Figure 2: Time to plan in the Roadlog domain.

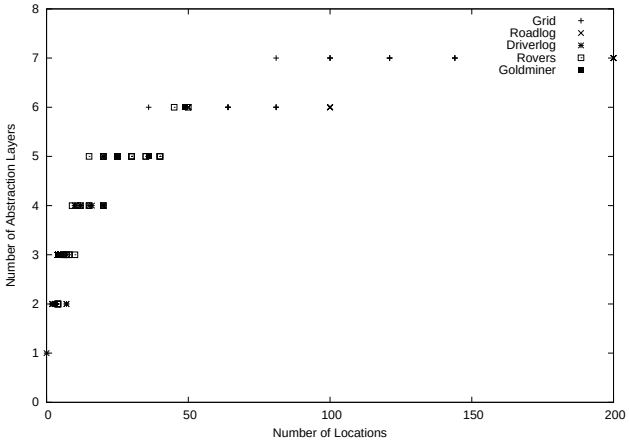

Figure 3: Number of layers in the abstraction stacks of the test instances.

necessarily include paths that visit the locations where keys are stored for the locked locations that might actually appear on the path once it is refined. This problem demonstrates that the abstraction process we have proposed does not preserve completeness and, in fact, does not have the downward refinement property: backtracking across abstraction layers is necessary if a solution is to be found in this case.

This problem does not affect all domains, of course. Classical transportation domains, such as Driverlog and Logistics, do not have mutually exclusive properties attached to locations that can be merged. It is possible that domain analysis could inform the abstraction process preventing the merging of objects with mutually exclusive properties. This is not implemented in our current system. We present results below for domains (Grid and Goldminer) in which the incompleteness of our algorithm causes potential problems, in order to determine whether these problems arise in practice.

Empirical Evaluation

We compare the performance of Abstract-and-Refine FF (AR-FF from here) with that of JavaFF (Coles et al. 2008) (a reimplementation of FF in Java, useful as a teaching tool). Since AR-FF is built directly on JavaFF, this comparison highlights the effect of the abstract and refine process most effectively. We provide results for a range of domains, each containing problems with different structures. All of our experiments are run on a desktop PC, with an Intel 3.16GHz Dual Core CPU, a 2GB memory limit and a 15 minute cutoff.

We present results for the five domains Roadlog, Driverlog, Rovers, Goldminer and Grid. Driverlog and Rovers were introduced at the third IPC (Long and Fox 2003). The Driverlog instances are the competition instances. The Rovers instances are a combination of the competition instances and also the so-called 'hand coded' competition instances, originally intended for planners which use human domain knowledge. The Roadlog domain is a simplification of the Driverlog domain in which the drivers are removed,

and therefore the trucks are autonomous. The Goldminer domain was created for the learning track of the sixth IPC. We generated a collection of Grid instances ourselves as the competition benchmark set (McDermott 2000) contains only five instances. These five instances are the first of those presented in our results. The remainder were generated using the instance generator from the FF domain set.

The speed of AR-FF is determined by two opposing factors: solving several different abstract planning problems slows search down, while refining planning problems reduces the size of the state-space that must be searched, which reduces the time to solve each problem. Figure 3 shows the how the number of abstraction layers scale as the number of locations in the benchmark set increases. The number of layers scales approximately logarithmically as the number of locations is increased. This seems stable across domains.

The process of refinement removes choices for the planner at each level of the abstraction. This creates a second trade-off in the use of the abstraction approach: the abstraction-based search can speed up the discovery of a solution, but the pruned search space can lead to the loss of some high quality solutions, so that the final result is a lower quality solution than could be found by direct search in the original search space.

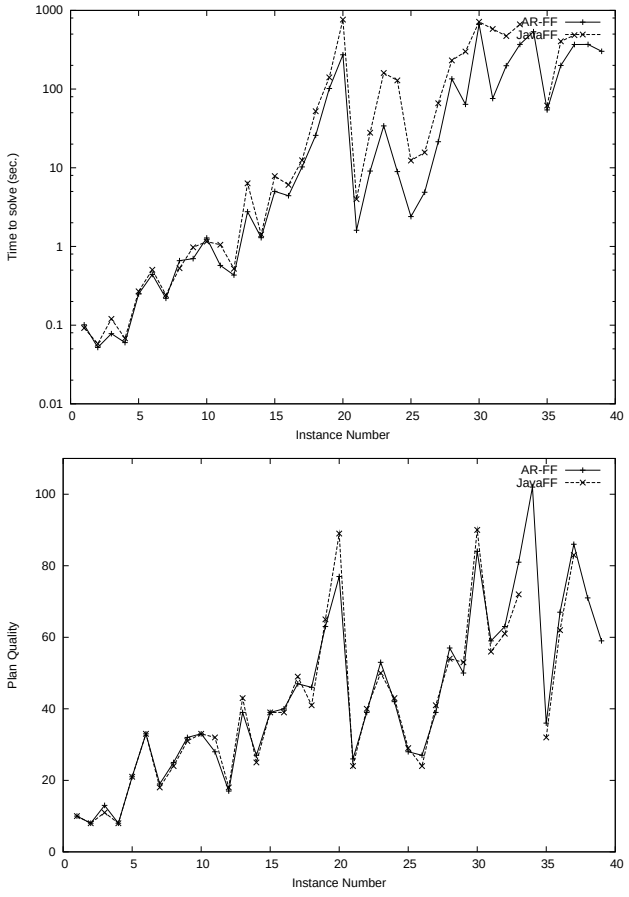

Figure 4: Time to plan (top graph) and Quality of plan (bottom graph) in the Rovers domain.

Domain	AR-FF	JavaFF	Equal
Rovers	15	17	8
Driverlog	4	4	12
Roadlog	6	8	6
Total	25	29	26

Table 1: Plan quality summary across three domains, showing the number of instances for which each planner produced the highest quality plan.

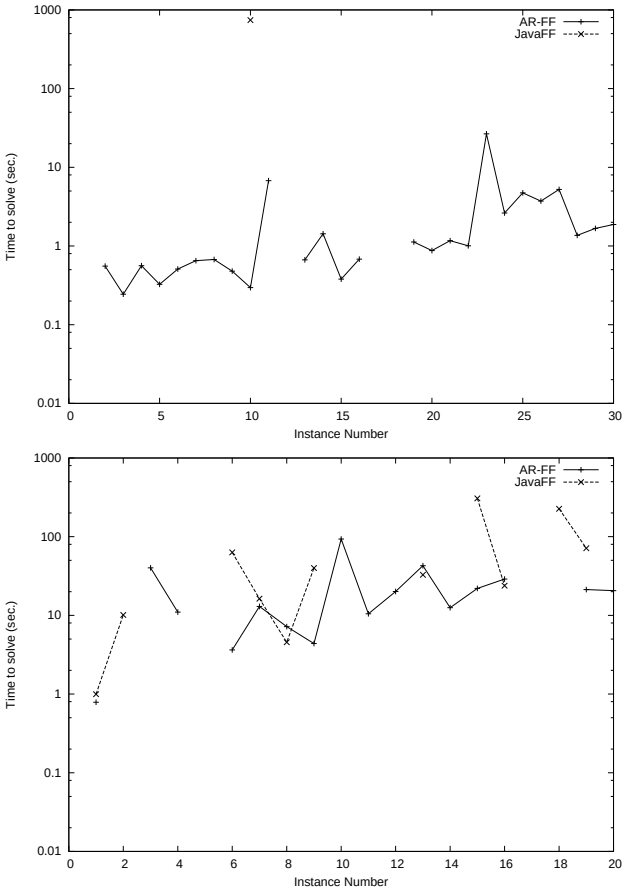

Figure 5: Time to plan in the Goldminer domain (top) and the Grid domain (bottom).

In the Roadlog domain, Figure 2 shows that time performance is improved. We show that in the Rovers domain (Figure 4) plans are typically found more quickly, especially in the larger instances. In the Grid and Goldminer domains (Figure 5), abstraction means that more instances are solved than before. Table 1 shows that the improvements in time do not come at great cost in terms of plan quality.

When discussing the incompleteness of the AR-FF algorithm, we identified two domains for which the abstraction process does no guarantee the downward refinement prop-

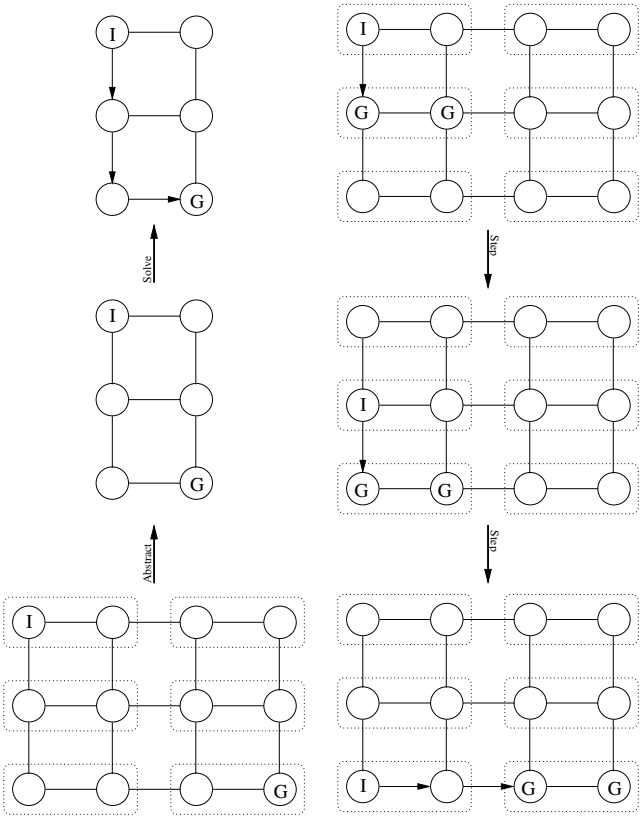

Figure 6: An example of an abstract and conquer process. On the left, the base problem is abstracted once. The resulting problem is solved, providing the plan at the top level. On the right, each abstract action defines a new planning problem, these are solved sequentially from top to bottom.

erty and, hence, could lead to failure to solve instances (we do not backtrack across the abstraction layers so once a poor abstract solution is generated as the basis for refinement, no solution will be found). These domains are Goldminer and Grid; the results for these domains are shown in Figure 5. AR-FF fails to solve four of the 30 Goldminer instances. All of these failures were due to problems with incompleteness. Despite this problem, AR-FF solves 25 more instances overall than JavaFF. In this domain at least, the advantage gained in scalability improvements outweighs the disadvantage presented by the incompleteness. Nevertheless, it is clear that further mechanisms are required to avoid the incompleteness we have noted.

Abstract and Conquer

Another way to exploit abstraction in planning is by using what we call an *abstract and conquer* approach. In this approach, an abstract plan is found, and the abstract effects of each action are used as intermediary goals. This is similar to the STeLLa planner (Sebastia, Onaindia, and Marzal 2002) which uses landmarks in order to create a sequence of planning problems.

A picture of the abstract and conquer algorithm can be seen in Figure 6. The problem is the same as in Figure 1. In this example, the abstract solution shown at the top-left is used as a skeleton for the concrete plan. On the right-hand side of the diagram, we see the seqeuential search. For each action in the abstract plan, there is a corresponding planning problem. The initial state of this problem is the final *concrete* state of the previous plan, and the goal state is the *abstract* effect of the action. This abstract goal corresponds to a disjunctive concrete goal, as denoted by the multiple goals in the graph.

To define the algorithm more formally, we rely on the definition of what we call the *augmented* planning task, which we is loosely defined as a combination of both an abstract and a concrete version of the same problem. The goal is the concrete goal. This definition is crucial to the working of the abstract and conquer algorithm detailed later in the paper.

Definition 7 (Augmented SAS$^+$ Planning Task)**.** *Given a planning task, $P = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$, and an abstract planning task $P' = \langle \mathcal{V}', \mathcal{O}', s'_0, s'_\star \rangle$, an augmented planning task is defined as $P^+ = \langle \mathcal{V}^+, \mathcal{O}^+, s^+_0, s^+_\star \rangle$, where:*

$\mathcal{V}^+ = \mathcal{V} \cup \mathcal{V}'$,

For all $\langle \mathcal{I}, \mathcal{T}, c \rangle \in \mathcal{O}$ and corresponding $\langle \mathcal{I}', \mathcal{T}', c \rangle$, $O^+ = \langle \mathcal{I} \cup \mathcal{I}', \mathcal{T} \cup \mathcal{T}' \rangle, c$

$s^+_0 = s_0 \cup s'_0$

$s^+_\star = s_\star$

More precisely, the algorithm works as follows:

Abstract and Conquer Algorithm

Given a planning task, P:

1. Let P' be an abstraction of P.
2. Let $\langle \mathcal{V}^+, \mathcal{O}^+, s^+_0, s_\star \rangle$ be the augmentation of P and P'.
3. Let π' be a plan for P', let π be the empty plan, let $s_c = s^+_0$.
4. For all $O' \in \pi'$:

(a) Let $\mathbf{effs}(O')$ be the effects of O'

(b) Let π^+ be a plan for $\langle \mathcal{V}^+, \mathcal{O}^+, s_c, \mathbf{effs}(O') \rangle$

(c) Let $\pi_c = \pi_c + \pi^+$, let s_c be the end state of π^+

5. Let π^+ be a plan for $\langle \mathcal{V}^+, \mathcal{O}^+, s_c, s_\star \rangle$.
6. Return $\pi_c + \pi^+$

Algorithm Properties

One clear advantage of the abstract and conquer algorithm over the previously discussed abstract and refine is that the problems associated with downward-refinement now disappear. This is a straightforward consequence of the fact that 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 for each abstract step. The final step in the algorithm ensures the concrete goals are met by asserting them as new goals. In the best scenario, there will be nothing to do in this step and the abstract plan will have been an effective guide to the solution. In the worst-case, too much information will be lost in the abstraction and there will be lots of extra work to be performed.

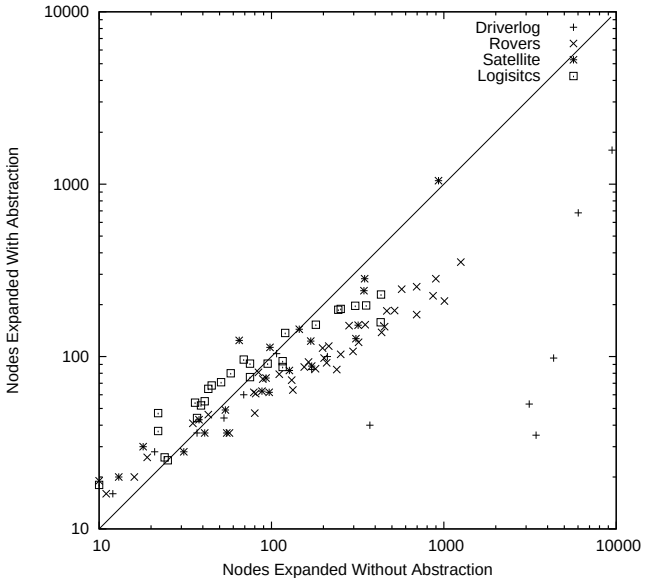

Figure 7: Nodes expanded in LAMA to find first solution with Abstract and Conquer approach (log-log-scaled).

However, there is still a potential problem with incompleteness that could be resolved by backtracking. The problem occurs if, in achieving an intermediary abstract goal, the plan moves into a dead-end. There may be some resource rendered unreachable that is not used in the abstract plan, but is needed to achieve the concrete part of the problem. Our implementation does not handle this possibility and simply fails to find a plan if a dead-end is encountered; this eventuality has not occurred in any of our testing.

Empirical Analysis

Our implementation of abstract and conquer is based on the LAMA (Richter, Helmert, and Westphal 2008) planner, constructed as a very simple harness around the based planner. The implementation creates a SAS+ instance for each of the individual steps which is then fed back to a new invocation of the planner. As a consequence of this straightforward approach to implementation, the majority of execution time is spent writing sub-problem instances to file and in LAMA reading them in again. Because of this, exact run-times do not provide an accurate reflection of the difficulty of planning with this approach. However, LAMA reports the number of search nodes that are expanded in finding the solution and this provides a measure that more accurately reflects the difficulty of search. (A more integrated implementation remains a short-term goal for future work).

Figure 7 shows the number of nodes expanded in finding the first solution when using abstract and conquer against when not using the approach. There is a clear trend that the greater the number of nodes expanded to solve the problem, the greater the advantage gained from using abstraction. Note that the graph is log-scaled, so the benefits are actually favour the abstraction approach with exponential improvements.

FUTURE WORK

By abstracting together only DTG values that change single variables, we restrict ourselves to performing abstractions over graph-like structures, such as road networks. Thus, it could be argued that the approach is only beneficial in these domains. At present, there is little performance penalty in terms of time when abstraction produces no interesting results, and there is no penalty in terms of nodes. Secondly, many planning domains contain graph-like structure, and so approaches to scale better on these problems is useful. However, we believe that planning domains containing more complex structure can still benefit from our abstraction approaches, by using less restrictive value abstractions.

As noted earlier, there are mitigating steps that can be taken to avoid merging DTG states that have conflicting properties. Following these steps can help to avoid the problems associated with the incompleteness of the algorithm.

The contribution of this work is to show that an abstract and refine approach can improve the performance of a planner, and our current system demonstrates that. The abstract and refine approach is, however, completely general. We intend to create a planner-independent version of the abstract and refine algorithm. We can then hope to improve the scalability of other planners.

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 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 and, we believe, can achieve a similar performance in the time taken to solve problems once the implementation is fully integrated. Nevertheless, one of the attractions of the abstract and conquer approach is that it can be easily attached to any underlying planning system. We believe that 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 *IJCAI'91: Proceedings of the 12th international joint conference on Artificial intelligence*, 286–292. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

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, Twenty-Third AAAI Conference on Artificial Intelligence*.

Edelkamp, S. 2002. Symbolic pattern databases in heuristic search planning. In Ghallab, M.; Hertzberg, J.; and Traverso, P., eds., *AIPS*, 274–283. AAAI.

Fox, M., and Long, D. 2003. PDDL2.1: An extension of PDDL for expressing temporal planning domains. *Journal of AI Research* 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 *AAAI*, 1007–1012. AAAI Press.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In Boddy, M. S.; Fox, M.; and Thiébaux, S., eds., *ICAPS*, 176–183. AAAI.

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

Hoffmann, J. 2001. FF: The Fast-Forward Planning System. *AI Magazine* 22(3):57–62.

Knoblock, C. A.; Tenenberg, J. D.; and Yang, Q. 1991. Characterizing abstraction hierarchies for planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, volume 2, 692–697. Anaheim, California, USA: AAAI Press/MIT Press.

Long, D., and Fox, M. 2000. Automatic synthesis and use of generic types in planning. In *AIPS*, 196–205.

Long, D., and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *Journal of AI Research* 20:1–59.

McDermott, D. 2000. The 1998 AI Planning Systems Competition. *AI Magazine* 21(2):35–56.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks Revisited. In *AAAI*, 975–982.

Sacerdoti, E. D. 1973. Planning in a hierarchy of abstraction spaces. In *IJCAI'73: Proceedings of the 3rd international joint conference on Artificial intelligence*, 412–422. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

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

Sturtevant, N. R., and Buro, M. 2005. Partial pathfinding using map abstraction and refinement. In Veloso, M. M., and Kambhampati, S., eds., *AAAI*, 1392–1397. AAAI Press / The MIT Press.

Yang, Q.; Tenenberg, J. D.; and Woods, S. 1991. Abstraction in nonlinear planning. Technical Report CS-92-32, University of Waterloo.