Reformulation in Planning

Derek Long, Maria Fox and Muna Hamdi

Department of Computer Science
University of Durham, UK
d.p.long@dur.ac.uk maria.fox@dur.ac.uk

Abstract. Reformulation of a problem is intended to make the problem more amenable to efficient solution. This is equally true in the special case of reformulating a planning problem. This paper considers various ways in which reformulation can be exploited in planning.

1 Introduction

The reformulation of a problem is intended to make the problem more amenable to efficient solution. While problems can usually be expressed in many ways, it is often the case that a particular problem-solving strategy is applicable only to problems expressed in a certain form. In this case, reformulation is the means by which a useful strategy can be brought to bear on a problem — the problem is reformulated in the canonical form in which that the particular strategy can be applied. Similar benefits can be obtained when the original problem expression is already in a form that can be tackled by a strategy, but reformulation can allow the strategy to be applied more effectively. Both of these situations can be seen as cases within the scenario depicted in figure 1. The figure illustrates how reformulation of a problem can allow different elements (or strategies) within a problem-solving system to be brought to bear on a problem by reformulating it. The figure illustrates that reformulation of a problem can allow different problem-solving strategies to be applied to it. A special case is where reformulation can allow the same strategy to be applied but in a more efficient way. The figure also suggests an important point: a single problem might require multiple strategies to solve it and reformulation might change the combination of strategies that can be applied to solving the problem.

In this paper we consider the role of reformulation in planning, examining several of the ways in which it has been exploited. We then turn to an important use of reformulation based on our notion of *generic types* and discuss how this approach can be used to improve planner performance. We also consider how generic types can be used to support alternative approaches to reformulation and, finally, discuss how the strategy that we have identified, for applying reformulation to planning, might generalise to other problems.

2 Reformulation in Planning

Planning problems are no exception to the possibility of benefits from reformulation. If we consider the problem-solving system in figure 1 to be a loose collection of strategies that might include constraint satisfaction strategies, general planning strategies,

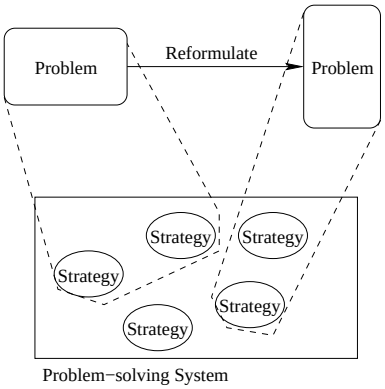


Fig. 1. Reformulation and redeployment: reformulating a problem can allow different problem-solving strategies to be deployed to solve it.

SAT-solvers and other, more specialised, problem solvers, then there have already been several efforts at reformulation of planning problems described in the literature. For example, Kautz and Selman have considered reformulating planning problems as SAT-problems in order to apply SAT-solving strategies to them [25]. Van Beek showed how planning problems can be reformulated as finite-domain constraint satisfaction problems and a CSP solver applied to solve them [39]. Where van Beek reformulated the problems by hand, Kambhampati and Binh Do have shown that automatic reformulation to CSPs is possible [10], allowing automatic redeployment of a planning problem from a generic planning strategy (such as Graphplan [4] or FF [24]) to a CSP-solver. Again, Cimatti, Roveri and Traverso have shown that planning problems can be reformulated as model-checking problems and an OBDD strategy applied to them [6]. All of these pieces of work have relied on a complete reformulation of planning problems from a classical action-based planning domain representation into the forms appropriate for each of the respective general problem-solving strategies (SAT-solving, CSP-solving or model-checking). This can be seen as the replacement of one "pure" problem-solving strategy with another.

The benefits of such a wholesale replacement of one generic problem formulation into another are sometimes rather ambiguous: the objective is often to take advantage of a well-developed strategy, perhaps one having made recent gains in performance. For example, the early work on reformulation to SAT-problems took advantage of then recent advances in SAT-solving technology to seek performance gains in planning. However, planning by SAT-solving has not demonstrated a convincing long-term benefit (no SAT-solver planning technology has demonstrated challenging performance compared with the current leading generic purpose-built planning technology such as FF [24], HSP2 [5] or the more recent LPG [23]). Planning by model-checking has also not shown convincing performance benefits and the significant number of model-checking approaches participating in the 2nd International Planning Competition in 2000 has dropped to there being no examples of pure model-checking in the 3rd (and most recent) competition although MIPS [12] adopts a hybrid approach which includes a

model-checking component. CSP-solving has been rather more successful as a strategy for planning by complete reformulation of planning problems, with Sapa [9] showing some promising performance in the 3rd IPC.

In the context of the competitions, at least, application of reformulation is constrained to be via fully automatic reformulation, unless one considers the "hand-coding" planners (such as TLPlan [3], TALPlanner [27] and SHOP2 [34]), for which domains are recoded by hand, to be exploiting reformulation. This would be somewhat misleading, since the recoding of the domains for these planners is not simply to reshape the domain, but to allow control rules to be added to the encoding that control the search in the planning systems. Although one can consider the addition of new information as part of the process of reformulation, it is clearly a far more expensive and knowledge-intensive task if this is intended and the subsequent benefits in problem-solving performance are therefore bought at a significant price in the reformulation efforts. Considered as a complete approach to solving planning problems there remains controversy in the community about the extent to which planning by "reformulating" through the addition of knowledge-intensive control information represents a generally acceptable tradeoff between reformulation effort and performance gains.

2.1 Reformulation and Expressiveness

One benefit that reformulation can offer is the ability to tackle problems expressed in an enriched formalism, where the formalism can express problems that extend beyond the capabilities of existing solving strategies. For example, Gazen and Knoblock [21] collected, completed and formalised common techniques for conversion of various elements of the expressive power of the ADL [36] extensions of planning domain descriptions into the simpler STRIPS subset. This reformulation allows the simpler STRIPS planning strategies to tackle problems expressed in the richer language. The greater expressive power of the ADL extension can be seen in the cost of the reformulation, which can be exponential in the size of the problem encoding in certain cases. Thus, the extra expressive power allows an exponential compression relative to a STRIPS encoding and can therefore lead to exponentially worse performance from a STRIPS planning strategy that the size of the original ADL encoding might suggest should be expected. A more practical reformulation is applied in IPP [26] to handle ADL expressive power more effectively. In that system the use of conditional effects, responsible for the most common exponential blow-up in the encodings of problems in the scheme proposed by Gazen and Knoblock, is handled by an extension to a STRIPS planning strategy, leaving the remaining elements of ADL to be reformulated into simpler forms. The gain is that the treatment of conditional effects in IPP can often be managed efficiently, while the Gazen and Knoblock approach will always cause combinatorial growth in domain encodings. A similar approach is taken in SGP [2] and in the more recent FF. Nebel has shown how the relative expressive power of these language extensions can be compared more formally, precisely by a reformulation technique, in [35]. In that work, Nebel considers the effect of reformulation of the original problem in such a way that a solution to the reformulated problem *allows recovery* of a solution to the original problem. This is, in fact, a common technique when applying reformulation approaches to problem-solving: a problem is reformulated and solved, if all goes as expected, more

efficiently, and then the solution to the original problem is extracted from the solution to the reformulated problem.

Another example of the same approach is in Fox and Long's work on temporal planning in the system LPGP [17]. In that system, planning problems in which there are durative actions, which are actions with duration that can have conditions and effects attached to both their start and end points and invariant conditions attached to the duration of their activity, are reformulated as collections of simple non-durative actions. These actions, together with some important linking constraints, allow a relatively straightforward extension of a non-temporal Graphplan planning strategy to handle much more expressive temporal planning problems.

A reformulation of this kind could, in principle, also be used to tackle the use of numeric valued expressions within planning domains. Since a finite plan can only ever introduce finitely many new numeric values, which it is possible to identify by a finite reachability analysis (such as a plan-graph construction), a propositional planning strategy can be used to tackle problems involving numbers using a continual reformulation as the strategy considers longer and longer possible solutions in its search for a plan. Whether such a strategy could be useful in practice would depend on the number of numeric values introduced during reachability analysis.

Occasionally reformulation can be used to to exploit a more powerful solver more effectively, where a problem has been expressed using a simpler expressive power that the solver can exploit. A simple example is that in which the domain analysis system, TIM [13], can be used to infer types in an untyped planning domain description, enriching the domain description and allowing a system that can exploit type information to improve its performance accordingly. Another example is the use of TIM to identify symmetries in a planning domain [14, 16], reformulating a problem in order to allow exploitation of symmetry elimination in the planning machinery. TIM and another fully automatic system, DISCOPLAN [22], both support reformulation of planning problems by the addition of mutex information and other constraints that can be exploited by planning systems.

3 Reformulation of Planning Problems for Deployment of Multiple Problem-Solving Strategies

The majority of the work described so far is directed at the reformulation of a planning problem into a form that can be tackled by a complete problem-solving strategy. Although this is obviously an important role for reformulation, it is also possible, as figure 1 suggests, to reformulate a problem in order to exploit multiple problem-solving strategies. An hypothesis that has driven an important line of research is that generic problem-solving strategies are unlikely to be the most efficient tools with which to tackle specific sub-problems that commonly arise as a part of larger problems. For example, the general search approaches that underlie many planning and CSP solving systems are not ideally suited to tackling problems that involve finding efficient routes for agents moving around while executing a plan. Much more effective is a tool that can exploit the fact that the problem involves finding shortest paths, possibly including

paths that visit specific locations in sequence, or as an unordered collection, and can use a problem-solving strategy that is tailored to that problem.

Several researchers have identified the benefits of specialised treatments of problems that arise naturally as an element of planning problems, particularly scheduling and resource handling [11, 28, 38, 15]. In almost all of these systems the planning problem is reformulated by hand in order to allow the planner to identify the separation of the sub-problem or sub-problems from the remainder of the planning problem. The whole problem can the be deployed across multiple solvers, including, possibly, a generic planning strategy to be applied to the core of the planning problem left once the sub-problems have been separated. The separation of a *hard* problem (NP-hard or worse) from a planning problem offers far more hope for handling it effectively than attempting to use general planning technology. It is unlikely in the extreme that a general planning strategy can prove a powerful heuristic approach to managing, for example, combinatorial resource management problems, Travelling-Salesman-variant route planning problems, Job-Shop-Scheduling-variant resource allocation problems and so on. In IxTeT [28] resources are handled by reformulating planning problems to make explicit the resource-producing and resource-consuming actions and then by using a resource-constraint manager to handle the constraints on the resources used within developing solutions. Similarly, resource profiling is used in OPlan [11]. In RealPlan [38] the problem of scheduling transporters to cargoes is handled by a separate scheduler. In all of these systems the communication between sub-solvers (resource manager, profiler or scheduler) and the rest of the planning system is a sophisticated technical problem.

In [15] Fox and Long describe Hybrid STAN, a planning system in which a special purpose strategy for addressing route-planning sub-problems is integrated with a general planning system. This system is based on *automatic reformulation* of a planning problem from a standard action-based formulation into one in which the sub-problems are identified and linked to the remainder of the problem using specialised expressions associated with actions that rely on conditions established within the fragment of the planning problem that is identified as a sub-problem. In [15] the need for a more general form of interface between the planner and its sub-solvers is indicated, allowing the communication of constraints between all the processes participating in the solution of a diversely structured problem. In this paper we outline some of the progress we have made towards the development of such an interface. We describe the notion of *active precondition* — a general means by which information can be communicated between a planner and one or more sub-solvers. Active preconditions represent an expressive form that can be exploited in the reformulation of a planning problem from a pure action-based model into one that makes explicit the relationships between those actions and sub-problems within a planning problem. We proceed to discuss this process of reformulation. We then demonstrate the use of active preconditions in the integration of STAN with two specialised solvers: one for planning the routes to be followed by mobile objects committed to visiting various locations in a plan, and one for allocating drivers to these mobiles. The domains we use for demonstrating the power of our integrated system feature mobiles that must be driven (in contrast to mobiles that are self-propelled, such as those in the Logistics domain).

3.1 Automatic Reformulation of Planning Problems through Generic Types

The automatic reformulation of planning problems rests on a well-established line of research that developed from the TIM system [13]. The development introduced the notion of a *generic type* [29] (the developments and the relationship to the original system can be seen in more detail in [32]).

A type, in a planning domain, is a set of objects that can all be used to instantiate the same subset of arguments of action schemas within the domain (although not all of the instantiated actions will necessarily be applicable, because of unsatisfied preconditions). Thus, types in planning domains are actually based on a functional commonality between objects within a single domain. In contrast, generic types are collections of types, characterised by specific kinds of behaviours, examples of which appear in many different planning domains. Thus, generic types are defined *across* domains, rather than within single domains. For example, domains often feature *transportation* behaviours since they often involve the movement of self-propelled or portable objects between locations. In the context of recognising transportation domains TIM can identify *mobile* objects, even when they occur implicitly, the operations by which they move and the maps of connected locations on which they move, the *drivers* (if appropriate) on which their mobility depends, any objects they can carry, together with their associated *loading* and *unloading* operations. The analysis automatically determines whether the maps on which the mobiles move are static (for example, road networks) or dynamic (for example, corridors with lockable doors). The recognition of transportation features within a domain suggests the likelihood of route-planning sub-problems arising in planning problems within the domain.

In addition to mobility-related generic types, other generic types have been identified and characterised. A generic type has been identified for *construction* behaviours [7]. Construction problems are commonly associated with iterative behaviour, suggesting the presence in the domain of types with inductive structure (analogous to lists and trees) associated with well-defined inductive operations. Recognition of these features supports an abstract level of reasoning about the domain. Generic types representing certain kinds of *resources* which restrict the use of particular actions in a domain have also been characterised [30]. The presence of these features suggest that processor and resource allocation sub-problems might arise and might be related to combinatorial sub-problems such as Multi-processor Scheduling or Bin Packing. TIM is able to recognise the existence, in a domain, of finite renewable resources which can be consumed and released in units [31].

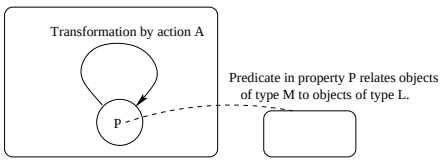


Fig. 2. A simple generic type fingerprint: the mobile type.

The analysis performed by TIM takes as input a standard STRIPS or, following recent extensions [8], an ADL description of a domain and problem. Some experiments have also been conducted with domains using numbers [18]. It should be emphasised that the analysis is automatic: no annotations are required to identify special behaviours and the analysis is completely independent of the syntactic labels used to describe the objects and operations. As a consequence, the analysis can recognise generic behaviours in domains which do not obviously fall into the categories indicated by the names of the generic types. This allows automatic reformulation of problems where a human domain-engineer might not identify the possibility. Recognition is based on the discovery of patterns within the structure of a domain encoding that can be described as *fingerprints* denoting the occurrence of specific behaviours amongst objects within the domain. The fingerprints are described in terms of relationships between finite-state machines that are extracted by TIM from a domain description and which show how objects in a domain make state transitions as actions are applied to them. For example, figure 2 represents the simplest pattern, indicating the existence of mobile objects. Figure 3 is an example of a more sophisticated pattern, this one identifying the existence of objects that display a particular constrained resource behaviour corresponding to a Multi-Processor Scheduling problem. In the figure the type P is the processor type and a processor object must be allocated to a task (type T) instance before that task can execute (moving from stage to stage). An example of a domain encoding that displays this behaviour (in its most explicit form) is given in figure 4.

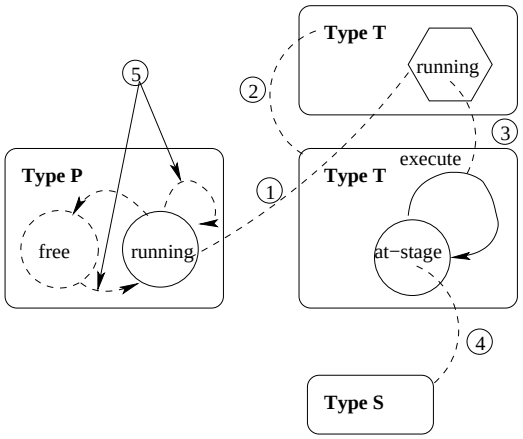
Although the recognition of different generic types is still carried out using *ad hoc* analysis techniques, based around the core TIM analysis, work is in progress towards the unification of these techniques based on a single matching strategy and a common framework for the description of generic types and relationships between them.

Once generic types have been identified in a domain, the planning problem can be reformulated in terms of sub-problems. A *sub-problem* is defined to contain the following components:

- A collection of objects capable of a specific behaviour (eg: trucks are capable of mobility);
- A collection of predicates capturing this behaviour (eg: *at* captures locatedness, *in* captures portability, *free* captures ability to be allocated to a task, etc).
- A collection of operators that affect these predicates (eg: *drive* affects *at*, *load* affects *in*, *allocate* affects *free*, etc).

TIM identifies the relevant objects, predicates and operators in a given domain. For example, figure 5 shows a particular encoding of the Multi-Processor Scheduling sub-problem, associated with the generic type of *processable task* (type T in figure 3). In this example there are three different types of objects participating in the sub-problem, and predicates associated with each of the three types. These predicates are the critical ones for solving the MPS problem. The collection of operators contains operators responsible for changing the states of the objects with respect to these key predicates.

The reformulation of the planning problem links sub-problem structures to actions by a process of abstraction. Abstraction of a recognised sub-problem from the planning domain involves the removal from the domain description of the sub-problem operators and the modification of all remaining operators to remove reference to the sub-problem



1: A type, P, contains a state defined by a predicate that links to a single instance of a type, T. The T instance acquires the corresponding property as an attribute.
2: The type, T, has a second property space, defining state−transition behaviour.
3: The attribute of T is an enabling condition for the transition, execute, in the second space for T.
4: The state for T is a property that links T to instances of another type, S.
5: The elements of P can enter the state in which they are related to elements of T by either a simple loop (reallocate) or by a transition (release) to a second state (free) and a transition back (allocate).

Fig. 3. The fingerprint of the generic type of driver objects.

```
(:action execute
        :parameters (?p ?j ?s ?t)
        :precondition (and (running ?p ?j) (stage ?j ?s) (nextto ?s ?t))
        :effect (and (stage ?j ?t) (not (stage ?j ?s))))

(:action swap-to
        :parameters (?p ?j)
        :precondition (and (idle ?p) (job ?j))
        :effect (and (running ?p ?j) (not (idle ?p))))

(:action swap-from
        :parameters (?p ?j)
        :precondition (and (running ?p ?j))
        :effect (and (idle ?p) (not (running ?p ?j))))
```

Fig. 4. Canonical Multi-Processor Scheduling as a Planning Domain.

MPS Sub−problem

Objects:	*tasks, stages, processors*
Predicates:	*free, allocated, priocessed*
Operators:	*allocate, de−allocate, re−allocate, process*

Fig. 5. An encoding of the MPS Sub-problem

predicates. In this way, all references to the sub-problem are removed from the planning domain description, leaving a *core* problem for the planner to solve. In [15] is described a simple form of integration between a forward planner and a route-planning sub-solver. This integration is based on passing information from the planner to the sub-solver about the current and required locations of mobiles that are required to move. The interface mediating the interaction between the planner and sub-solvers described in that work has now been generalised into a more powerful form of interface called an *active precondition*.

We begin by defining the structure of an *object specification*:

Definition 1 *An* