Division of Informatics, University of Edinburgh

Centre for Intelligent Systems and their Applications

Efficient Flexible Planning via Dynamic Flexible Constraint Satisfaction.

by

Ian Miguel, Qiang Shen, Peter Jarvis

Informatics Research Report EDI-INF-RR-0131

Division of Informatics **May 2001**
http://www.informatics.ed.ac.uk/

Efficient Flexible Planning via Dynamic Flexible Constraint Satisfaction.

Ian Miguel, Qiang Shen, Peter Jarvis

Informatics Research Report EDI-INF-RR-0131

DIVISION *of* INFORMATICS
Centre for Intelligent Systems and their Applications

May 2001

Presented in: Engineering Applications of Artificial Intelligence, 14(3), pages 301-327.

Abstract :

Recent advances in AI planning have centred upon the reduction of planning to a constraint satisfaction problem (CSP) enabling the application of the efficient search algorithms available in this area. This paper continues this approach, presenting a novel technique which exploits (restriction/relaxation-based) dynamic CSP (rrDCSP) in order to further improve planner performance. Using the standard Graphplan framework, it is shown how significant efficiency gains may be obtained by viewing plan extraction as the solution of a hierarchy of such rrDCSPs. Furthermore, by using flexible constraints as a formal foundation, it is shown how the traditional boolean notion of planning can be extended to incorporate prioritised and preference-based information. Plan extraction in this context is shown to generalise the boolean rrDCSP approach, being systematically supported by the recently developed solution techniques for dynamic flexible CSPs (DFCSPs). The proposed techniques are evaluated via benchmark boolean problems and a novel flexible benchmark problem. Results obtained are very encouraging.

Keywords : Restriction/relaxation-based dynamic CSP, prioritised and preference-based planning, dynamic flexible CSPs, Graphplan

Copyright © 2002 by The University of Edinburgh. All Rights Reserved

The authors and the University of Edinburgh retain the right to reproduce and publish this paper for non-commercial purposes.

Permission is granted for this report to be reproduced by others for non-commercial purposes as long as this copyright notice is reprinted in full in any reproduction. Applications to make other use of the material should be addressed in the first instance to Copyright Permissions, Division of Informatics, The University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN, Scotland.

Efficient Flexible Planning via Dynamic Flexible Constraint Satisfaction

Ian Miguel[†], Qiang Shen[†] and Peter Jarvis[‡]

†: Division of Informatics
University of Edinburgh, 80 South Bridge
Edinburgh EH1 1HN, UK

‡: SRI International, 333 Ravenswood Ave.
Menlo Park, CA 94024, USA

email: †{ianm,qiangs}@dai.ed.ac.uk, ‡jarvis@ai.sri.com

Abstract

Recent advances in AI planning have centred upon the reduction of planning to a constraint satisfaction problem (CSP) enabling the application of the efficient search algorithms available in this area. This paper continues this approach, presenting a novel technique which exploits (restriction/relaxation-based) dynamic CSP (*rr*DCSP) in order to further improve planner performance. Using the standard Graphplan framework, it is shown how significant efficiency gains may be obtained by viewing plan extraction as the solution of a hierarchy of such *rr*DCSPs. Furthermore, by using flexible constraints as a formal foundation, it is shown how the traditional boolean notion of planning can be extended to incorporate prioritised and preference-based information. Plan extraction in this context is shown to generalise the boolean *rr*DCSP approach, being systematically supported by the recently developed solution techniques for dynamic flexible CSPs (DFCSPs). The proposed techniques are evaluated via benchmark boolean problems and a novel flexible benchmark problem. Results obtained are very encouraging.

1 Introduction

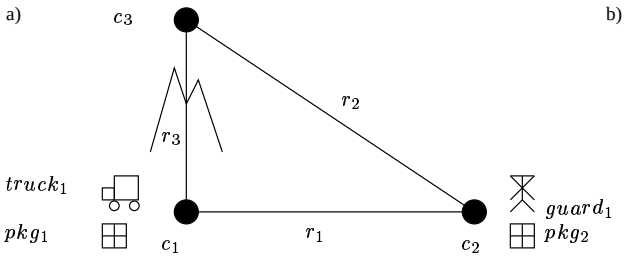
Artificial Intelligence planning [44] is an active and long-established research area in which the planner seeks to identify a sequence of actions which will achieve a set of goals given a set of specified initial conditions. It has a wide applicability to such tasks as automating data-processing procedures [6], game-playing [40], and large-scale logistics problems [46]. Solving such problems demands a highly efficient approach due to their size and complexity.

Reducing the classical AI planning problem to the solution of a constraint satisfaction problem (CSP) allows the use of the efficient solution techniques developed in that field [18]. Graphplan [4], although not directly CSP-based, has proven to be successful by creating a structure amenable to search via CSP methods in order to synthesise a valid plan. The process of plan extraction in the Graphplan framework is usually formulated as the solution of an *activity*-based dynamic constraint satisfaction problem [18, 30]. Activity constraints specify that new parts of a problem must be considered (or that existing parts can now be ignored), based on the choices made when searching for a solution. This type of constraint naturally corresponds to the relationship between a plan action and its preconditions, i.e. if an action is to be considered as part of the plan, search must be performed to support its preconditions also.

An activity-based formulation allows the problem of plan extraction to be considered in one large piece. This can be unwieldy because of its size, and inefficient because a large portion of this problem may be irrelevant to the final solution. An alternative formulation of dynamic constraint satisfaction, based on restriction (addition of constraints) and relaxation (removal of constraints) [8] is considered herein as a means of significantly improving the efficiency of the plan extraction process by tightly focusing the search and the re-use of as much search effort as possible. The application of this technique is made possible through a novel hierarchical structuring of the search process such that each level of the hierarchy constitutes a restriction/relaxation-based dynamic CSP.

Furthermore, it has become increasingly clear that classical AI planning is insufficient to capture the full subtlety of many real world problems. Hard constraints (which must all be satisfied and can only be considered to be either fully satisfied or fully violated) prove too restrictive [28]. A means of expressing priorities and preferences via flexible constraints is required. Recent developments in the field of constraint satisfaction [10, 29] extend classical CSP to support the 'soft' constraints often found in reality. In particular, dynamic flexible constraint satisfaction problems support such constraints in the context of a dynamically changing problem. Through the generalisation of the hierarchical search technique introduced for boolean problems, this paper, consolidating the success of initial attempts [27], also shows how to exploit dynamic flexible CSP techniques to extend the basic Graphplan framework to reason about flexible planning problems.

The structure of the rest of this paper is as follows. The next section outlines the basic Graphplan framework for synthesising plans from standard domains (e.g. STRIPS [12]). Section 3 provides an explanation of constraint satisfaction in general and the differences between two types



b) (**operator** LOAD-TRUCK
(**params** (t truck) (p package) (l location)
(g guard))
(**preconds** (at t l) (at p l) (on g t))
(**effects** (on p t) (not(at p l))))

Figure 1: An Example Problem and one of its Operators

of dynamic constraint satisfaction. It then describes a novel approach to plan extraction from a Graphplan-produced planning graph using a hierarchy of restriction/relaxation-based dynamic CSPs. It is shown how significant performance gains can be obtained using this method. Section 4 describes a new type of flexible planning problem which is able to express tradeoffs and soft constraints between actions. Section 5 then describes the flexible Graphplan (FGP) solution algorithm which can synthesise flexible plans. Experimental results are presented in section 6, demonstrating the success of the present research. Section 7 relates the approaches presented in this paper to work reported elsewhere. Section 8 concludes the paper and points out important future work. Finally, appendix A contains the details of an example flexible problem domain.

2 An Overview of Graphplan

The Graphplan [4] framework is a natural choice of synthesis technique upon which to base both the improved boolean and flexible planning systems. It possesses some important properties, including: optimality in the number of actions and length of a synthesised plan; soundness and completeness; and polynomial time and space complexity of the creation of the planning graph structure (see section 2.1) that is central to the CSP methods employed in this paper. In addition, Graphplan is a planning-specific approach, with the planning graph providing some structure that may lead to more efficient solution methods than those based solely on a more basic representation. See [2] for recent work in this regard.

Graphplan continues the long tradition in planning research of concentrating on a subset of issues known as the STRIPS domain [12]. Maintaining this tradition allows the development and evaluation of a flexible capability in a conceptually clean framework. This domain type consists of plan goals, represented as a propositional conjunction, and a set of *actions* available to the planner. Each action is described by a conjunctive *precondition* and a conjunctive *effect*, defining a transition function from one universe of discourse to another. The action can be executed in any universe of discourse satisfying the preconditions.

An example problem from the STRIPS domain, shown in figure 1a, is used throughout this paper first to illustrate the operation of Graphplan and then (in an extended form) the operation of FGP. In the figure, the c_i represent three cities, r_1 and r_2 represent major roads and r_3 an

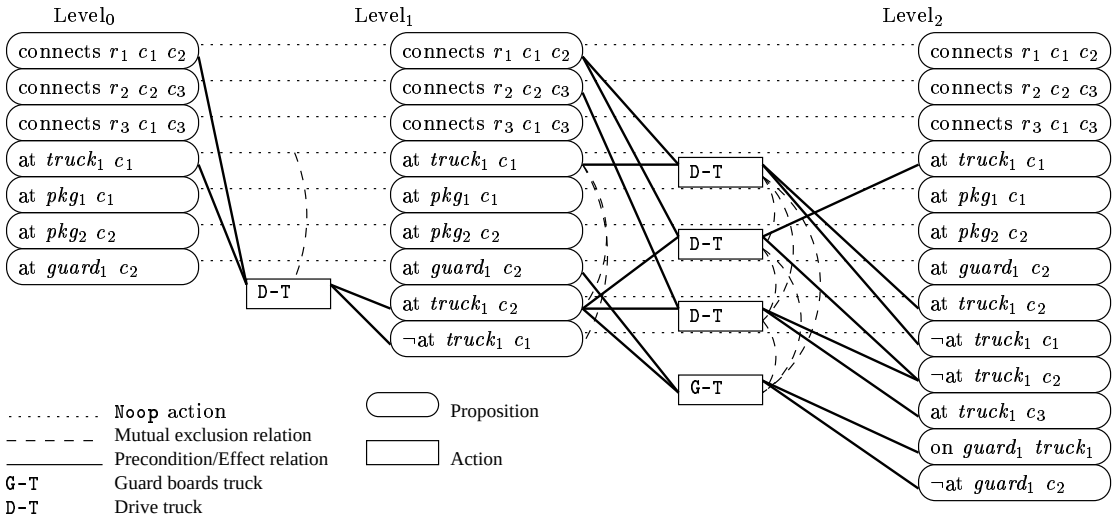


Figure 2: Planning Graph $Levels_{0-2}$ for the Problem Given in Figure 1. For clarity, only mutual exclusion relations between non-Noop actions are shown in $actions_2$ and mutual exclusion relations are not shown in $propositions_2$.

unsafe track through the hills. The single goal of this problem is to transport package pkg_1 to city c_3. Figure 1*b* shows one action from the example problem concerning the loading of a package onto a truck. This operator requires that the truck and package are at the same location and also makes the implicit assumption that the package is valuable - the presence of a guard on the truck is required. Further operators are defined for this problem to allow unloading of packages, driving the truck between cities (using main roads only) and to allow the guard to board the truck.

2.1 The Planning Graph

Graphplan synthesises plans through constructing and analysing a structure called a Planning Graph in two interleaved phases: a forward phase where the planning graph is extended until the plan goals are found, and a backward phase where the planning graph is searched to extract a valid plan. A planning graph consists of two types of nodes, propositions and actions, divided into *levels*. $Level_i$ contains a set of actions ($actions_i$) and a set of propositions ($propositions_i$). $Level_0$ is special in that it contains only proposition nodes which capture the initial state of the problem.

Extending the planning graph to $level_i$ from $level_{i-1}$ consists of introducing every action whose preconditions exist (and are mutually consistent) in $propositions_{i-1}$ to form $actions_i$. The effect propositions of these actions are placed in $propositions_i$. A special action, Noop, records the persistence of a proposition from level to level. Figure 2 shows levels 0 to 2 of the planning graph constructed for the example problem.

A second component of a planning graph is the use of mutual exclusion constraints. These binary constraints record the fact that the two specified actions (or propositions) cannot occur

together at one level of a valid plan. A pair of nodes in $actions_i$ are inconsistent if:

Inconsistent Effects An effect of one action in $propositions_i$ is the negation of an effect of the other action in $propositions_i$. For example, driving the truck from c_1 to c_2 asserts that the truck is at c_2, while driving the truck from c_2 to c_3 asserts that it is *not* at c_2, but at c_3.

Interference An effect of one action in $propositions_i$ is the negation of a necessary precondition of the other action in $propositions_{i-1}$. For example, driving the truck from c_2 to c_1 asserts that the truck is not at c_2, a necessary precondition for driving the truck from c_2 to c_3.

Competing Needs The actions have mutually exclusive preconditions in $propositions_{i-1}$. For example, driving the truck from c_1 to c_2 requires that the truck is currently at c_1, whereas the guard boarding the truck at c_2 requires that the truck is at c_2.

A pair of nodes in $propositions_i$ are inconsistent if:

Negation One proposition is the negation of the other.

Inconsistent Support All nodes in $actions_i$ which produce one proposition are exclusive of all actions that produce the other. For example, all methods of asserting that the truck is at c_1 are exclusive of all methods of asserting that it is at c_2.

