On The Extraction of Disjunctive Landmarks from Planning Problems via Symmetry Reduction *

Peter Gregory & Stephen Cresswell
& **Derek Long** & **Julie Porteous**
University of Strathclyde
Glasgow, UK
{firstname.lastname}@cis.strath.ac.uk

Abstract

The exploitation of symmetry in combinatorial search problems has typically focussed on using information about symmetries to control search. This work describes an approach that exploits symmetry to get more detailed domain-analysis rather than as a method of search-control. We describe both the concepts of atomic and disjunctive landmarks in planning (a concept analogous to the backbone of a SAT/CP problem) and algorithms for extracting them. We categorise various types of symmetry encountered in planning, and show examples of each on actual problems. We introduce the Orlando API, a tool that extracts disjunctive landmarks using symmetry-breaking techniques and atomic landmarks analysis in concert. This provides a polynomial-time algorithm for extracting landmarks with respect to any equivalence function. We introduce the standard equivalence functions supplied with Orlando and show situations in which these are useful.

1 Introduction

Many different research communities have seen the attraction of exploiting symmetry in problems. Symmetry seems to be a characteristic of many combinatorial search problems and has been examined in the CP [Joslin and Roy, 1997; Gent and Smith, 2000], planning [Fox and Long, 1999; 2002] and model-checking [Ip and Dill, 1996; Clarke *et al.*, 1996; Emerson and Sistla, 1996] communities. Much effort has been spent in finding ways break symmetries during search. This work, however, details a method of gaining a rich domain-analysis by analysing *symmetrically reduced problems*. Symmetric reduction is performed by casting the properties of all the objects in a symmetric group on to a representative member of that group.

Landmarks are facts that occur in every valid solution to a planning problem [Porteous *et al.*, 2001]. Atomic landmarks are single literals. There are often no atomic landmarks in problems with symmetry in them. Disjunctive landmarks are disjunctions of literals that represent a set of facts, any of which could occur in a valid plan [Porteous and Cresswell, 2002]. These disjunctions arise because of the symmetry in problems. If the problems are first symmetrically reduced then atomic landmarks can be extracted from the new problem and interpreted as disjunctive landmarks. This method is faster than other ways of extracting disjunctive landmarks, it also attaches more meaning to the landmarks we extract.

In Section 2 we will briefly introduce planning problems. We will discuss atomic landmarks, an algorithm for extracting them and give reasons for their weakness. We discuss how symmetry arises in planning problems, and how it renders atomic landmarks analysis useless. We then discuss disjunctive landmarks, along with an algorithm to extract them.

Section 3 discusses the Orlando API. The approach of extracting disjunctive landmarks using a combination of symmetry reduction and atomic landmarks analysis is detailed. The equivalence functions that are supplied with Orlando as standard are described. The use of landmarks analysis in several planners is discussed in Section 4. Also discussed are planners that make use of symmetry-breaking (although not in concert with landmarks analysis.)

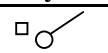
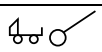
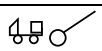
2 Planning Problems

A planning problem is composed of a set of actions, an initial state and a goal formula. An action is itself composed of a condition formula that must be satisfied for successful application of the action, and an effect that modifies the state in some way. In today's planning formalisms [Fox and Long, 2003], it is possible to represent problems with a very rich structure. Problems with temporal structure, numeric resources and conditional effects amongst others are representable. The conditions of an action can be pre- or post-conditions to an action or can be an invariant that must hold over the duration of the action. The initial state is a conjunction of atomic predicates. The goal is a formula that, when satisfied, indicates a valid plan.

This paper, for reasons of clarity, will only consider a subset of this expressivity. We consider actions to have only pre-conditions (and not post-conditions and state-invariants). These pre-conditions will take the form of a conjunction of atomic predicates. Also, the effects are represented as two sets of atomic predicates; the add-list and the delete-list. When an action is applied, the facts in the add-list are added, and those in the delete-list are removed, from the current

*This work is part funded by EPSRC grant GR/S11015/01.

state. The goal formula is also assumed to be a conjunction of atomic predicates.

Predicate	**Key**
`( at parcel location )`	
`( at truck location )`	
`( road loc1 loc2 )`	
`( in parcel truck )`	

(a) The predicates.

:action	load
:parameters	?parcel ?truck ?location
:preconditions	(at ?parcel ?location) (at ?truck ?location)
:add-effects	(in ?parcel ?truck)
:del-effects	(at ?parcel ?location)
:action	unload
:parameters	?parcel ?truck ?location
:preconditions	(in ?parcel ?truck) (at ?truck ?location)
:add-effects	(at ?parcel ?location)
:del-effects	(in ?parcel ?truck)
:action	drive
:parameters	?truck ?loc1 ?loc2
:preconditions	(at ?truck ?loc1) (road ?truck ?location)
:add-effects	(at ?truck ?loc2)
:del-effects	(at ?truck ?loc1)

(b) The Action Schema

Table 1: The domain description of a logistics domain, with diagrammatic key, where appropriate

Table 1 describes a logistics domain used throughout this paper for explanatory purposes. It is intentionally simplistic so that clear examples of important concepts can be shown throughout the paper.

2.1 Atomic Landmarks Analysis

A *landmark* can be informally described as a fact that occurs in every valid solution to a problem [Porteous *et al.*, 2001]. The concept of landmarks in a planning problem is closely related to that of the *backbone* in the general CP/SAT problem. The main difference between the two is that the backbone is described as the set of fixed variables in the *optimal* solution [Slaney and Walsh, 2001]; landmarks are the facts that occur in *all* solutions.

There are algorithms in the literature [Porteous *et al.*, 2001; Zhu and Givan, 2003] for extracting atomic landmarks from a planning problem. The approach we use is described in 2. Contrary to the method employed in [Porteous *et al.*, 2001] the algorithm does not need to reconstruct an RPG to test each literal. The concept of an RPG (*relaxed plan graph* is described in [Hoffmann, 2003]. The important aspect of the RPG for this work is that it is constructed in polynomial time.

We associate a set of *ancestors* with each literal and each action. The ancestors of a fact or action are literals necessarily visited in reaching it. These are carried through the process of RPG construction. We obtain the necessary ancestors of an action at a given level by computing the union of the ancestors of its preconditions. We obtain the necessary ancestors of a fact at a given level by computing the intersection of the ancestors of its achieving actions. Trivially, the initial and goal states are landmarks. The algorithm is detailed in Figure 2.

```
Advance through Level until no change to
action ancestors set

   For each Literal reachable at Level
      Ancestors(Literal,Level):=
         intersection of ancestors of
         achievers of Literal
            reachable at Level-1

   For each Action reachable at Level
      Ancestors(Action,Level):=
         Union of ancestors of
         preconditions(Action)
            reachable at Level

Output ancestors of the top-level goals
as landmarks.
```

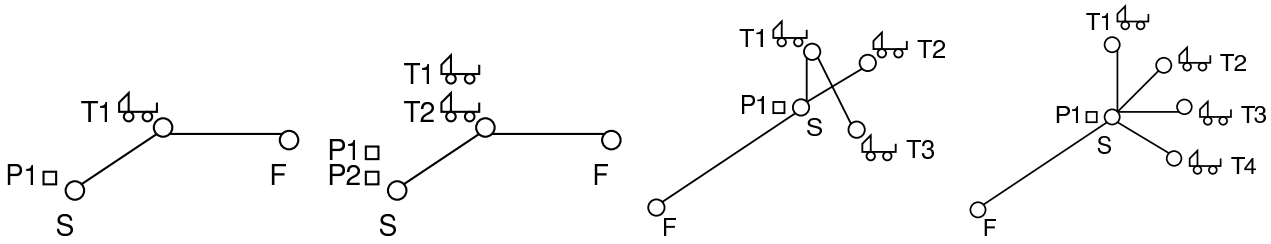
Figure 2: Algorithm for extracting atomic landmarks.

If the parcel in Figure 1(a) needs to be delivered to location G then atomic landmarks analysis would find the landmarks `(at T1 S)`, `(in P1 T1)` and `(at T1 G)` other than the facts in the initial and goal state. These facts, along with their orderings, are almost enough to infer a plan.

Atomic landmarks analysis is weak in situations like Figure 1(b). Because each parcel could be delivered by either T1 or T2, the landmarks analysis counts neither `(in P1 T1)` nor `(in P1 T2)` as landmarks. The *symmetry* in the problem means that the algorithm is unable to find any landmarks.

2.2 Symmetries in Planning

The literature identifies several types of symmetry that arise in planning and CSP problems [Fox and Long, 1999; Long and Fox, 2003b; Joslin and Roy, 1997]. Techniques to break symmetry in search have been identified and implemented in several different planners [Fox and Long, 2002; Chen *et al.*, 2004; Edelkamp and Helmert, 2000]. General symmetry-breaking approaches have also been studied in the CP[Joslin and Roy, 1997; Gent and Smith, 2000] and model-checking communities [Ip and Dill, 1996; Clarke *et al.*, 1996; Emerson and Sistla, 1996]. The CP community has also considered static symmetry-breaking constraints. The CP lan-



(a) Example of a problem with no symmetry. Atomic landmarks analysis extracts all landmarks.

(b) Example of a problem with functional equivalences. The trucks, T1 and T2, are functionally equivalent, as are the parcels P1 and P2.

(c) Example of a problem with almost-symmetry. The trucks are each clustered about location S but are not functionally equivalent.

(d) Example of a problem with structural symmetry. This does not constitute functional equivalence since it is the composite structures of 'truck-and-road' that are symmetric.

Figure 1: Pedagogical examples of problems from the simple logistics domain defined in Table 1. In all problems, each parcel is initially located at S; the goal is that each parcel is delivered to location F. Atomic landmarks analysis extracts all landmarks in (a) but none from the rest.

guage eclipse is packaged with a SBDS (symmetry-breaking during search) library and the model-checking tool Spin include a symmetry package 'SymmSpin'.

We now describe four major categories of symmetries that arise in planning problems. Illustrations drawn from the simple logistics domain demonstrate an example of each:

Functional Equivalences: Two objects are functionally equivalent if they play exactly the same role in the initial (or current) and goal states of a planning problem and are not distinguished in any actions. In this case, it is only the names of the objects that distinguishes the objects and they can be freely permuted in any plan to yield a plan that achieves the same goals [Fox and Long, 1999]. Each of the trucks (parcels) in Figure 1(b) are functionally equivalent. In any valid plan, the roles of T1 and T2 could be reversed and the new plan would remain sound and would be equivalent to the original.

Functional identities are the most commonly exploited type of symmetry in planning. Typically, the only exploited identities are those which can be derived from the initial and goal states. [Fox and Long, 2002; Long and Fox, 2003b] used an innovative approach to keep track of the symmetric groups during search to take advantage of symmetries that arise during search.

Plan Permutation: Two plan fragments are symmetric when they achieve the same facts from the same state [Long and Fox, 2003b]. A successful plan for the problem in Figure 1(b) could include the plan fragment `(load P1 T1) ; (load P2 T1)`. It could also include the fragment `(load P2 T1) ; (load P1 T1)`, so these fragments are an example of plan permutation symmetry.

Any two non-interfering actions executed sequentially may be reversed. Any n non-interfering actions executed sequentially can be permuted in any of $n!$ ways. More generally, certain collections of actions will be equivalent under particular permutation operations (that is, symmetries) and will therefore play equivalent roles in any plans that contain them. As with other types of symmetry, exponential growth in the search-space is possible. Some preliminary results are presented in [Long and Fox, 2003a] for exploitation of this form of symmetry in a Graphplan planner (STAN).

Almost Symmetry: Symmetry arises as a consequence of abstraction in the representation of a problem. In many planning problems, potential symmetries are broken by elements of the problem description that are not sufficiently abstracted. In some cases, it is possible to apply an abstraction to a problem and thereby create opportunities for exploitation of symmetries that are exposed. The key to this approach is to find abstractions that lead to problems for which the solutions remain direct and relevant guides to the solutions of the original problems. A simple abstraction is to remove information from the domain. If the information is irrelevant to the problem then its removal will not prevent a solution from being found and the solution remains (trivially) a solution of the original problem.

A more sophisticated abstraction is to ignore certain initial conditions. An example of this idea can be applied to the example in Figure 1(c). The trucks clustered around the initial location of the parcel are *not* functionally equivalent, but since the goal location is so distant the small differences in location at the clustered side could be abstracted out. A plan prefix or postfix can then be applied to resolve the differences between the solution to the abstracted problem and the solution to the original problem.

There are many unresolved challenges in the exploitation of almost symmetry and it remains a current topic of research.

Structural Symmetry: The trucks in Figure 1(d) are equidistant from location S. They are not functionally equivalent because they are connected to S by different roads. It is clear, however, that the structure of each of the initial (Truck,

Location) pairs is symmetric with respect to any of any other initial (Truck, Location) pairs.

These higher-order structures can be found using NAUTY, a tool that finds graph automorphisms. Joslin and Roy [Joslin and Roy, 1997] have illustrated use of this approach. Donaldson et. al. have also used the approach in model-checking [Donaldson *et al.*, 2004].

Much symmetry in planning is clear to a person but is not as uniform as functional equivalence. This is exemplified in the concept of almost symmetry. If symmetry can be defined as a mapping of a structure onto itself then there must be methods of representing these types of abstractions as mappings.

Can these types of symmetry be generalised to CSP? Yes, many problems exhibit these forms of symmetry. Consider any problem where some ordered schedule has to be found; identical jobs can clearly be permuted, a stronger claim could be that any jobs can be permuted so long as the permutation doesn't affect the cost of the schedule. This exhibits plan permutation symmetry. Also, since none of the jobs need be identical, it exhibits almost-symmetry.

All types of symmetry prevent atomic landmarks analysis from performing well. One of the goals of our work is to extend earlier work on landmarks analysis to exploit information on symmetry.

2.3 Disjunctive Landmarks

Atomic landmarks are incapable of representing the landmarks that are present because of the symmetry in problems. Disjunctive landmarks analysis (referred to as 'resource abstracted landmarks' in [Porteous and Cresswell, 2002]) looks to remedy this situation by representing landmarks as disjunctions of facts. If the reader again refers to Figure 1(b) then he will recall that atomic landmarks analysis fails to recover any landmarks.

There are several recoverable disjunctive landmarks in this problem, three examples of which are given in Table 2.

1. `(in P1 T1)` $\vee$ `(in P1 T2)`
2. `(in P2 T1)` $\vee$ `(in P2 T2)`
3. `(in P1 T1)` $\vee$ `(in P1 T2)` $\vee$ `(in P2 T1)` $\vee$ `(in P2 T2)`

Table 2: Disjunctive landmarks from problem in Figure 1(b).

The first two tell us that each of the parcels respectively are required to be in one of the trucks if the goal is to be achieved. The last is more general; it states that either of the parcels must be in either of the trucks. We now introduce an algorithm for extracting disjunctive landmarks based on the algorithm in Figure 2.

2.4 Disjunctive Landmarks via Forwards Propagation

In this section, we extend the forwards algorithm to deal with disjunctive landmarks. The part of the algorithm which changes is the computation of an ancestor set for a literal from ancestor sets of achieving actions. When we consider landmarks to be simple literals, this computation is an intersection of the ancestor sets. A literal has to occur as an ancestor of *every* achieving action.

In the case of disjunctive landmarks, we have the option of selecting one condition from each achiever, and forming a disjunctive set. A problem that we now encounter is that there are very many possible disjunctive sets that could correctly be considered to be ancestors, and may turn out to be landmarks. We need some criteria by which to select which disjunctive sets are worth keeping.

Selecting disjunctions

An obvious criterion to use is to select literals which have the same predicate. However, we need to also impose further restrictions for this to work well.

We now give an example where this simple approach works poorly. Suppose we have a landmark which requires that some truck (truck1 or truck2) has a driver (driver1 or driver2). The disjunction of preconditions of achieving actions would include the following:

$$(at(driver1, loc1) \wedge at(truck1, loc1)...) \vee$$
$$(at(driver2, loc1) \wedge at(truck1, loc1)...) \vee$$
$$(at(driver1, loc1) \wedge at(truck2, loc1)...) \vee$$
$$(at(driver2, loc1) \wedge at(truck2, loc1)...)$$

We would like to get two disjunctive landmarks, requiring that at least one truck and one driver are available.

$$at(driver1, loc1) \vee at(driver2, loc1)$$
$$at(truck1, loc1) \vee at(truck2, loc1)$$

However, if we group according to the common predicate at, two disjunctive landmarks become merged into one:

$$at(driver1, loc1) \vee$$
$$at(driver2, loc1) \vee$$
$$at(truck1, loc1) \vee$$
$$at(truck2, loc1)$$

Hence we need to be able to make a distinction between drivers and trucks which is not obvious from the context. A solution to this problem is to make use of the type data which is generated by analysis of the domain and problem using TIM [Fox and Long, 1998]. Fig. 3 shows part of a type hierarchy that is produced by TIM analysis. Given this information, when we attempt to generalise a disjunctive set, we do so by making the smallest available step towards the root of the type hierarchy. The categorisation of types in TIM makes use of a fingerprint for each object, which is the set of TIM spaces in which the object may participate. TIM spaces group together objects which are functionally equivalent. If we represent types as sets of space identifiers, then for types $Type1$ and $Type2$, the subset relation $Type1 \subset Type2$ means that the type $Type2$ is more specific than $Type1$, and so $Type2$ is subtype of $Type1$. For present purposes, we also regard specific object identities as type information, and where appropriate we add those to the set. As a consequence, we can find a node in the type hierarchy that generalises two types simply by computing the intersection of their signatures.

We represent each ancestor as a pair $\langle Sig, DJset \rangle$, where Sig is a signature which specifies the predicate common to

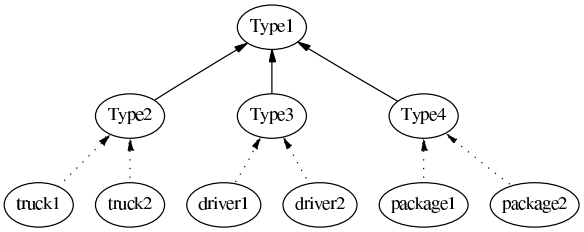


Figure 3: Example type hierarchy

the disjuncts and the types of the arguments common between disjuncts, and $DJset$ is a disjunctive set of literals.

The step in the forwards propagation algorithm in which the intersection of the ancestors of the achievers is computed is replaced by a procedure which uses signatures to derive generalisations where necessary. The generalisation of two pairs is defined as follows:

least_generalisation($\langle Sig0, DJset0\rangle$, $\langle Sig1, DJset1\rangle$):
$Sig2 := intersect\_signatures(Sig0, Sig1)$
$DJset2 := DJset0 \cup DJset1$
return $\langle Sig2, DJset2\rangle$

intersect_signatures($Sig0$, $Sig1$):
$P :=$ predicate of $Sig0$, $Sig1$
$Sig2$ is a new signature with predicate P
For each (N^{th}) arg of $Sig0$
$Arg(Sig2, N) := Arg(Sig0, N) \cap Arg(Sig1, N)$
return $Sig2$

We generate disjunctive landmarks such that each achieving action has at least one of the disjuncts as an ancestor. We keep only the most specific disjunctive sets — i.e. we discard any disjunctive set if its signature subsumes the signature of another disjunctive set.

Our algorithm based on these principles is effective at finding disjunctive landmarks in benchmark problems. However the space requirement is high, and under certain circumstances the generalisation algorithm could generate a number of disjunctive sets which is exponential in the number achievers for a literal. Hence the type information is not always sufficient on its own to select useful disjunctive landmarks.

2.5 Other Algorithms

There is another algorithm described in the literature for extracting disjunctive landmarks from a planning problem. [Porteous and Cresswell, 2002] introduce a different approach based on propagating an RPG backwards. No disjunctive landmarks analysis has yet been used in practical planning.

Here we introduce the Orlando API, a program that we have created to extract and manage disjunctive landmarks. Orlando does not use the algorithm from [Porteous and Cresswell, 2002] or the one in Section 2.4. Rather, the authors have discovered a novel way of combining symmetry-breaking techniques and atomic landmarks analysis to discover disjunctive landmarks.

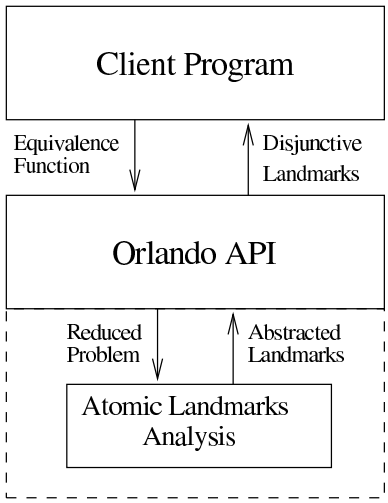


Figure 4: Architecture of Orlando.

3 The Orlando API

Orlando is a tool for extracting and managing disjunctive landmarks. By symmetric reduction, we can reveal disjunctive landmarks using only conventional landmarks analysis. The API takes a planning problem as input; the user can then request landmarks analysis be performed on several abstractions of the original problem. The landmarks obtained from these abstracted problems are equivalent to disjunctive landmarks.

Figure 4 shows the architecture of Orlando. A client program requests that landmarks analysis be performed with respect to some equivalence function. All of the equivalence functions are hard-coded into Orlando and are described below. Orlando reduces the problem and performs atomic landmarks analysis on this reduced problem. A translation of the results are returned to the client program. The Orlando algorithm can be described, in greater detail, in the following steps:

- Find the symmetry groups S of all the objects in the problem with respect to equivalence function F.
- For each symmetry group, choose a representative member object S_r. All occurances of any member of group S in the problem should be replaced by S_r. This stage is the symmetric reduction stage. It is likely that all members of some symmetry group share common functionality. Therefore the new problem will be smaller, and less complex, than the old one; hence *symmetric reduction.*
- Perform conventional landmarks analysis using the algorithm in Figure 2. The landmarks obtained can now be reinterpreted into the original problem.
- Any atomic predicate including S_r is be expanded into a disjunctive landmark. The number of objects in S gives the number of disjuncts in the landmark. For all m in S, create a new disjunct identical to the atomic landmark replacing S_r with m.

This technique exploits symmetry so that we can generalise a problem. This is a novel use of symmetry. Symmetry has been exploited in planning to prune the search space, but here we are proposing to use symmetries to identify a way to structure the search space before exploring it. It is also a novel way of extracting disjunctive landmarks. Orlando depends on different equivalence functions, and so finds landmarks with respect to different symmetries. It can be said that we are gaining 'structural analysis from symmetry'.

This is in juxtaposition to the current algorithms, which find disjunctive landmarks in one sweep; there is no indication of what *type* of symmetry the landmarks represent. This can be described as 'symmetry from structural analysis'. These algorithms can suffer from the fact that they can generate meaningless landmarks [Porteous and Cresswell, 2002]. In contrast to this, the landmarks we generate have a very specific meaning relating to the equivalence function used to reduce the problem. This should aid a user in his decision-making process. Since the algorithm uses only traditional landmarks analysis then the time complexity is better than the older algorithms.

However, the landmarks we find are only ordered with respect to each equivalence function used. The older algorithms order all of the disjunctive landmarks that they find. Orlando requires the user to know something about symmetry, if only very little at this stage. There are good arguments for both types of disjunctive landmarks analysis and a forward-propagating algorithm such as that described in Section 2.4 is seriously considered for the next version of Orlando.

We will now examine the equivalence functions supplied with Orlando. We demonstrate, by example, what type of landmark these reveal. We also discuss the individual merits and shortfalls of each.

Functional equivalences: The functional equivalence functions come in two different flavours:

1. **With respect to all goals.** Often reducing a problem by functional equivalences recovers only a negligible number of new landmarks. Reducing the problem with respect to all of the goals has the problem that the landmarks that are found will not distinguish between goals and the resources associated with those goals.

   The third landmark in Table 2 is extracted using this equivalence function; the first two, however, are not. The third landmark has to be achieved twice for a successful plan. It would be much better if it were represented as two distinct landmarks.

2. **With respect to individual goals.** Interesting landmarks can be deduced by removing each of the goals, save one, and then reducing the problem by functional identities. This method typically finds more specific, and useful, landmarks than the previous one.

   If we only consider one goal at a time, each parameter of that goal predicate is enforced as asymmetric to all objects of the same type. Thus, each object in that goal predicate will appear as an atom. Returning to Table 2, landmarks 1 and 2 are now found but 3 is not.

Relax complete types: Another way of abstracting the problem is to consider each object of the same type to be symmetric. This abstraction can counter the intuitive, but hard to define, almost-symmetries in a problem. In our logistics example Figure 1(c), for instance, relaxing the truck type would yield the landmark:

`[(in P1 T1) \/ (in P1 T2) \/ (in P1 T3)]`

This landmark is useful in the sense that it reveals the resource in the problem (i.e. the trucks). However, it does not give us any *specific* information about the problem. In this particular situation it is a useful landmark (all of the trucks are located close to each other).

It could easily be the case that there was a very distant truck in the problem that should never be considered as a candidate for delivering the parcel. This naïve relaxation would include the distant truck in the disjunction. This is clearly unattractive, but the abstraction is still useful when more specific abstractions fail to discover landmarks.

Reducing the static structure: Static predicates are those that cannot be modified. An example of a static predicate is the `road` predicate in the logistics domain. Since no actions can add or delete a `road` predicate, it is said to be *static*. Sometimes structural symmetry arises because of symmetry in the static structure of the problem.

If only the static structure of Figure 1(d) is considered then by functional identities, we can collapse the structure such that all of the trucks now occupy one location. After reintroducing the dynamic predicates, it is clear that the problem can be reduced further (as the trucks are all now at the same location in the abstraction). Note that two abstractions were performed here. One to remove the symmetry in the static structure, and then one to remove the symmetry which was then introduced.

We would now extract the following landmarks from the problem:

1. `(at T1 S) \/ (at T2 S) \/ (at T3 S) \/ (at T4 S)`
2. `(in P1 T1) \/ (in P1 T2) \/ (in P1 T3) \/ (in P1 T4)`
3. `(at T1 G) \/ (at T2 G) \/ (at T3 G) \/ (at T4 G)`

These landmarks are enough to solve this problem. Any sequence of actions that achieve these landmarks in order is a valid plan.

The previous example shows that the application of compound equivalence functions on a problem can collapse greater amounts of symmetry than one alone. We can imagine the abstractions spanning from the original problem in a hierarchy. At the top levels are those landmarks with the greatest specificity. As we go deeper into the tree, the extracted landmarks are necessarily more general.

4 Future and Related Work

The initial work on atomic landmarks in [Porteous *et al.*, 2001] has demonstrated that the performance of a planner can be improved with use of landmarks. There has remained moderate interest in the field of landmarks analysis.

The planner SGPlan[Chen *et al.*, 2004] uses landmarks (Chen et. al. refer to them as 'hidden subgoals', but their definition is the same). SGPlan decomposes the problem by finding goal-wise atomic landmarks, but not disjunctive landmarks. In the recent International Planning Competition it performed exceptionally, coming first and second place in the 'Suboptimal Metric Temporal Track' and the 'Suboptimal Propositional Track' respectively [EdelKamp and Hoffmann, 2004]. This shows that a planner that utilises landmarks can be successful. Zhu and Givan also entered a planner in the same competition, but performed less well [Zhu and Givan, 2004]. Their planner discriminates between different types of landmarks and then partially orders these to create what they call 'roadmaps'. [Sebastia *et al.*, 2003] produced a planner that uses landmarks analysis to partition a planning problem so it can be solved in a parallel architecture.

Interesting future work could include creating a planner that uses disjunctive landmarks analysis to guide its search. Analysis of the equivalence functions in different situations will give a deeper understanding of how to automate such a process. Once the landmarks are found, one possible search strategy could be searching across the disjuncts. Also, we could proceed by trying to unify the disjuncts (intersect disjunctive landmarks where facts overlap.) Perhaps it would be better to make the landmarks distinct. Only further experimentation can answer these questions.

5 Conclusions

In this paper we have introduced Orlando, a domain-analysis tool for extracting disjunctive landmarks. This API uses symmetry-breaking techniques to enable disjunctive landmarks analysis to be performed at the cost of atomic landmarks analysis. Several equivalence functions are detailed, these are the standard equivalence functions supplied with Orlando. The functions that rely on functional identities are shown to have special plan-preserving properties. They are however quite brittle. Some structural symmetries can be broken by reducing the static structure of the problem. We have discussed weaknesses of the standard equivalence functions, there is a need for the user to be able to supply custom functions.

In real problems, few objects are functionally identical. The domain engineer knows in which situations objects should be considered symmetric, but sometimes cannot abstract that out when writing the domain. In this circumstance then supplying a custom equivalence function to Orlando should be possible. These functions must depend on structural aspects of a problem known by the user, such as the map in our examples. It seems imperative that, to write reusable functions, we should have to reason about generic types. It is indeed a goal of the authors to create a generic type specification language for this purpose.

The relationship between planning and CSP is very interesting. The comparison between landmarks in planning and the backbone in CSP problems is one that should be studied further. In [Slaney and Walsh, 2001] it is found that for one planning problem (the blocks-world domain) the relationship between size of backbone and difficulty of solving that problem was slightly negatively-correlated. This was in contradiction to how size of backbone was correlated with difficulty of solving several CSP problems (strongly positive correlation). It would be interesting to find out if this difference is global for all planning/CSP problems.

Landmark-extraction has been seen as a way of making planning easier. Much work has been carried out examining the relationship between the backbone of a CSP and the phase transition of the problem [Singer *et al.*, 2000; Slaney and Walsh, 2001; Culberson and Gent, 2000]. In [Dubois and Dequen, 2001], a heuristic search algorithm for solving SAT problems, based of backbone information, is introduced. Positive results in this paper suggest that heuristics based on backbone information should be studied elsewhere.

Acknowledgments

The authors would like to thank the EPSRC. This work is part funded by EPSRC grant GR/S11015/01. We would also like to thank Alex Coddington for the use of her table format.

The authors would like to thank the reviewers for their insightful input.

References

[Chen *et al.*, 2004] Y. Chen, C. Hsu, and B.W. Wah. Sgplan: Subgoal partitioning and resolution in planning. International Planning Cometition Booklet, ICAPS 2004, June 2004.

[Clarke *et al.*, 1996] E.M. Clarke, R. Enders, T. Filkhorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1–2):77–104, 1996.

[Culberson and Gent, 2000] J Culberson and IP Gent. Frozen development in graph coloring. *Theoretical Computer Science*, 2000. In press.

[Donaldson *et al.*, 2004] A. Donaldson, A. Miller, and A. Calder. SPIN-to-GRAPE: a tool for analysing symmetry in promela models. In *Proceedings of the 6th International AMAST Workshop on real-time systems (ARTS 2004)*, Stirling, Scotland, July 2004. To appear.

[Dubois and Dequen, 2001] Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In *Proceedings of Seventeenth International Joint Conference on Artificial Intelligence*, pages 248–253, 2001.

[Edelkamp and Helmert, 2000] S. Edelkamp and M. Helmert. the implementation of mips, 2000.

[EdelKamp and Hoffmann, 2004] S. EdelKamp and J. Hoffmann. Results of the fourth international planning competition. http://ls5-www.cs.uni-dortmund.de/ edelkamp/ipc-4/, 2004.

[Emerson and Sistla, 1996] E. Allen Emerson and A. Prasad Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1–2):105–131, August 1996.

[Fox and Long, 1998] M. Fox and D. Long. The automatic inference of state invariants in TIM. *Journal of AI Research*, 9:367–421, 1998.

[Fox and Long, 1999] M. Fox and D. Long. The detection and exploitation of symmetry in planning problems. In *Proceedings of IJCAI'99*, pages 956–961, 1999.

[Fox and Long, 2002] M. Fox and D. Long. Extending the exploitation of symmetries in planning. In *Proceedings of AIPS'02*, pages 83–91, 2002.

[Fox and Long, 2003] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[Gent and Smith, 2000] Ian P. Gent and Barbara M. Smith. Symmetry Breaking in Constraint Programming. In Werner Horn, editor, *Proceedings ECAI 2000*, pages 599–603. IOS Press, 2000.

[Hoffmann, 2003] J. Hoffmann. The metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*, 20:291–341, 2003.

[Ip and Dill, 1996] C.Norris Ip and D. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9:41–75, 1996.

[Joslin and Roy, 1997] David Joslin and Amitabha Roy. Exploiting symmetry in lifted CSPs. In *AAAI/IAAI*, pages 197–202, 1997.

[Long and Fox, 2003a] D. Long and M. Fox. Plan permutation symmetries as a source of planner inefficiency. In *Proceedings of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG '03)*, 2003.

[Long and Fox, 2003b] D.P. Long and M. Fox. Symmetries in planning problems. In *Proceedings of SymCon'03: Third International Workshop on Symmetry in Constraint Satisfaction Problems*, 2003.

[Porteous and Cresswell, 2002] J. Porteous and S. Cresswell. Extending landmarks analysis to reason about resources and repetition. In *Proceedings of the 21st Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG '02)*, 2002.

[Porteous *et al.*, 2001] J. Porteous, L. Sebastia, and J. Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In *Proceedings of European Conference on Planning*, 2001.

[Sebastia *et al.*, 2003] L. Sebastia, E. Onaindia, and E. Marzal. Concurrent resolution of decomposed planning problems. In *Proceedings of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group*, 2003.

[Singer *et al.*, 2000] J. Singer, I.P. Gent, and A. Smaill. Backbone fragility and the local search cost peak. *Journal of Artificial Intelligence Research*, 12:235–270, 2000.

[Slaney and Walsh, 2001] J. Slaney and T. Walsh. Backbones in optimization and approximation. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 254–259, 2001.

[Zhu and Givan, 2003] Lin Zhu and Robert Givan. Landmark Extraction via Planning Graph Propagation. In *Printed Notes of ICAPS'03 Doctoral Consortium*, June 2003. Trento, Italy.

[Zhu and Givan, 2004] L. Zhu and R. Givan. Heuristic planning via roadmap deduction. International Planning Cometition Booklet, ICAPS 2004, June 2004.