2.2 Basic Plan Extraction

Extracting a plan from a planning graph consists of identifying a subgraph connecting the goal propositions at some $level_g$ back to the initial conditions. The subgoals and action nodes at each level of this subgraph must be mutually consistent. A basic method of finding a valid plan is as follows:

- Expand the graph until all goal propositions exist in the current level ($level_g$) and are mutually consistent.
- For each set of consistent nodes in $actions_g$ which *support* (i.e. have as effects) the goal propositions, the preconditions of this set of actions in $propositions_{g-1}$ become subgoals which must also be supported, and so on. If, for example, 'at $truck_1$ c_2' (meaning that $truck_1$ is at c_2) becomes a sub-goal at $level_2$, it can be supported by either a `Noop` or `Drive-truck` action.
- For each set of subgoals at $level_i$:
  - If no consistent set of nodes in $actions_i$ can be found to support the current set of subgoals in $propositions_i$, backtrack to $level_{i+1}$ to find an alternative set of consistent nodes in $actions_{i+1}$.
  - If the search reaches $level_0$ and the subgoals match the initial conditions, a valid plan has been found.

- If no valid plan can be found, extend the graph to $level_{g+1}$ and repeat.

Using this method, an eight-step plan can be synthesised in this way for the example problem. It involves driving to c_2 to fetch the guard and load pkg_2 before returning to load pkg_1. The packages are then transported via the main roads to c_3 and unloaded.

2.3 Memoisation

The basic scheme described above has no ability to prune the search tree so that search paths which will necessarily lead to failure are abandoned as soon as possible. A mutually-consistent set of nodes in $propositions_i$ are *conjunctively unsupportable* if all combinations of nodes in $actions_i$ that assert these propositions are either directly inconsistent or their own preconditions are unsupportable. In the example problem, the goal propositions 'at pkg_1 c_3' and 'at pkg_2 c_3' first occur together at $level_7$. Although they are not directly inconsistent, it may quickly be established that these two propositions are unsupportable at this level: at $level_5$ no mutually-consistent combinations of supporting actions exist for the propositions 'at $truck_1$ c_2', 'on pkg_1 $truck_1$', and 'on pkg_2 $truck_1$', which are necessary to achieve the goals in seven steps.

Memoisation is the process of recording such unsupportable sets, known as *memosets* [27]. If a memoset (or a superset thereof) is encountered again at the level at which it was recorded, the current search branch can be pruned immediately. Memoisation in its simplest form records the entire set of nodes currently considered at $propositions_i$ once the fact that they are unsupportable has been established. However, it is likely in general that some of the propositions are irrelevant to the failure. More effective methods of memoisation can significantly improve the efficiency of plan extraction, as is discussed in the next section.

3 Plan Extraction via Dynamic Constraint Satisfaction

This section describes an efficient method of plan extraction in the Graphplan framework founded on restriction/relaxation-based dynamic CSP techniques called GP-rrDCSP (a definition of rrDCSP follows). It is shown how repair-based solution techniques [28, 42] for such dynamic CSPs can reduce overall search effort and facilitate the exploitation of memoset information.

3.1 Concepts of Dynamic Constraint Satisfaction

A classical CSP involves a set of n variables, $X = \{x_1, ..., x_n\}$, and a set of constraints, C, over these variables. Each $x_i \in X$ has an associated domain, D_i, describing its potential values. A variable *assignment* is an assignment to a variable, x_i, of one of the values from D_i. A *partial assignment* to X is an assignment to one or more $x_i \in X$, and a *complete assignment* to X is an assignment to every $x_i \in X$. A classical (hard) constraint $c(x_i, ..., x_j) \in C$ specifies a subset of the Cartesian product $D_i \times ... \times D_j$, indicating variable assignments that are compatible with each

other. These constraints are imperative (each one *must* be satisfied) and inflexible (they are fully satisfied or fully violated). Further, the CSP is assumed to be static, precluding any changes to X, D, or C after their initial specification.

In the context of the Graphplan planning graph, a subgoal in $propositons_i$ is regarded as a CSP variable, with a domain consisting of the set of nodes in $actions_i$ which support this subgoal. For example, if 'at $truck_1$ c_2' in $propositions_2$ (see figure 2) is regarded as a CSP variable, its domain comprises the `Noop` and `Drive-truck` actions that specify it as an effect. Binary constraints come directly from the mutual exclusion relations between the action nodes in the domains of the variables. If 'on $guard_1$ $truck_1$' is a second variable, with the `Guard-boards-truck` action being its single domain element, a constraint is constructed over the two variables allowing the `Noop` but not `Drive-truck` to be combined with `Guard-boards-truck`, since a mutual exclusion relation disallows the latter combination.

It should be noted that there are different interpretations of the term 'Dynamic Constraint Satisfaction Problem'. One version consists of a classical CSP together with *activity* constraints describing how variables are added (or removed) from a problem based on the assignments to existing variables. This type of problem, which will be referred to herein as aDCSP, corresponds naturally to the structure of the planning graph. Activity constraints describe the relationship between an action node and its preconditions, specifying that a further variable/subgoal must be considered when this action is chosen [30]

The other type involves a sequence of standard CSPs linked by restrictions (constraints are added) and relaxations (constraints are removed) [8]. This model of DCSP will be referred to as rrDCSP. Unlike the problems that may be described via aDCSP, it is not usually known beforehand how a problem will change. It is this ability to cope with an unpredictably changing problem which makes rrDCSP useful in the context of Graphplan, as will be shown. Solution techniques for rrDCSPs [28] usually work by using the solution to the previous problem in the sequence as a basis, then attempting to repair it to find a solution to the current problem, maintaining as much of the effort that went into solving the original problem as possible.

3.2 A Hierarchical Approach

Previously, rrDCSP has not been applied to the problem of plan extraction due to its lack of ability to cope with activity constraints. However, by considering not just a single rrDCSP but a hierarchy of such problems, each corresponding to a level in the planning graph, activity constraints can be decoupled from the solution procedure within a single level. Consider the problem formed at $level_g$, where the plan goals exist. Depending on the structure of the mutual exclusion constraints between the action nodes which support the plan goals, there may be several solutions (in terms of mutually consistent sets of supporting actions) to this initial problem. Each such solution, through the preconditions of the associated actions, specifies a different sub-problem at $level_{g-1}$. Instead of viewing this new set of sub-problems individually, they may instead be considered as

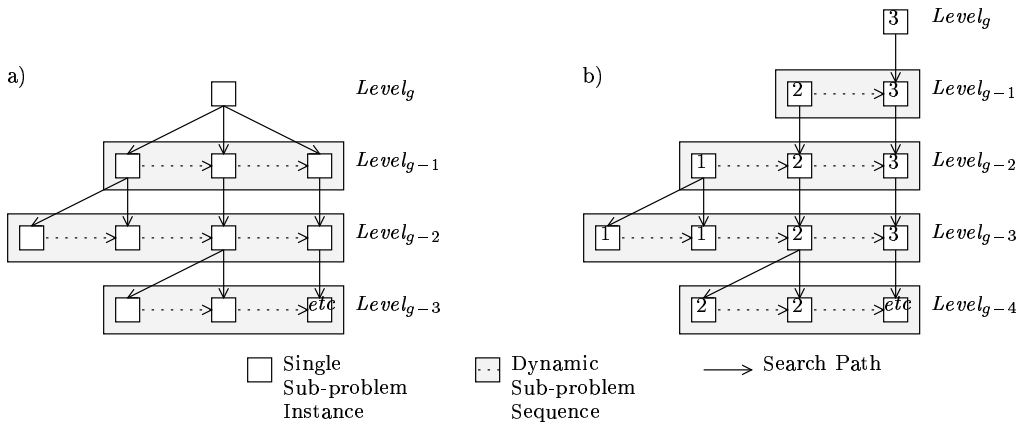


Figure 3: Plan Extraction as a Hierarchy of *rr*DCSPs. Note that the planning graph has been rotated anti-clockwise by 90 degrees.

an *rr*DCSP, as shown in figure 3*a*.

Each sub-problem in a dynamic sequence can then be solved without explicit reference to activity constraints. The present approach makes use of the Local Changes *rr*DCSP algorithm [42] to solve these sequences. This is a repair-based algorithm which attempts to minimise the effort required in solving the current problem in a dynamic sequence by maintaining as much of the solution to the previous problem as possible.

This method of plan extraction has two immediate advantages:

1. Assuming that the solutions to the problem at $level_g$ have some intersection (a likely occurrence in all but the most tightly constrained cases), there will be similarities between the sub-problems at $level_{g-1}$. An *rr*DCSP algorithm can exploit these similarities by re-using effort applied in solving the previous problem in the sequence when solving the current sub-problem.

2. Effort used in levels visited in a failed attempt at plan extraction can be re-used when the graph is expanded and a further attempt at plan extraction is made. This process is shown in figure 3*b*: each sub-problem is labelled with the number of the extraction phase that explored it.

A further important advantage of this method is realised through the propagation of memosets, as shown below.

3.3 Memoisation in the Hierarchical Context

In order to increase efficiency in searching for a valid plan, it is necessary to prune branches of the search space which will necessarily lead to failure. The simple memoisation strategy described in section 2.3 is of limited usefulness because the size of the recorded memoset is likely to be

large. Such large sets are unlikely to be encountered many times in the search process and hence will not contribute greatly to the pruning of the search tree. Since pruning the search tree can significantly reduce the number of sub-problems that must be solved, it is worthwhile adopting a more intelligent strategy to discover much smaller memosets which are likely to be encountered many times.

3.3.1 Local Memoset Discovery via Sub-problem Pre-processing

The memoset discovery strategy adopted herein is based on the CSP technique of enforcing *arc consistency* [28]. Local arc consistency holds for a constraint $c(x_i, x_j)$ with respect to x_i if, for each $d_i \in D_i$ there exists at least one element, $d_j \in D_j$ such that $c(x_i, x_j)$ is satisfied by the assignments $x_i = d_i$, $x_j = d_j$. All elements that do not fulfil this criterion may be removed from D_i. This local update is the basis for further filtering, hence the result of any change is propagated through the problem.

If the mutual exclusion constraints are such that a supporting action in the domain of a subgoal is incompatible with all elements in the domain of another subgoal, that element may be filtered from consideration since it cannot take part in any valid solution to the sub-problem. The corresponding action node is *not* removed from the planning graph since its removal can only be inferred in the context of this sub-problem. The process continues until no further elements may be filtered. The problem is then said to be arc consistent. However, if at any stage all elements in a domain are removed the problem is inconsistent (i.e. insoluble). Efficient algorithms exist to enforce arc consistency [3, 26, 31].

Figure 4 shows a sub-problem at $level_5$ taken from the example problem. As noted in section 2.3, this sub-problem is encountered when trying to establish support for the plan goals ('at pkg_1 c_3' and 'at pkg_2 c_3') at $level_7$. First, the `Noop` in the domain of 'on pkg_1 $truck_1$' can be filtered, since it is inconsistent with all elements of the domain of 'on pkg_2 $truck_1$'. The `Load-truck` action in the same domain can also be filtered, since it is inconsistent with all elements in the domain of 'at $truck_1$ c_2'. Hence, the sub-problem is inconsistent and its constituent propositions are conjunctively unsupportable.

Having established arc consistency, assuming it is not inconsistent a sub-problem is easier to solve due to its reduced size. However, this is not the chief goal of enforcing arc consistency in this context. By keeping track of the reason for the removal of each element, explanations for failure to find a solution can be constructed. The *eliminating explanation* of a domain element d_i, a term borrowed from [13], is used herein to specify the variable, x_e, responsible for the removal of d_i, i.e. that there are no elements in D_e which are compatible with d_i. In figure 4, for example, 'on pkg_2 $truck_1$' is an eliminating explanation for `Noop` in the domain of 'on pkg_1 $truck_1$'.

In the case of an inconsistent problem, the eliminating explanations for each element of the totally filtered domain can be traced to provide an overall explanation for the inconsistency of the problem in terms of a conjunctively unsupportable subset of propositions. The basic tracing pro-

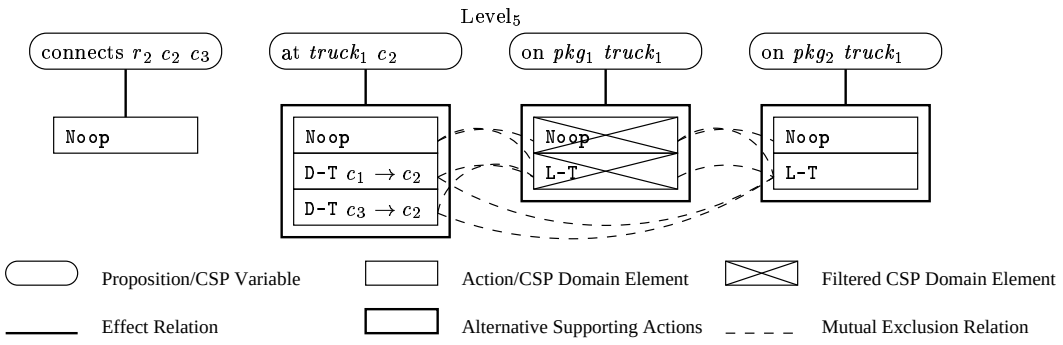


Figure 4: Enforcing Arc Consistency on a Sub-problem

cedure, using eliminating explanations gained while enforcing arc consistency, is shown in figure 5. The domains of all eliminating explanations must also be examined since they may contain elements filtered by variables external to the current memoset. If these external variables are not also added to the memoset, the filtered domain elements would consequently be reinstated, potentially allowing combinations of assignments to the memoset variables that are in fact supportable.

The resulting subset of propositions can then be recorded as a memoset. The benefit of this process comes from the size of the memosets recorded. In the example problem, 'connects r_1 c_1 c_2' has no bearing on the inconsistency of the problem and is therefore excluded from the memoset. In this example the saving is small, but in more complex examples with larger sub-problems the saving is generally greater, leading to a significant pruning of the search tree.

```
01. Procedure ConstructMemoSet(D_i)
02.   Trace ← {D_i}
03.   MemoSet ← {x_i}
04.   While Trace ≠ φ
05.     Select and Remove D_t ∈ Trace
06.     For Each d_t−f ∈ D_t where d_t−f has been filtered
07.       x_e is the eliminating explanation for d_t−f
08.       If x_e ∉ MemoSet And D_e ∉ Trace
09.         Add x_e To MemoSet
10.         Add D_e To Trace
```

Figure 5: Basic Memoset Construction Following Total Filtering of D_i

Figure 4 shows only one of several ways in which to enforce arc consistency on the given sub-problem. It is equally possible, for example, to start with 'on pkg_2 $truck_1$', filtering the `Load-truck` action since it is incompatible with all elements of the domain of 'on pkg_1 $truck_1$. In this simple example the starting point makes no difference to the memoset identified, but this is not uniformly the case. A poor variable/constraint selection order can lead to sub-optimal memosets being constructed.

Section 3.4 outlines a heuristic method of identifying small memosets, but there is a simple method of improving the memoset construction procedure itself to remove variables that are

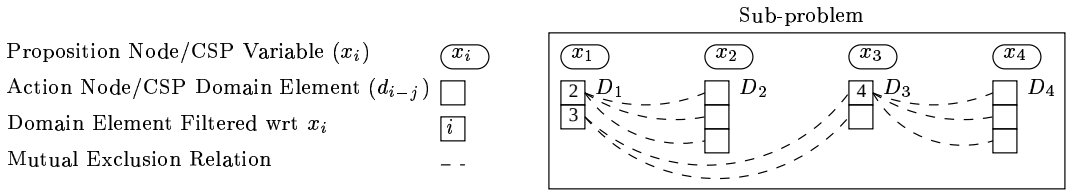


Figure 6: Avoiding Irrelevant Variables during Memoset Construction

not part of the inconsistency. Consider figure 6, which shows an inconsistent sub-problem with all of the elements of D_1 filtered. Tracing the eliminating explanations for this initial domain immediately derives the partial memoset $\{x_1, x_2, x_3\}$. Ordinarily, x_4 must also be added to the memoset. In this case, however, it is not necessary since a mutual exclusion constraint exists between d_{1-2} and d_{3-1}. Hence, even if d_{3-1} had not been eliminated by x_4, x_3 would eliminate d_{1-2}. This *irrelevance test* may be performed on all filtered domain elements to test whether it is actually necessary to add their eliminating explanations to the current partial memoset. The test is whether or not every domain element (in this case d_{1-2}) whose eliminating explanation causes the current domain (D_3) to be considered is incompatible with the filtered domain element currently under consideration (d_{3-1}).

Irrelevance testing works to limit the effects of a poor order of constraint selection when enforcing arc consistency. In the example shown in figure 6, if the constraints involving x_1, x_2 and x_3 are processed first, all domain elements are still filtered from D_1, leading to the memoset $\{x_1, x_2, x_3\}$ without ever considering x_4.

3.3.2 Local Memoset Discovery via Search

Enforcing arc consistency is not guaranteed to detect an inconsistent problem [28]. It is possible that a more complex inconsistency will be uncovered only during subsequent search. For this reason, there must also be a mechanism for constructing a memoset while solving a sub-problem. When a dead end is discovered in the search, all elements of the current variable's domain are incompatible with the current partial assignment, forming a *conflict set* with it. This inconsistency can be recorded as a partial memoset for this sub-problem.

For the purpose of memoset discovery, only the variables involved in the conflict are recorded. *Shallow Learning* [7] is then used to remove irrelevant elements from this conflict set, where an irrelevant variable has an assignment which is compatible with all possible assignments to the current variable. If the sub-problem is inconsistent, the memoset is constructed from the union of all discovered conflict sets. The number and size of these conflict sets may be limited heuristically, as described in section 3.4 below.

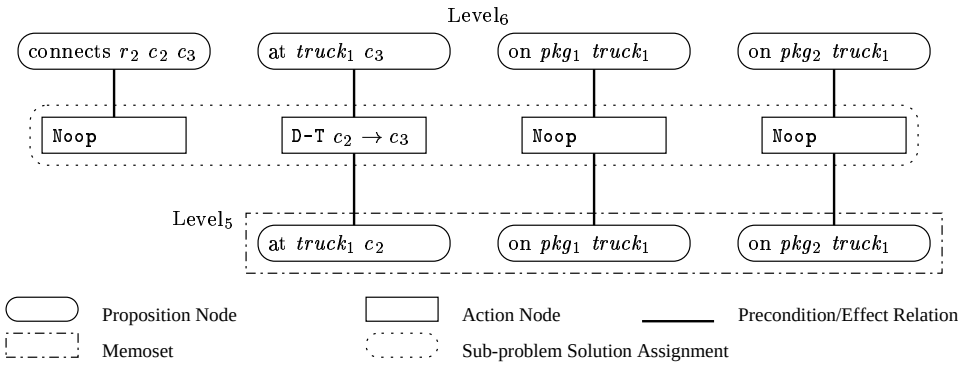


Figure 7: Reverse Mapping from a Memoset to a Sub-assignment via Activity Constraints

3.3.3 Propagated Repair

Memoset information gathered at child sub-problems is useful in guiding further solution extraction at the parent sub-problem. The memoset gained from the current sub-problem at $level_{i-1}$ can be mapped back onto the action nodes in the solution assignment at $level_i$ which generated sub-problem$_{i-1}$ by following the activity constraints in reverse. Figure 7 shows the propagation of the memoset constructed from the sub-problem given in figure 4. The partial assignment thus specified can then be repaired via the Local Changes algorithm to find an alternative complete assignment to the sub-problem at $level_6$ which does not include the combination of assignments (i.e. 'at $truck_1$ c_3' $\leftarrow$ `Drive-truck`, 'on pkg_1 $truck_1$' $\leftarrow$ `Noop`, 'on pkg_2 $truck_1$' $\leftarrow$ `Noop`) which led to the memoset.

Essentially, propagating memosets up the hierarchy from $level_{i-1}$ to $level_i$ specifies extra constraints which are added to the sub-problem at $level_i$ to restrict the set of solutions. This process is in contrast to the simple enumeration of all solutions to a sub-problem, of which there may be many. Propagating memosets in conjunction with *rr*DCSP repair techniques narrows down exhaustive search considerably.

3.3.4 Propagated Memoset Discovery

The propagation of memoset information up the level hierarchy is also essential for the discovery of memosets at sub-problems which have valid solutions, but each leads to a dead end further down the hierarchy (such as the sub-problem at $level_6$ shown in figure 7). In this case, a memoset may be constructed by considering, for each solution found to a sub-problem at $level_i$, the memoset generated or matched at the associated sub-problem at $level_{i-1}$.

Each memoset thus found is used to generate a partial memoset at $level_i$. The first step in this process is the same as for propagated repair: the child memoset must be mapped onto the nodes in $actions_i$ which are responsible for its generation. Second, the nodes in $propositions_i$ whose assignments collectively correspond to the culprit nodes identified in $actions_i$ must be identified.

In the example given in figure 7, there is a simple one to one mapping, yielding the partial memoset {'at $truck_1$ c_3', 'on pkg_1 $truck_1$', 'on pkg_1 $truck_1$'}. In general, however, each action can map to multiple proposition nodes, corresponding to multiple effects. In the case where a memoset maps back onto more than one sub-assignment, it is clearly best to choose the one which maps back onto the set of variables in sub-problem$_i$ which has the greatest intersection with the current partial memoset. For each solution to a sub-problem at $level_i$ explored, as dictated by propagated repair, the memoset from the child sub-problem is used to enlarge (if necessary) the current partial memoset.

The variables of a sub-problem at $level_{i-1}$ might be a superset of more than one recorded memoset at $level_{i-1}$. This situation arises when there are multiple reasons for the lack of support for the associated propositions. Since all such memosets represent equally valid reasons for failure, it is possible to choose which one to propagate to $level_i$. This choice is made on the basis of increasing the size of the current partial memoset for the sub-problem at $level_i$ as little as possible.

3.3.5 Removing Irrelevant Variables from Memosets Created by Propagation

When constructing a memoset at $level_i$ through propagation, the *core memoset* is defined as those elements of $propositions_i$ that are directly specified by the propagation of memosets from $level_{i-1}$. It is also necessary to consider eliminating explanations for elements of this core set. This is because, as per the local case described in section 3.3.1, filtered domain elements may allow different combinations of assignments to the core memoset variables that are supportable. An irrelevance test in this case is not as simple as when considering local constraints only. It is, however, worthwhile as it minimises the size of the memosets hence pruning overall search.

The solution adopted herein is based on the creation of a *micro-problem* with variables consisting of the propositions in the core memoset. Through solving this initial micro-problem and likewise the micro-problems that arise at lower levels based on each solution found, support is sought for the core memoset. If a consistent sub-graph exists connecting the core memoset to the initial conditions then, by definition, its constituent variables are supportable at this level. Therefore, the eliminating explanations for the core memoset variables must be relevant, and a new memoset containing both the core memoset and its associated eliminating explanations is recorded. If, on the other hand, no such sub-graph exists, then the core memoset is established as unsupportable and recorded as a new memoset in its own right.

Micro-problems constructed for propagated memoset irrelevance testing are fitted seamlessly into the dynamic sequence of problems, as shown in figure 8. This is possible because in both 'normal' and 'relevance' modes, GP-*rr*DCSP is performing the same task: the identification of a consistent sub-graph (if one exists) connecting a subset of propositions at some level to the initial conditions. A useful side-effect of propagated relevance testing is that memosets may also be discovered and recorded during this process. Since the micro-problems are typically small, memosets generated in this way are likely to be small also. Furthermore, the small size of the

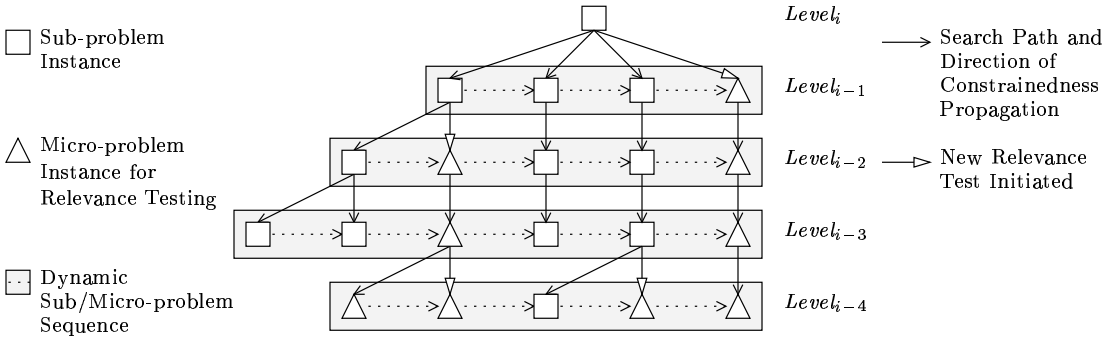


Figure 8: Propagated Irrelevance Testing and Propagation of Constrainedness Information

micro-problems means that they are easier to solve on an individual basis.

Propagated irrelevance testing is used recursively (as also shown in figure 8), such that if a core memoset discovered at a micro-problem also contains variables whose domains have elements filtered by variables external to it, a further irrelevance test can be initiated. This technique leads to a highly focused exploration of the search space, extracting the smallest memosets possible.

3.4 Focusing the Search

Exploring the path most likely to fail (the *fail-first* principle [16]) is a well established means of reducing the effort required in solving a CSP. Detecting dead ends in the search early leads to an effective pruning of the search space. In the hierarchical context, it is helpful to be able to examine the 'right' variables/constraints first to extract the smallest memosets. After all, a single sub-problem may contain many conjunctively unsupportable sets of variables, but as noted in section 3.3, the smallest ones will lead to the most effective pruning of the search space.

A method following the fail-first principle is to select variables with the smallest domains first. These variables have a higher likelihood of being part of an inconsistency exactly because they have fewer potential assignments. In the hierarchical context, this principle may be effectively used by propagating such information down the hierarchy, as shown in figure 8. Hence, a *rank* may be calculated for a variable based on the cumulative size of its parents (and grandparents, etc) domains. This rank may then be used in selecting which constraint to process during arc consistency enforcement (using the sum of the ranks of the two variables in each constraint) and which variable to select next during search. This technique has the effect of reducing the sizes of discovered memosets. Ties may be broken by biasing the ranks towards variables whose ancestors are already in partial memosets. Hence, when the memoset is mapped back, this is likely to lead to as small an addition to the partial memoset as possible.

4 Flexible Planning Problems

Many real-world planning problems present the need for soft constraints [28]. Reconsider the example problem given in figure 1. The preconditions of the `Load-truck` action state that (i) the truck and the (implicitly) valuable package must be co-located, and (ii) that a guard must be present. While precondition (i) is imperative, precondition (ii) is a preference or soft constraint and can be relaxed with an associated damage to the resultant plan. The *flexible planning problem*, described below, adds the ability to reason about the relaxation of soft constraints and the resultant damage to the plan synthesised to the Graphplan framework. Section 5 then describes Flexible Graphplan (FGP), a solution technique capable of synthesising flexible plans given a flexible planning problem.

4.1 Representation

A flexible planning problem, Ψ, consists of a 4-tuple, $< \Phi, \mathrm{O}, \mathrm{I}, \Gamma >$, denoting sets of plan objects, flexible operators, initial conditions consisting of flexible propositions, and flexible goal conditions, respectively. Boolean propositions are herein replaced by flexible propositions, ρ, of the form $(\rho\ \phi_1, \phi_2, ..., \phi_j\ k_i)$, where $\phi_i \in \Phi$ and k_i is an element of a totally ordered set, K, which denotes the subjective degree of truth of the proposition. K is composed of a finite number of membership degrees, $k_\perp, k_1, ..., k_\top$. The original boolean proposition type is captured at the end points of K, with $k_\perp \in K$ and $k_\top \in K$ indicating total falsehood and total truth respectively. For brevity, when dealing with propositions which only ever take a truth value of $k_\perp$ or $k_\top$, the boolean style of $\neg(\rho\ \phi_1, \phi_2, ..., \phi_j)$ and $(\rho\ \phi_1, \phi_2, ..., \phi_j)$ is adopted respectively.

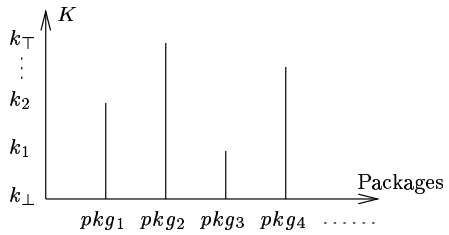


Figure 9: Discrete Unary Fuzzy Relation Underlying the Flexible Proposition $(valuable\ ?pkg_i\ k_j)$

A flexible proposition can easily be described by a *fuzzy relation* [35], R, which is defined by a membership function $\mu_R(.) : \Phi_1 \times \Phi_2 \times ... \times \Phi_j \rightarrow K$, where $\Phi_1 \times \Phi_2 \times ... \times \Phi_j$ is the Cartesian product of the subsets of Φ allowable at this place in the proposition. For instance, flexible proposition $(valuable\ ?pkg_i\ k_j)$ shown in figure 9, where $?pkg_i$ is instantiated to a package in a logistics domain, is represented by a discrete unary fuzzy relation which maps each package to an associated truth degree in K.

A flexible operator, $o \in \mathrm{O}$, must recognise *how well* its preconditions are satisfied. Flexible operators are described by fuzzy relations which map from the precondition space onto a totally

Precondition Operator, τ	Arguments, κ
$\equiv$	k_i
$\leq$	k_i
$\geq$	k_i
range	k_i k_j
discrete	k_a k_b ...

Table 1: Operators Used to Define Flexible Preconditions and Goals

ordered *satisfaction scale*, L and a set of flexible effects propositions. L is distinct from K in that it represents the degree of compromise required to apply an operator in the current situation as opposed to the truth of a particular proposition.

As per K, L is composed of a finite number of membership degrees, $l_\perp, l_1, ..., l_\top$. The endpoints, $l_\perp \in L$ and $l_\top \in L$ respectively denote a complete lack of satisfaction (in which case the operator is not added to the planning graph based on these preconditions) and complete satisfaction. The `Noop` action is a special case which has a satisfaction of $l_\top$.

a) (**operator** *o*
(**params** $param_1$, $param_2$, ...)
σ_i: {**when** (**preconds** θ_{i_1} θ_{i_2} ...)
(**effects** ρ_{i_1} ρ_{i_2} ...)
(**satisfaction** l_i)}
σ_j: {**when** (**preconds** θ_{j_1} θ_{j_2} ...)
(**effects** ρ_{j_1} ρ_{j_2} ...)
(**satisfaction** l_j)} *etc*)

b) (**goal** γ
{**when** θ_i (**satisfaction** l_i)}
{**when** θ_j (**satisfaction** l_j)} *etc*)

Figure 10: General Formats of Flexible Operators and Goals

A flexible operator (see figure 10*a*) consists of a set of disjoint conditional clauses, Σ (similar to conditional effects [34]). Each $\sigma \in \Sigma$ is a triple $< \Theta, \mathrm{E}, k_i >$ respectively denoting a conjunction of flexible preconditions, a conjunction of flexible effect propositions and the satisfaction degree of this operator given these preconditions. Each $\theta \in \Theta$ has the form $(\rho\ \phi_1, \phi_2, ..., \phi_j\ \tau\ \kappa\ )$, where τ is a precondition operator with argument set κ, as defined in table 1. Each σ_i maps a subset of the space of preconditions to a particular set of effects and a satisfaction degree in L. Again, when dealing with preconditions which only ever take a truth value of $k_\perp$ or $k_\top$, the boolean style of $\neg(\rho\ \phi_1, \phi_2, ..., \phi_j)$ and $(\rho\ \phi_1, \phi_2, ..., \phi_j)$ is adopted for θ respectively.

A flexible plan goal $\gamma \in \Gamma$ maps from the space of flexible propositions to L. Each goal is defined using a number of clauses, as shown in figure 10*b*. The preconditions are defined exactly as those used in the flexible operators. More than one set of mutually-consistent propositions may exist which satisfy the plan goals to some extent. Hence, the satisfaction degree of the plan as a whole must take into account the goal satisfaction degrees as well as those of the flexible operators.

The satisfaction degree of a flexible plan can be determined by the conjunctive combination of the satisfaction degrees of each operator and each goal used in the plan. The conjunctive combi-

nation of two fuzzy relations, $R_i \otimes R_j$, is usually interpreted as the minimum membership value assigned by either relation [35]. The *min* operator has the desirable property of idempotency ($\forall u, u \oplus u = u$) which enables classical CSP k-consistency enforcing techniques to be straightforwardly extended to support flexible CSP [10] and is central to the computational efficiency of the FGP solver. The disadvantage of using an idempotent operator is the so-called *drowning effect* [38], where a low satisfaction degree resulting from one assignment 'drowns' several others whose satisfaction degrees, whether optimal or sub-optimal, are not reflected in the overall satisfaction degree. This problem can, however, be counteracted to an extent by exploring the path of highest satisfaction first, as demonstrated in section 6.

The *quality* of a flexible plan is its satisfaction degree combined with its length, where the shorter of two plans with equivalent satisfaction degrees is better. Hence, plans with a satisfaction degree below $l_\top$ should not be viewed simply as degraded versions of a 'perfect' plan, but as compromises which trade the satisfaction of flexible operators and goals against a (potentially significant) reduction in plan length. Indeed, by introducing flexibility it may be possible to find a compromise plan where no plan at all existed in the boolean case because the problem was over-constrained.

4.2 An Example (Continued)

A flexible version of the problem given in figure 1 is now presented to illustrate the operation of FGP. The basic structure of the problem remains as shown in the figure in terms of the connections between the three cities and the initial placement of plan objects, but flexibility is added to the plan operators and goals to allow FGP to consider compromise solutions. For simplicity, the following definitions are used for K and L: $K = \{k_\bot, k_1, k_2, k_\top\}$, $L = \{l_\bot, l_1, l_2, l_\top\}$. In general, K and L do not have to have the same cardinality.

a)

(**operator** `Load-truck`
 (**params** (?*t* `truck`) (?*p* `package`) (?*l* `location`)
  (?*g* `guard`))

 {**when** (**preconds** (*at* ?*t* ?*l*) (*at* ?*p* ?*l*)
  (*valuable* ?*p* `<=` k_1))
  (**effects** (*on* ?*p* ?*t*) (*not* (*at* ?*p* ?*l*)))
  (**satisfaction** $l_\top$)}

 {**when** (**preconds** (*at* ?*t* ?*l*) (*at* ?*p* ?*l*) (*on* ?*g* ?*t*)
  (*valuable* ?*p* `>=` k_2))
  (**effects** (*on* ?*p* ?*t*) (*not* (*at* ?*p* ?*l*)))
  (**satisfaction** $l_\top$)}

 {**when** (**preconds** (*at* ?*t* ?*l*) (*at* ?*p* ?*l*)
  (*not* (*on* ?*g* ?*t*)) (*valuable* ?*p* `>=` k_2))
  (**effects** (*on* ?*p* ?*t*) (*not* (*at* ?*p* ?*l*)))
  (**satisfaction** l_2)})

b)

(**operator** `Drive-truck`
 (**params** (?*v* `vehicle`) (?*o* `location`) (?*d* `location`)
  (?*r*1 `major-road`) (?*r*2 `track`))

 {**when** (**preconds** (*at* ?*v* ?*o*) (*connects* ?*r*1 ?*o* ?*d*))
  (**effects** (*not* (*at* ?*v* ?*o*)) (*at* ?*v* ?*d*))
  (**satisfaction** $l_\top$)}

 {**when** (**preconds** (*at* ?*v* ?*o*) (*connects* ?*r*2 ?*o* ?*d*))
  (**effects** (*not* (*at* ?*v* ?*o*)) (*at* ?*v* ?*d*))
  (**satisfaction** l_1)})

c)

(**goal** pkg_1-destination
 (**when** (*at* pkg_1 c_3) $l_\top$))

(**goal** pkg_2-destination
 (**when** (*at* pkg_2 c_2) l_1)
 (**when** (*at* pkg_2 c_3) $l_\top$))

Figure 11: Flexible Versions of 2 Operators and Goals for the Example Problem

Figure 11*a* shows the flexible operator `Load-truck` which considers the value of the package (in this problem case the propositions 'valuable pkg_1 k_2' and 'valuable pkg_2 k_1' are asserted) and

the presence of a guard. If the package is not very valuable as is the case for pkg_2, the guard's presence is immaterial - a satisfaction degree of $l_\top$ is assigned. If the package is quite valuable, as is the case for pkg_1, loading it onto the truck without a guard damages the plan, resulting in a lower satisfaction degree, l_2.

Figure 11*b* shows the flexible `Drive-truck` operator that expresses the damage done to a plan by sending a truck across the dangerous track, while indicating that using the major roads has no damaging effect on the plan. The other operators used are the same as for the original problem, using inflexible preconditions and assigning a satisfaction degree of $l_\top$. The plan goals, shown in figure 11*c*, remain to transport both packages to c_3. However, since pkg_2 is not very valuable it has a lower priority and one possible compromise is to deliver only pkg_1, although this does considerable damage to the plan satisfaction degree.

4.3 Flexible Graph Expansion

In order to describe the flexible planning graph expansion process given an input flexible planning problem, mutual exclusivity is first defined in this context. Two flexible propositions are labelled as exclusive if either they express a different truth degree for the same core proposition or (as per Graphplan) all ways of creating one are exclusive of all ways of creating the other. Mutual exclusion constraints between action nodes express that no valid plan could contain both actions. Two flexible actions in $actions_i$ are mutually exclusive for the same basic reasons as in the Graphplan framework:

Inconsistent Effects An effect of one action in $propositions_i$ expresses a different truth degree concerning the same core proposition to the effect of the other action in $propositions_i$.

Interference An effect of one action in $propositions_i$ expresses a different truth degree concerning the same core proposition to a necessary precondition of the other action in $propositions_{i-1}$.

Competing Needs The actions have mutually exclusive preconditions.

In flexible graph expansion, initial conditions are again placed in the first proposition level of the graph ($level_0$). A generic action level is generated as follows. Each clause of each flexible operator is instantiated in all possible ways to propositions of the previous level. Provided that all preconditions are mutually consistent, an action instance with the associated satisfaction is added to the planning graph for each such instantiation. `Noop` actions are added in the same manner as for Graphplan. Mutual exclusion relations are added to the planning graph according to the conditions defined above.

For the example problem, $levels_{1,2}$ of the flexible planning graph are shown in figure 12. This graph is relatively complex compared to the boolean planning graph shown in figure 2 since it includes the compromise options afforded by the flexible operators (see figure 11). The extra complexity is, however, offset by the fact that a first 'compromise' plan can be extracted as early

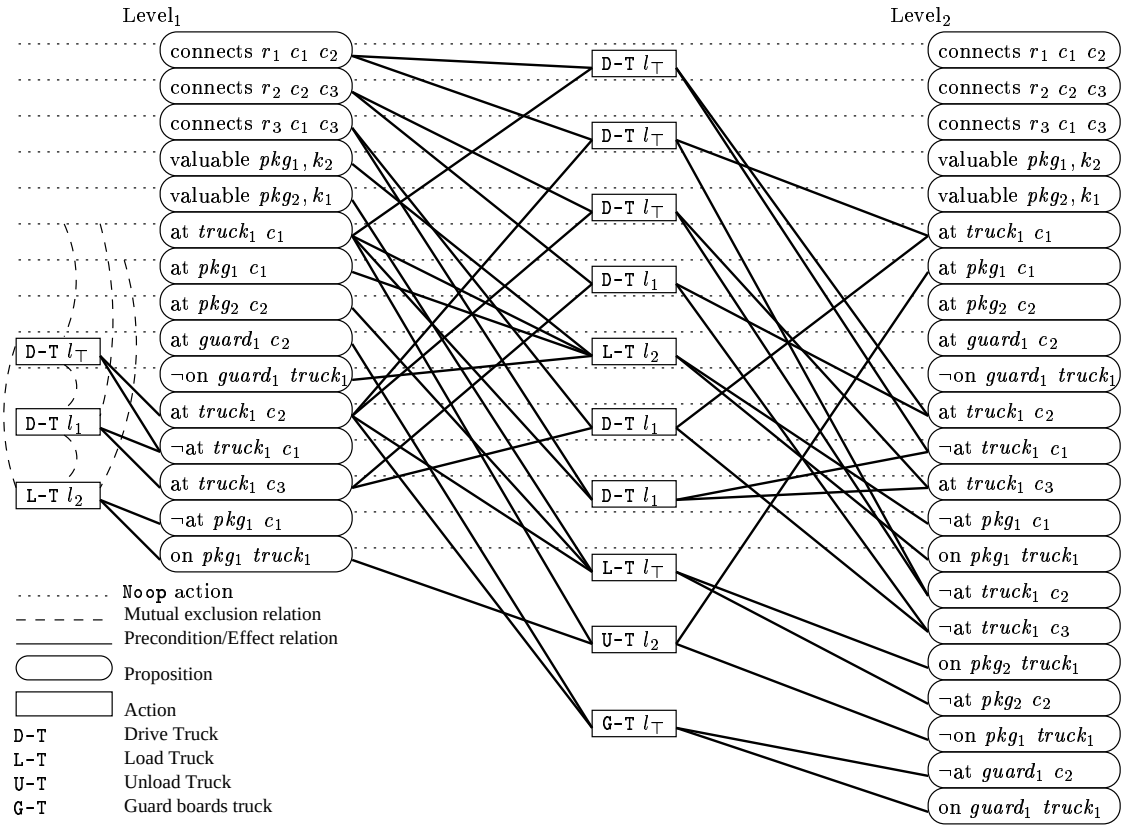


Figure 12: Flexible Planning Graph $Levels_{1,2}$ for the Flexible Version of the Problem given in Figure 1. For clarity, mutual exclusion relations are shown only in $actions_1$. Note that all pairs of non-Noop nodes in $actions_2$ except {G-T $l_\top$, L-T $l_\top$} are mutually exclusive.

as $level_3$, as detailed in section 5.2. By contrast, recall that an optimal plan synthesised for the original boolean version of the problem contains eight steps.

4.4 Limited Graph Expansion

Clearly, flexible graph expansion has an increased cost compared with boolean graph expansion. However, this cost can be reduced by imposing limits over the expansion process. Consider the case where a plan of satisfaction degree l_i has been found. If $l_i < l_\top$, FGP searches onwards in order to look for a plan with a higher satisfaction degree. Yet, it can be seen that there is no point in instantiating flexible operator clauses with a satisfaction degree less or equal to l_i: a plan with this satisfaction degree has been found already - a longer plan with the same satisfaction degree is deemed to be of a lower quality. The conjunctive combination rule implemented via the *min* operator ensures that no plan of satisfaction degree l_j can contain an action of satisfaction l_i, where $l_i < l_j$. Hence, the completeness of the search is not affected by omitting such actions in future flexible planning graph levels.

4.5 Satisfaction Propagation

A further step which can considerably reduce the search effort required during plan extraction is to propagate satisfaction degrees forwards as the graph is expanded, exploiting the fact that conjunctive combination is defined using the *min* operator. While expanding the graph to $level_i$, keep track of the maximum satisfaction degree of all the actions that produce each node in $propositions_i$ as an effect. Then, when expanding the graph to $level_{i+1}$ the satisfaction degree of a node in $actions_{i+1}$ is the minimum of the satisfaction degree generated by the instantiated operator clause and the minimum of maximum satisfaction degrees attached to each of its preconditions. This is possible because an activity constraint specifies that an action must be supported by all of its preconditions and the conjunctive combination rule guarantees that the overall satisfaction degree of an action plus its supported preconditions is the minimum satisfaction degree over all the actions involved.

Search during plan extraction is thus reduced because it will be obvious at an earlier stage (since extraction starts at the highest current level) that, say, a certain action can never be part of a plan with satisfaction degree $l_\top$. This process may be seen in figure 12: the `Unload-truck` action requires 'on pkg_1 $truck_1$' as a precondition. The sole way of asserting this proposition at $level_1$ is via a `Load-truck` action which has a satisfaction degree of l_2. The other precondition of `Unload-truck`, 'at $truck_1$ c_1', is an initial condition, so the minimum of maximum satisfaction degrees attached to the preconditions of this action is l_2. Hence, the satisfaction degree of `Unload-truck` at $level_2$ is the minimum of l_2 and the satisfaction degree of the instantiated operator clause ($l_\top$), resulting in a satisfaction degree of l_2 as shown in the figure. Similarly, the satisfaction degrees attached to several of the `Noop` nodes in $actions_2$ are revised and a satisfaction degree of l_1 is assigned to the `Drive-truck` action which moves the truck from c_3 to c_2.

5 Flexible Plan Extraction via rrDFCSP

Flexible plan extraction starts when a set of mutually-consistent flexible propositions exist in the current level which, to some extent, satisfy the flexible plan goals. Extracting a plan from a flexible planning graph may be viewed as a generalisation of extracting a plan in the original Graphplan framework. If only two satisfaction degrees are used, $l_\bot$ and $l_\top$, and similarly only two truth degrees, $k_\bot$ and $k_\top$, then the problem reduces to a boolean problem solvable by Graphplan.

In general, however, the flexible planning graph contains extra information in the form of the satisfaction degree associated with each action. A unary *preference* constraint [28] is constructed which specifies a preference degree for each of the assignments that satisfy it. Boolean binary constraints are generated from the mutual-exclusion relations in the planning graph as usual.

Utilising the recently-developed techniques of dynamic flexible CSP [29] (rrDFCSP herein, since the adopted techniques are also based on restriction and relaxation) flexible plan extraction can also be viewed as solving a linked hierarchy of dynamic problem sequences. The Flexible Local Changes (FLC) rrDFCSP solution algorithm [29] is used to solve each sequence. FLC, like Local

Changes, is a repair-based algorithm which attempts to minimise the effort required in solving the current flexible problem in a dynamic sequence by maintaining as much of the solution to the previous flexible problem as possible.

5.1 Generalisation of Plan Extraction via Banded Solution Extraction

In order to generalise the plan extraction procedures developed as part of GP-*rr*DCSP described in section 3, an approach is adopted which separates both the solution extraction process and the various available heuristic information sources into satisfaction degree *bands*. The exploration of the space of sub-problems is fundamentally the same as is described in figure 8, but utilising a branch and bound [24] technique (since each sub-problem is satisfied to a degree) to find the combination of sub-problems which may be optimally solved conjunctively.

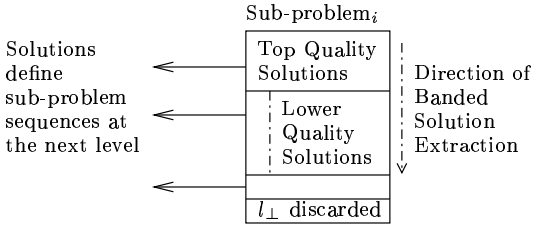


Figure 13: Banded Solution Extraction of a Sub-problem

Since it is desirable to find plans with higher satisfaction degrees as early as possible, the plan extraction process is organised to consider goal combinations and sub-problem solutions in descending order of satisfaction. Given an identified set of mutually-consistent flexible goals at $level_g$, the task is now to find the best possible path back to the initial conditions. Figure 13 shows how the solution of each sub-problem$_i$ where $0 < i \leq g$ proceeds. Ordering sub-problem solution in this way limits the effects of wasting much effort on search paths of a low satisfaction which may have been pruned much earlier. Indeed, this technique leads to an improvement in solution extraction efficiency in the flexible context on a similar basis to that for limiting graph expansion. Once a plan of satisfaction degree l_i has been found, all search paths which necessarily lead to a plan of lower or equal satisfaction can be pruned. Hence, the banded extraction of solutions need only extend to l_j such that $l_j > l_i$, guaranteed by the use of the *min* operator.

The memoisation-based techniques used to increase the efficiency of the search process described in section 3 can similarly be generalised to the flexible context. Memosets at each level are separated into distinct satisfaction degree bands. In particular, a memoset at $level_i$ associated with satisfaction degree l_j conveys the information that a flexible plan cannot exist of satisfaction degree l_j that includes the proposition nodes in the memoset at this level.

5.2 Synthesised Plans for the Flexible Example Problem

Three plans can be synthesised for the flexible example problem. The first short plan is as follows, with the major compromises being the route across the unsafe track and the fact that only pkg_1 is delivered.

1. `LOAD-TRUCK` pkg_1 $truck_1$ (l_2)
2. `DRIVE-TRUCK` $truck_1$ c_1 to c_3 via r_3 (l_1)
3. `UNLOAD-TRUCK` pkg_1 $truck_1$ $(l_\top)$

Using the main roads to avoid the unsafe track, both packages may be carried to c_3 in 5 steps as follows, with a satisfaction degree of l_2 since the guard is not present when pkg_1 is loaded.

1. `LOAD-TRUCK` pkg_1 $truck_1$ (l_2)
2. `DRIVE-TRUCK` $truck_1$ c_1 to c_2 via r_1 $(l_\top)$
3. `LOAD-TRUCK` pkg_2 $truck_1$ $(l_\top)$
4. `DRIVE-TRUCK` $truck_1$ c_2 to c_3 via r_2 $(l_\top)$
5. - `UNLOAD-TRUCK` pkg_1 $truck_1$ $(l_\top)$
   - `UNLOAD-TRUCK` pkg_2 $truck_1$ $(l_\top)$

The 'no compromise' takes 7 steps, with the truck first being driven to c_2 to load pkg_2 and fetch the guard before returning to c_1 to load pkg_1 and finally transporting both packages to c_3 using the main roads only. This simple example demonstrates how a range of plans of different lengths and containing alternative compromises can be synthesised from a given flexible planning problem. The user can then select the one which is deemed to offer the best compromise between length and satisfaction degree.

6 Experimental Results

This section presents three groups of experimental results, to demonstrate the success of the present work.

6.1 Boolean Plan Synthesis

GP-*rr*DCSP is first of all evaluated against leading boolean solvers to establish the efficacy of the novel hierarchical *rr*DCSP approach used. Table 2 shows the results obtained when running GP-*rr*DCSP, Graphplan, STAN[1] [25], IPP v4 [22] and BLACKBOX 3.6b (SATZ-RAND) [19] on benchmark problems. Note that GP-*rr*DCSP is implemented in Java and the other solvers in C.

GP-*rr*DCSP performs strongly on these benchmarks, being competitive over the whole range of problems. STAN and IPP contain alternative enhancements to the Graphplan framework not

[1] The latest version of STAN available to the authors, version 4, was used on the logistics and rockets problems but failed to synthesise the plans for the blocks world problems. For these problems, version 3 was used.

included in GP-*rr*DCSP. An opportunity exists to achieve further performance improvements by integrating these techniques with the existing approach. The strong average performance from the BLACKBOX solver on these boolean problems suggests the exploration of the compilation of the more complex flexible planning graph to a propositional satisfiability problem. Preliminary, and promising, results in this regard are reported in [17]. The size and complexity of these benchmarks is, however, relatively small compared to real problems. Further improvements are necessary for this approach to scale to such problems.

Problem	Length	GP-*rr*DCSP	GP	STANv3/v4	IPP	BLACKBOX (SATZ-RAND)
Rocket-a	7	14	75	33	47	5
Rocket-b	7	21	154	2	76	8
Logistics-a	11	8	1955	1	1513	7
Logistics-b	13	99	862	2	633	24
Logistics-c	13	123	-	704	-	46
Logistics-d	14	171	-	7096	-	108
bw-12step	12	1	<1	<1	<1	6
bw-Large-a	12	20	3	<1	<1	48
bw-Large-b	18	100	198	32	15	-

Table 2: Boolean problem comparison (in seconds) between FGP and leading solvers. A dash indicates no solution found in 24 hours. Hardware used: Sun Ultra 5.

6.2 Flexible vs Boolean Plan Synthesis

The second series of experiments examine the overhead of looking for a range of solutions as compared with extracting a single plan from a boolean version of the same problem. The test suite contains 12 problems of a similar type (using the same operators) to the example problems used throughout this paper, with a boolean and flexible version of each problem. The flexible problems use truth and satisfaction degree scales: $K = \{k_\perp, k_1, k_2, k_\top\}$ and $L = \{l_\perp, l_1, l_2, l_\top\}$. Using flexible plan synthesis, three different plans may be produced, whilst the boolean problems contain only the equivalent of the plan with satisfaction degree $l_\top$.

As an example, figure 14 shows an overview of the second problem in the test suite. The target is to transport both packages to their respective destinations. The other problems in the test suite are similar, but with more/less complex internal connections in the three cities. The composition of the problems, in terms of plan lengths at each satisfaction degree, are shown in figure 15.

Figure 15 also presents the comparative run-times for FGP on the test suite of problems. As expected, it always takes longer to find a compromise-free plan when also searching for shorter 'compromise' plans than it does to simply solve the boolean problem. However, the time taken to produce shorter compromise plans is significantly lower than to solve the boolean problem, providing an attractive 'anytime' behaviour. This advantage is only reduced when the lengths

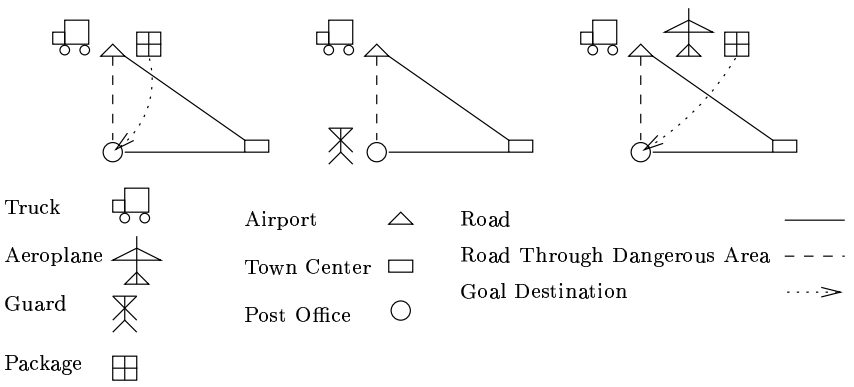


Figure 14: Flexible Problem 2 (of 12) for Comparing Boolean/Flexible Extraction

of compromise plans and the plan with satisfaction degree $l_\top$ are similar. The three pronounced peaks visible in the run-times in figure 15 directly correspond to an increase in the difference in length between the plans of satisfaction degree l_1 and l_2. Such problems are more difficult due to the reduced effectiveness of limited graph expansion and satisfaction propagation. The advantages of having several plans trading compromises made against plan length versus a single 'no compromise' plan, however, outweigh the extra search cost involved.

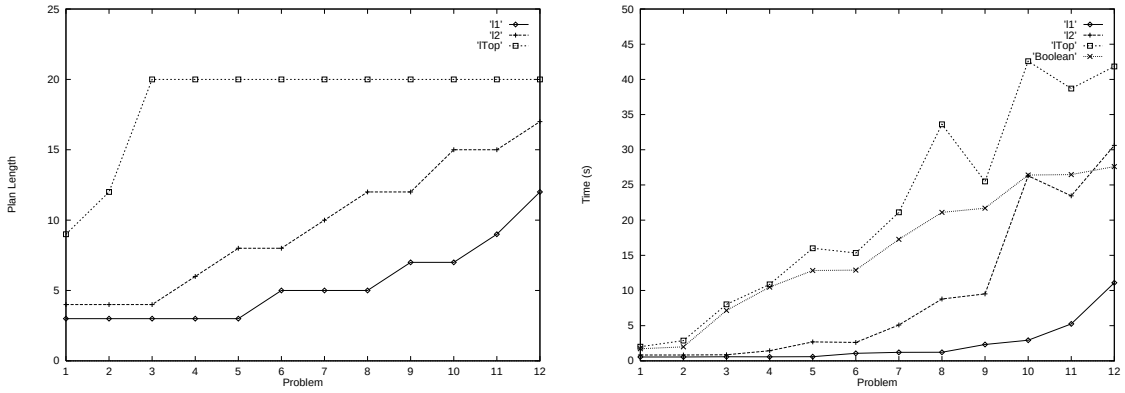


Figure 15: Test Suite of Flexible Problems: Composition and Run-times

6.3 The Rescue Problem

To demonstrate the utility of flexible planning on a more substantial problem, the example given in figure 16 is used. The scenario considered is a rescue operation: a volatile volcano on Volcano Island has started to erupt. The target is to evacuate the scientists and civilians present on the island and *preferably* their equipment and belongings too. There are three points to make the evacuation to, with ascending degrees of satisfaction: the (relatively) 'Safe Point' on the other side of Volcano Island, the nearby village on Neighbour Island and finally the safety of Far Island. A minimum target is to evacuate just the people to Safe Point. Slightly better is to also save the

equipment and possessions. A target with a higher satisfaction is to remove people and possessions to Neighbour Village. The 'no compromise' target is to safely evacuate all people to the safety of Far Island and make sure that all equipment and possessions are removed from Volcano Island.

Apart from the plan goals, there are several other constraints and sources of flexibility which lead to a range of different plans. Firstly, the helicopter is small and usually carries only one passenger. Because of the emergency, however, it is possible to get two passengers on board, although this is not a preferred option. There is no facility to carry equipment/belongings in this vehicle. The pilot would also like to avoid Volcano Island, if at all possible. The trucks and boat can carry people or possessions. One of the routes that the boat may take between Volcano Island and Neighbour Island is treacherous, and to be avoided if possible.

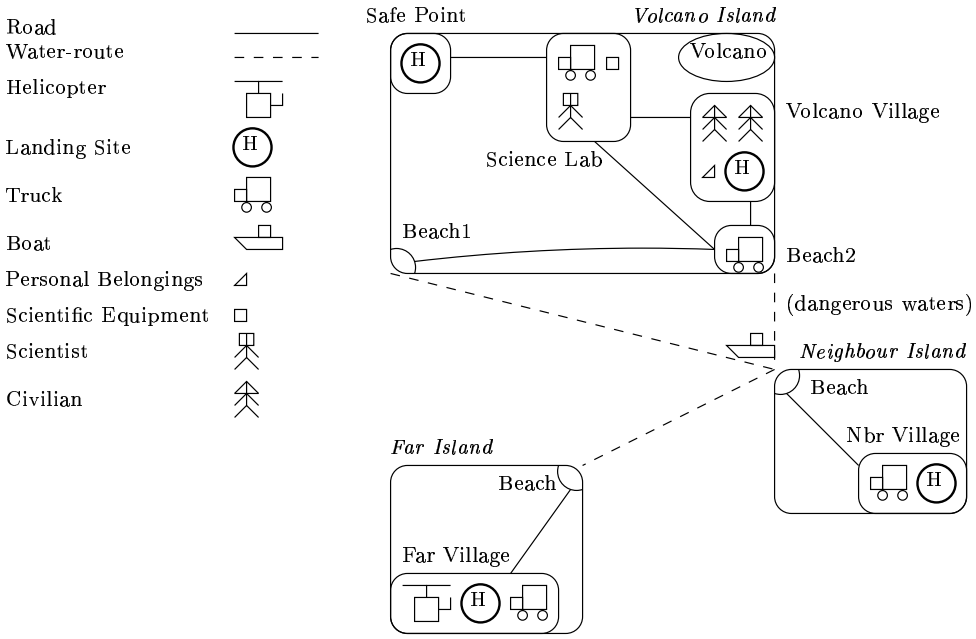


Figure 16: Overview of the Rescue Problem

This problem uses the satisfaction scale $L = \{l_\bot, l_1, l_2, l_3, l_4, l_\top\}$ and a truth degree scale, K, of the same size (though a different size could easily be employed). The exact details of the problem and domain specification are presented in appendix A. This is not a trivial problem: there are 432 possible flexible goal combinations alone. FGP is able to synthesise 5 plans for the rescue problem in under 30 seconds, one corresponding to each satisfaction degree, discounting $l_\bot$. This is not the case for all problems since, as noted in section 4, given two plans of the same length the one with the higher satisfaction degree is returned. There are, of course, alternative plans with the same satisfaction degree and length as the ones shown, but this is no different to the situation found in boolean planning.

The shortest plan of four steps is shown below. Although the other actions in the plan are effectively 'drowned' by the fact that the helicopter must fly to Volcano Village, by exploring the

highest quality goals first the effects of this have been counteracted: the helicopter is flown to Far Village when strictly speaking a plan with an equivalent satisfaction degree would be to take the civilians to the Safe Point.

1. `LOAD-TRUCK` lab-truck scientific-equipment ($l_\top$)
   `LOAD-TRUCK` lab-truck scientist ($l_\top$)
   `FLY-HELICOPTER` Far Village, Volcano Village (l_1)
2. `LOAD-HELICOPTER` civilian1, civilian2 (l_4)
   `DRIVE-TRUCK` lab-truck Science-lab, Safe Point ($l_\top$)
3. `UNLOAD-TRUCK` lab-truck scientific equipment ($l_\top$)
   `UNLOAD-TRUCK` lab-truck scientist ($l_\top$)
   `FLY-HELICOPTER` Volcano Village, Far Village ($l_\top$)
4. `UNLOAD-HELICOPTER` civilian1, civilian2 ($l_\top$)

By contrast, a 'no compromise' plan, an example of which is shown below, takes 12 steps to perform. This plan is so much longer since the preferences of the helicopter pilot to avoid Volcano Island and the boat's captain to avoid the dangerous route between Volcano Beach2 and Neighbour Beach must both be satisfied.

1. `LOAD-TRUCK` lab-truck scientist1 ($l_\top$)
   `LOAD-TRUCK` lab-truck scientific equipment ($l_\top$)
   `DRIVE-TRUCK` v-truck Volcano Beach2, Volcano Village ($l_\top$)
2. `LOAD-TRUCK` v-truck personal possessions ($l_\top$)
   `LOAD-TRUCK` v-truck civilian1 ($l_\top$)
   `LOAD-TRUCK` v-truck civilian2 ($l_\top$)
   `DRIVE-TRUCK` lab-truck Science Lab, Volcano Beach 2 ($l_\top$)
3. `SAIL-BOAT` Neighbour Beach, Volcano Beach1 ($l_\top$)
   `DRIVE-TRUCK` v-truck Volcano Village, Volcano Beach2 ($l_\top$)
   `DRIVE-TRUCK` v-truck Volcano Beach2, Volcano Beach1 ($l_\top$)
4. `UNLOAD-TRUCK` lab-truck scientist ($l_\top$)
   `UNLOAD-TRUCK` lab-truck scientific equipment ($l_\top$)
   `DRIVE-TRUCK` v-truck Volcano Beach2, Volcano Beach1 ($l_\top$)
5. `LOAD-BOAT` scientist ($l_\top$)
   `LOAD-BOAT` scientific equipment ($l_\top$)
   `UNLOAD-TRUCK` v-truck civilian1 ($l_\top$)
   `UNLOAD-TRUCK` v-truck civilian2 ($l_\top$)
   `UNLOAD-TRUCK` v-truck personal possessions ($l_\top$)
6. `LOAD-BOAT` civilian1 ($l_\top$)
   `LOAD-BOAT` civilian2 ($l_\top$)
   `LOAD-BOAT` personal possessions ($l_\top$)
7. `SAIL-BOAT` Volcano Beach1, Neighbour Beach ($l_\top$)
   `DRIVE-TRUCK` f-truck Far Village, Far Beach ($l_\top$)
8. `SAIL-BOAT` Neighbour Beach, Far Beach ($l_\top$)
9. `UNLOAD-BOAT` scientist ($l_\top$)
   `UNLOAD-BOAT` civilian1 ($l_\top$)
   `UNLOAD-BOAT` civilian2 ($l_\top$)
   `UNLOAD-BOAT` scientific equipment ($l_\top$)
   `UNLOAD-BOAT` personal possessions ($l_\top$)
10. `LOAD-TRUCK` f-truck scientist ($l_\top$)
    `LOAD-TRUCK` f-truck civilian1 ($l_\top$)
    `LOAD-TRUCK` f-truck civilian2 ($l_\top$)
    `LOAD-TRUCK` f-truck scientific equipment ($l_\top$)
    `LOAD-TRUCK` f-truck personal possessions ($l_\top$)

11. DRIVE-TRUCK f-truck Far Beach, Far Village ($l_\top$)
12. UNLOAD-TRUCK f-truck scientist ($l_\top$)
    UNLOAD-TRUCK f-truck civilian1 ($l_\top$)
    UNLOAD-TRUCK f-truck civilian2 ($l_\top$)
    UNLOAD-TRUCK f-truck scientific equipment ($l_\top$)
    UNLOAD-TRUCK f-truck personal possessions ($l_\top$)

The other synthesised plans, of lengths 5, 9 and 11 steps respectively, are detailed in appendix A. Each plan is returned as soon as it is found, providing an anytime solution to the problem. Having extracted all five solutions to the Rescue problem, however, each offers a different balance between compromises made and the length of the plan. A user may then select and execute the one which is considered to make the optimal tradeoff. The Rescue problem has thus provided further evidence of the utility of flexible planning when faced with the time limits and soft constraints which often pervade real problems.

7 Related Work

The assumption that an intelligent agent possesses complete and correct information has received much attention in the form of contingent and conformant AI planning research. In contingent planning a planner is equipped with sensing actions that can be used to determine the state of the world during plan execution [1, 14, 36, 37]. In conformant planning a planner is provided with knowledge about the possible states the world can be in at the start of execution and the possible outcomes of executing each action [5, 23, 39]. These approaches are complementary and their integration has been explored [9, 15].

The proposed flexible approach is distinct from but complementary to contingent and conformant techniques. With respect to contingent approaches, FGP makes the closed-world assumption and is not furnished with information gathering actions. However, the flexible propositions used complement the natural imprecision in real-world sensing actions as has successfully been exploited in the area of fuzzy control [35]. Conformant planners model uncertainty as a distribution over the effects of a given operator yet the propositions themselves are exclusively true or false. Within the proposed approach, flexible propositions are assigned subjective truth degrees and operators map deterministically from the space of flexible preconditions onto a set of flexible effects and a satisfaction degree. Therefore, subjective knowledge can be utilised to entail inferences over preferred combinations of actions.

Recently, probability-based planners have attracted significant attention in the literature. It might be conjectured that a probabilistic formalism could be overloaded so that probabilities attached to effects would signify operator satisfaction degrees. Invoking a probabilistic planner on such an overloaded domain description highlights the different semantics of satisfaction degrees and probabilities. For example, PGraphplan [5] produces a contingent plan that recommends repeated application of an operator in event of it not producing the desired effects on its first application. This is not correct in the flexible context as repeating an action will not affect its satisfaction

degree. While a probabilistic planner will not correctly solve flexible planning problems, the approaches are complementary and could be combined. An investigation into this remains as an important piece of future work.

Also recently, applied planning research explored a mixed-initiative paradigm where the user is supported in exploring the range of plans that can be used to achieve a task. In TRIPS [11] and O-Plan [41], the user is expected to interact extensively with the planner to explore the solution space. However, a user may not understand the solution space being explored and vital solution areas may be missed. The present research is in line with the developments of the the SIPE planner [45], reported in [33], where the system automatically produces a number of courses of action. The important difference is that FGP can automatically find a range of plans given a flexible planning problem using a single run, as opposed to the multiple runs required by SIPE.

There has also been recent work [20, 43] on incorporating numerically weighted constraints into planning to give a quantitative means of differentiating between different potential plans, as opposed to the qualitative methods in this paper. However, should there be knowledge available regarding the truth and/or satisfaction degrees in actual numerical terms the present approach can be readily adapted to make use of such knowledge without changing the underlying representation mechanism.

8 Conclusion

This paper has considered the AI Planning problem [44], in terms of both the techniques to improve the efficiency of plan synthesis and the extension of the classical domain definition to support soft constraints. A novel hierarchical structuring of the plan extraction process in the Graphplan framework [4] was presented in order to realise efficiency gains and as a means to support the synthesis of flexible plans. It was first shown how, by viewing a sequence of sub-problems at a particular level of the planning graph as a restriction/relaxation-based dynamic constraint satisfaction problem (rrDCSP), efficiency gains could be realised via the use of advanced search algorithms to solve such problems.

The discovery and propagation of small *memosets*, containing conjunctively unsupportable proposition nodes with respect to a particular level of the planning graph, was shown to further increase the efficiency of this process. This is achieved both by pruning the search in the space of sub-problems and by facilitating the repair of existing complete solutions to sub-problems rather than the simple enumeration of every such solution. Focus heuristics were also described in order to aid the plan extraction mechanism in finding the most useful memosets to learn. A solution mechanism based on these techniques, GP-rrDCSP, was tested on standard benchmarks against leading boolean solvers. The results demonstrate that the techniques proposed herein consistently improve the performance of plan extraction on a range of benchmarks significantly.

The novel *flexible planning problem* was introduced, a representation designed to capture the

inherent 'softness' found in many real world problems. This is facilitated through the assignment of subjective truth degrees to propositions which are exploited by flexible operators that express one of a range of satisfaction degrees depending on how well their preconditions are satisfied. A solution strategy, called Flexible Graphplan (FGP), which extends the Graphplan framework was established, utilising dynamic flexible constraint satisfaction problems (DFCSPs) [29] as a foundation for plan extraction.

It was demonstrated that plan extraction in this new framework is a generalisation of plan extraction in the boolean Graphplan framework via rrDCSP discussed earlier. Furthermore methods for reducing the added computational complexity of supporting flexible planning were discussed. These comprise the limitation of the addition of flexible action instances to those that could form part of a plan with a higher satisfaction degree than is currently known and the forwards-propagation of satisfaction degrees to aid the early pruning of the search space.

FGP was evaluated on a set of problems engineered to contain plans of various satisfaction degrees of different lengths to gauge the amount of extra effort required as compared to solving the boolean version of the same problem. Although the boolean problem was, unsurprisingly, always easier to solve than to extract plans of all satisfaction degrees from the flexible version, the time taken to produce short, low-satisfaction plans was lower still, providing an attractive 'anytime' behaviour. Furthermore, the overall quality of a plan containing compromises may even be higher than such a strict boolean perfect plan because of the differences in plan length. This was demonstrated on the more realistic Rescue problem.

One of the key challenges in developing plans in complex domains is to make rational trade-offs among conflicting objectives. The potential practical applications of a system based on flexible planning are therefore wide-ranging. The Rescue domain presented here is a simplified example of the real problem of planning non-combatant military evacuations. When deciding upon evacuation points and evacuation routes the examples used of avoiding dangerous landing sites and sea routes mirror the real world trade-offs. Commanders must also balance the preferences of avoiding ground routes that include bridges and providing armoured rather than soft-skinned vehicles in hostile situations while simultaneously trying to minimise resource usage and complete an evacuation quickly.

A second example is that of game-playing. Lacking the infeasible amount of time required to search an entire game tree, a flexible planner could instead look for a best compromise plan. In addition, since many games involve strict time limits, the ability to produce a compromise plan 'anytime' may be a crucial advantage in this domain. Further examples might extend the systems reported in [21]. It is not difficult to imagine the advantages of being able to compromise in a system to plan the emergency response of the coast guard to marine oil spills or in producing military air campaign plans, to quote two examples therein.

The work presented here lays an important foundation for the longer term goal of deploying the capability of reasoning about priorities and preferences in fielded planning tools. This paper

has developed a representation for this information, provided both a calculus for reasoning about the effect on plan satisfaction degree of violating preferences and a planning algorithm capable of producing a range of solutions containing different levels of compromise. Experimental results have identified the computational cost of including this class of reasoning, in particular the relationship between the number of satisfaction degrees modelled and the distribution of solutions in the domain. This understanding should provide the foundation for further work on including reasoning of this type in more application-oriented frameworks such as Hierarchical Task Network Planning [45].

Finally, as shown in section 6, the drowning effect of the idempotent *min* operator can be counteracted to an extent, but it would be preferable to have a finer resolution when comparing different plans (short of using a large number of satisfaction degrees). Leximin CSP [32] combines additive and possibilistic approaches to avoid this problem, although at a significantly higher computational cost. Making use of such an approach in the context of the flexible planning problem is an active research topic.

Acknowledgements

Ian Miguel is supported by UK-EPSRC grant number 97305803. Peter Jarvis was partly supported by DARPA contract F30602-97-C-0067 under the supervision of the AFRL-Rome. The authors are grateful to the anonymous referees for providing helpful comments on improving the work presented here.

References

[1] J.A. Ambros-Ingerson and S. Steel. Integrating planning, execution and monitoring. *Proceedings of the 7th National Conference on Artificial Intelligence*, pages 83–88, 1988.

[2] M. Baioletti, S. Marcugini, and A. Milani. Dpplan: an algorithm for fast solutions extraction from a planning graph. *Proceedings of the 5th International Conference on Artificial Intelligence Planning*, 2000.

[3] C. Bessiere, E.C. Freuder, and J. Regin. Using inference to reduce arc consistency computation. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 592–598, 1995.

[4] A. Blum and M. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

[5] A. Blum and J. Langford. Probabilistic planning in the graphplan framework. *Proceedings of the 5th European Conference on Planning Systems, Durham*, pages 320–332, 1999.

[6] S. Chien. Static and completion analysis for knowledge acquisition, validation, and maintenance of planning knowledge bases. *Journal of Human Computer Studies*, 48:499–519, 1998.

[7] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[8] R. Dechter and A. Dechter. Belief maintenance in dynamic constraint networks. *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 37–42, 1988.

[9] D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems*, pages 31–36, 1994.

[10] D. Dubois, H. Fargier, and H. Prade. Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty. *Applied Intelligence*, 6:287–309, 1996.

[11] G. Ferguson and J. Allen. TRIPS: An intelligent integrated problem-solving assistant. *Proceedings of the 15th National Conference on Artificial Intelligence*, pages 567–573, 1998.

[12] R. Fikes and N. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

[13] M.L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[14] R. Goldman and M. Boddy. Conditional linear planning. *Proceedings of the 2nd international conference on AI Planning Systems*, pages 80–85, 1994.

[15] R. Goldman and M. Boddy. Epsilon-safe planning. *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, pages 253–261, 1994.

[16] R.M. Haralick and G.L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[17] P. Jarvis, I. Miguel, and Q. Shen. Flexible blackbox. *Proceedings of the AAAI-2000 Workshop on Representational Issues for Real-World Planning Systems*, pages 43–49, 2000.

[18] S. Kambhampati. Planning graph as (dynamic) CSP: Exploiting EBL, DDB and other CSP techniques in graphplan. *Journal of AI Research*, 12:1–34, 2000.

[19] H. Kautz and B. Selman. Unifying SAT-based and graph-based planning. *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 318–325, 1999.

[20] H. Kautz and J. Walser. State-space planning by integer optimization. *Proceedings of the 16th National Conference on Artificial Intelligence*, pages 526–533, 1999.

[21] C. Knoblock. Ai planning systems in the real world. *IEEE Expert*, pages 4–12, 1996.

[22] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an adl subset. *Proceedings of the 4th European Conference on Planning*, pages 273–285, 1997.

[23] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76(1-2):239–286, 1995.

[24] E.L. Lawler and D.E. Wood. Branch-and-bound methods: a survey. *Operations Research*, 14:699–719, 1966.

[25] D. Long and M. Fox. Efficient implementation of the plan graph in STAN. *Journal of AI Research*, 10:87–115, 1998.

[26] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[27] I. Miguel, P. Jarvis, and Q. Shen. Flexible graphplan. *Proceedings of the 14th European Conference on Artificial Intelligence*, pages 506–510, 2000.

[28] I. Miguel and Q. Shen. Hard, flexible and dynamic constraint satisfaction. *Knowledge Engineering Review*, 14(3):199–220, 1999.

[29] I. Miguel and Q. Shen. Dynamic flexible constraint satisfaction. *Applied Intelligence*, 13(3):231–245, 2000.

[30] S. Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 25–32, 1990.

[31] R. Mohr and T. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[32] J. Moura Pires, F. Moura Pires, and R. Almeida Ribeiro. Structure and properties of leximin FCSP and its influence on optimisation problems. *Proceedings of the 7th Conference on Information Processing and the Management of Uncertainty*, pages 188–194, 1998.

[33] K. Myers and T. Lee. Generating qualitatively different plans through metatheoretic biases. *Proceedings of the 16th National Conference on Artificial Intelligence*, pages 570–576, 1999.

[34] E. Pednault. ADL: Exploring the middle ground between strips and the situation calculus. *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR-89)*, pages 324–332, 1989.

[35] W. Pedrycz and F. Gomide. *An Introduction to Fuzzy Sets: Analysis and Design*. MIT Press, 1999.

[36] M. Peot and D. Smith. Conditional non-linear planning. *Proceedings of the 1st International Conference on Artificial Intelligence Planning Systems*, pages 189–197, 1992.

[37] L. Pryor and G. Collins. Planning for contingencies: A decision based approach. *Journal of AI Research*, 4:287–339, 1996.

[38] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 631–637, 1995.

[39] D. Smith and D. Weld. Conformant graphplan. *Proceedings of the 15th National Conference on Artificial Intelligence*, pages 889–896, 1998.

[40] S. Smith, D. Nau, and T. Throop. Total-order multi-agent task-network planning for contract bridge. *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 108–113, 1996.

[41] A. Tate, J. Dalton, and J. Levine. Generation of multiple qualitatively different plans. *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems*, pages 27–34, 1998.

[42] G. Verfaillie and T. Schiex. Solution reuse in dynamic constraint satisfaction problems. *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 307–312, 1994.

[43] T. Vossen, M. Ball, A. Lotem, and D. Nau. Applying integer programming to ai planning. *Knowledge Engineering Review*, 15(1):85–100, 2000.

[44] D.S. Weld. Recent advances in ai planning. *AI Magazine*, 20(2):93–123, 1999.

[45] D. Wilkins. *Practical Planning.* Morgan Kaufmann, 1988.

[46] D. Wilkins and K. Myers. A multiagent planning architecture. *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems*, pages 154–162, 1998.

A The Rescue Problem in Detail

A.1 Domain Operators

A.1.1 Scales

```
(SatScale lbot l1 l2 l3 l4 ltop)
(TruthScale kbot k1 k2 k3 k4 ktop)
```

A.1.2 Helicopter Operators

```
// overloading is possible, but not preferred
// Helicopter is small and only has room for people, not belongings.
(operator LOAD-HELICOPTER
 (params (?h HELICOPTER) (?loc LOCATION) (?p1 PERSON) (?p2 PERSON))
 {when (preconds (at ?h ?loc) (at ?p1 ?loc) (empty ?h == ktop))
  (effects (not (at ?p1 ?loc)) (on ?p1 ?h) (empty ?h k2))
```

```
  ltop}
 {when (preconds (at ?h ?loc) (at ?p1 ?loc) (empty ?h == k2))
  (effects (not (at ?p1 ?loc)) (on ?p1 ?h) (empty ?h kbot))
  l4}
 {when (preconds (at ?h ?loc) (at ?p1 ?loc) (at ?p2 ?loc) (empty ?h == ltop))
  (effects (not (at ?p1 ?loc)) (not (at ?p2 ?loc)) (on ?p1 ?h)
           (on ?p2 ?h) (empty ?h kbot))
  l4}
)

(operator UNLOAD-HELICOPTER
 (params (?h HELICOPTER) (?loc LOCATION) (?p1 PERSON) (?p2 PERSON))
 {when (preconds (at ?h ?loc) (on ?p1 ?h) (empty ?h == kbot))
  (effects (at ?p1 ?loc) (not (on ?p1 ?h)) (empty ?h k2))
  ltop}
 {when (preconds (at ?h ?loc) (on ?p1 ?h) (empty ?h == k2))
  (effects (at ?p1 ?loc) (not (on ?p1 ?h)) (empty ?h ktop))
  ltop}
 {when (preconds (at ?h ?loc) (on ?p1 ?h) (on ?p2 ?h))
  (effects (at ?p1 ?loc) (not (on ?p1 ?h)) (at ?p2 ?loc)
           (not (on ?p2 ?h)) (empty ?h ktop))
  ltop}
)

// Will risk less suitable landing sites because of emergency.
(operator FLY-HELICOPTER
 (params (?h HELICOPTER) (?o LOCATION) (?d LOCATION))
 {when (preconds (at ?h ?o) (suitable-landing-site ?d >= k4))
  (effects (not (at ?h ?o)) (at ?h ?d))
  ltop}
 {when (preconds (at ?h ?o) (suitable-landing-site ?d range k2 k3))
  (effects (not (at ?h ?o)) (at ?h ?d))
  l3}
 {when (preconds (at ?h ?o) (suitable-landing-site ?d == k2))
  (effects (not (at ?h ?o)) (at ?h ?d))
  l1}
)
```

A.1.3 Truck Operators

```
(operator LOAD-TRUCK
 (params (?t TRUCK) (?loc LOCATION) (?i TRANSPORTABLE-ITEM))
 {when (preconds (at ?t ?loc) (at ?i ?loc))
  (effects (not (at ?i ?loc)) (on ?i ?t))
  ltop}
)

(operator UNLOAD-TRUCK
 (params (?t TRUCK) (?loc LOCATION) (?i TRANSPORTABLE-ITEM))
 {when (preconds (at ?t ?loc) (on ?i ?t))
  (effects (at ?i ?loc) (not (on ?i ?t))
  ltop}
)
```

```
(operator DRIVE-TRUCK
 (params (?t TRUCK) (?loc1 LOCATION) (?loc2 LOCATION) (?r1 ROAD))
 {when (preconds (at ?t ?loc1) (connects ?r1 ?loc1 ?loc2))
  (effects (not (at ?t ?loc1)) (at ?t ?loc2))
  ltop}
 {when (preconds (at ?t ?loc2) (connects ?r1 ?loc1 ?loc2))
  (effects (not (at ?t ?loc2)) (at ?t ?loc1))
  ltop}
)
```

A.1.4 Boat Operators

```
(operator LOAD-BOAT
 (params (?b BOAT) (?loc LOCATION) (?i TRANSPORTABLE-ITEM))
 {when (preconds (at ?b ?loc) (at ?i ?loc))
  (effects (not (at ?i ?loc)) (on ?i ?b))
  ltop}
)

(operator UNLOAD-BOAT
 (params (?b BOAT) (?loc LOCATION) (?i TRANSPORTABLE-ITEM))
 {when (preconds (at ?b ?loc) (on ?i ?b))
  (effects (at ?i ?loc) (not (on ?i ?b)))
  ltop}
)

// Will take a risk because of emergency situation.
(operator SAIL-BOAT
 (params (?b BOAT) (?loc1 LOCATION) (?loc2 LOCATION) (?wr WATER-ROUTE))
 {when (preconds (at ?b ?loc1) (connects ?wr ?loc1 ?loc2) (safe ?wr >= k3))
  (effects (not (at ?b ?loc1)) (at ?b ?loc2))
  ltop}
 {when (preconds (at ?b ?loc2) (connects ?wr ?loc1 ?loc2) (safe ?wr >= k3))
  (effects (not (at ?b ?loc2)) (at ?b ?loc1))
  ltop}
 {when (preconds (at ?b ?loc1) (connects ?wr ?loc1 ?loc2)
            (safe ?wr range k1 k2))
  (effects (not (at ?b ?loc1)) (at ?b ?loc2))
  l3}
 {when (preconds (at ?b ?loc2) (connects ?wr ?loc1 ?loc2)
            (safe ?wr range k1 k2))
  (effects (not (at ?b ?loc2)) (at ?b ?loc1))
  l3}
)
```

A.2 Problem Specification

A.2.1 Plan Objects

```
// Volcano Island:
(lab-truck TRUCK)
(v-truck TRUCK)
(science-lab LOCATION)
(volcano-village LOCATION)
```

```
(volcano-beach1 LOCATION)
(volcano-beach2 LOCATION)
(volcano-safe-point LOCATION)
(scientist PERSON)
(scientist TRANSPORTABLE-ITEM)
(scientific-equipment TRANSPORTABLE-ITEM)
(civilian1 PERSON)
(civilian1 TRANSPORTABLE-ITEM)
(civilian2 PERSON)
(civilian2 TRANSPORTABLE-ITEM)
(personal-possessions TRANSPORTABLE-ITEM)
(vroad1 ROAD)
(vroad2 ROAD)
(vroad3 ROAD)
(vroad4 ROAD)
(vroad5 ROAD)

// Neighbour Island:
(n-truck TRUCK)
(neighbour-village LOCATION)
(neighbour-beach LOCATION)
(nroad ROAD)

// Far Island:
(f-truck TRUCK)
(far-village LOCATION)
(far-beach LOCATION)
(froad ROAD)

// Water connections & boat:
(boat1 BOAT)
(wr1 WATER-ROUTE)
(wr2 WATER-ROUTE)
(wr3 WATER-ROUTE)

// Helicopter:
(helicopter1 HELICOPTER)
```

A.2.2 Initial Conditions

```
(preconds
// Volcano Island
 (connects vroad1 science-lab volcano-safe-point)
 (connects vroad2 science-lab volcano-village)
 (connects vroad3 science-lab volcano-beach2)
 (connects vroad4 volcano-village volcano-beach2)
 (connects vroad5 volcano-beach1 volcano-beach2)
 (suitable-landing-site volcano-safe-point k3)
 (suitable-landing-site volcano-village k1)
// Neighbour Island
 (connects nroad neighbour-beach neighbour-village)
 (suitable-landing-site neighbour-village)
// Far Island
 (connects froad far-beach far-village)
```

```
 (suitable-landing-site far-village ktop)
// Water Connections
 (connects wr1 volcano-beach1 neighbour-beach)
 (safe wr1 ktop)
 (connects wr2 volcano-beach2 neighbour-beach)
 (safe wr2 k2)
 (connects wr3 neighbour-beach far-beach)
 (safe wr3 ktop)
// Machines
 (at boat1 neighbour-beach)
 (at lab-truck science-lab)
 (at v-truck volcano-beach2)
 (at n-truck neighbour-village)
 (at f-truck far-village)
 (at helicopter1 far-village)
 (empty helicopter1 ktop)
// People/Items
 (at scientist science-lab)
 (at civilian1 volcano-village)
 (at civilian2 volcano-village)
 (at scientific-equipment science-lab)
 (at personal-possessions volcano-village)
)
```

A.2.3 Goals

```
(effects
 (goal scientist-destination
  (when (at scientist volcano-safe-point) l2)
  (when (at scientist neighbour-village) l4)
  (when (at scientist far-village) ltop))
 (goal civilian1-destination
  (when (at civilian1 volcano-safe-point) l2)
  (when (at civilian1 neighbour-village) l4)
  (when (at civilian1 far-village) ltop))
 (goal civilian2-destination
  (when (at civilian2 volcano-safe-point) l2)
  (when (at civilian2 neighbour-village) l4)
  (when (at civilian2 far-village) ltop))
 (goal scientific-eq
  (when (at scientific-equipment science-lab) l1)
  (when (at scientific-equipment volcano-safe-point) l3)
  (when (at scientific-equipment neighbour-village) ltop)
  (when (at scientific-equipment far-village) ltop))
 (goal possessions
  (when (at personal-possessions volcano-village) l1)
  (when (at personal-possessions volcano-safe-point) l3)
  (when (at personal-possessions neighbour-village) ltop)
  (when (at personal-possessions far-village) ltop))
)
```

A.3 Other Rescue Plans

This section contains plans of satisfaction degree l_2, l_3 and l_4 for the rescue problem not detailed in the main text.

A.3.1 Plan with Satisfaction Degree l_2

1. DRIVE-TRUCK v-truck Volcano Beach2, Volcano Village ($l_\top$)
   LOAD-TRUCK lab-truck scientific equipment ($l_\top$)
   LOAD-TRUCK lab-truck scientist ($l_\top$)
2. LOAD-TRUCK v-truck personal-possessions ($l_\top$)
   LOAD-TRUCK v-truck civilian1 ($l_\top$)
   LOAD-TRUCK v-truck civilian2 ($l_\top$)
   DRIVE-TRUCK lab-truck Science Lab, Safe Point ($l_\top$)
3. UNLOAD-TRUCK lab-truck scientific equipment ($l_\top$)
   UNLOAD-TRUCK lab-truck scientist ($l_\top$)
   DRIVE-TRUCK v-truck Volcano Village, Science lab ($l_\top$)
4. DRIVE-TRUCK v-truck Science Lab, Safe Point ($l_\top$)
5. UNLOAD-TRUCK v-truck civilian1 ($l_\top$)
   UNLOAD-TRUCK v-truck civilian2 ($l_\top$)
   UNLOAD-TRUCK v-truck personal possessions ($l_\top$)

A.3.2 Plan with Satisfaction Degree l_3

1. LOAD-TRUCK lab-truck scientist1 ($l_\top$)
   LOAD-TRUCK lab-truck scientific equipment ($l_\top$)
   FLY-HELICOPTER Far Village, Safe Point (l_3)
   DRIVE-TRUCK v-truck Volcano Beach2, Volcano Village ($l_\top$)
2. LOAD-TRUCK v-truck personal possessions ($l_\top$)
   LOAD-TRUCK v-truck civilian1 ($l_\top$)
   LOAD-TRUCK v-truck civilian2 ($l_\top$)
   DRIVE-TRUCK lab-truck Science Lab, Volcano Beach 2 ($l_\top$)
3. SAIL-BOAT Neighbour Beach, Volcano Beach2 (l_3)
   DRIVE-TRUCK v-truck Volcano-Village, Science Lab ($l_\top$)
   UNLOAD-TRUCK lab-truck scientist1 ($l_\top$)
   UNLOAD-TRUCK lab-truck scientific equipment ($l_\top$)
4. LOAD-BOAT scientist1 ($l_\top$)
   LOAD-BOAT scientific equipment ($l_\top$)
   DRIVE-TRUCK v-truck Science Lab, Safe Point ($l_\top$)
5. UNLOAD-TRUCK v-truck civilian1 ($l_\top$)
   UNLOAD-TRUCK v-truck civilian2 ($l_\top$)
   UNLOAD-TRUCK v-truck personal possessions ($l_\top$)
   SAIL-BOAT Volcano Beach2, Neighbour Beach (l_3)
6. LOAD HELICOPTER civilian1, civilian2 (l_3)
   UNLOAD-BOAT scientist1 ($l_\top$)
   UNLOAD-BOAT scientific equipment ($l_\top$)
   DRIVE-TRUCK n-truck Neighbour Village, Neighbour Beach ($l_\top$)

7. LOAD-TRUCK n-truck scientist1 ($l_\top$)
   LOAD-TRUCK n-truck scientific equipment ($l_\top$)
   FLY-HELICOPTER Safe Point, Far Village ($l_\top$)

8. UNLOAD-HELICOPTER civilian1, civilian2 ($l_\top$)
   DRIVE-TRUCK n-truck Neighbour Beach, Neighbour Village($l_\top$)

9. UNLOAD-TRUCK n-truck scientist1 ($l_\top$)
   UNLOAD-TRUCK n-truck scientific equipment ($l_\top$)

A.3.3 Plan with Satisfaction Degree l_4

1. LOAD-TRUCK lab-truck scientist1 ($l_\top$)
   LOAD-TRUCK lab-truck scientific equipment ($l_\top$)
   DRIVE-TRUCK v-truck Volcano Beach2, Volcano Village ($l_\top$)

2. LOAD-TRUCK v-truck personal possessions ($l_\top$)
   LOAD-TRUCK v-truck civilian1 ($l_\top$)
   LOAD-TRUCK v-truck civilian2 ($l_\top$)
   DRIVE-TRUCK lab-truck Science Lab, Volcano Beach 2 ($l_\top$)

3. SAIL-BOAT Neighbour Beach, Volcano Beach1 ($l_\top$)
   DRIVE-TRUCK v-truck Volcano Village, Volcano Beach2 ($l_\top$)
   DRIVE-TRUCK v-truck Volcano Beach2, Volcano Beach1 ($l_\top$)

4. UNLOAD-TRUCK lab-truck scientist ($l_\top$)
   UNLOAD-TRUCK lab-truck scientific equipment ($l_\top$)
   DRIVE-TRUCK v-truck Volcano Beach2, Volcano Beach1 ($l_\top$)

5. LOAD-BOAT scientist ($l_\top$)
   LOAD-BOAT scientific equipment ($l_\top$)
   UNLOAD-TRUCK v-truck civilian1 ($l_\top$)
   UNLOAD-TRUCK v-truck civilian2 ($l_\top$)
   UNLOAD-TRUCK v-truck personal possessions ($l_\top$)

6. LOAD-BOAT civilian1 ($l_\top$)
   LOAD-BOAT civilian2 ($l_\top$)
   LOAD-BOAT personal possessions ($l_\top$)

7. SAIL-BOAT Volcano Beach1, Neighbour Beach ($l_\top$)
   DRIVE-TRUCK n-truck Neighbour Village, Neighbour Beach ($l_\top$)

8. UNLOAD-BOAT scientist ($l_\top$)
   UNLOAD-BOAT civilian1 ($l_\top$)
   UNLOAD-BOAT civilian2 ($l_\top$)
   UNLOAD-BOAT scientific equipment ($l_\top$)
   UNLOAD-BOAT personal possessions ($l_\top$)

9. LOAD-TRUCK n-truck scientist ($l_\top$)
   LOAD-TRUCK n-truck civilian1 ($l_\top$)
   LOAD-TRUCK n-truck civilian2 ($l_\top$)
   LOAD-TRUCK n-truck scientific equipment ($l_\top$)
   LOAD-TRUCK n-truck personal possessions ($l_\top$)

10. DRIVE-TRUCK n-truck Neighbour Beach, Neighbour Village ($l_\top$)

11. UNLOAD-TRUCK n-truck scientist ($l_\top$)
    UNLOAD-TRUCK n-truck civilian1 ($l_\top$)
    UNLOAD-TRUCK n-truck civilian2 ($l_\top$)
    UNLOAD-TRUCK n-truck scientific equipment ($l_\top$)
    UNLOAD-TRUCK n-truck personal possessions ($l_\top$)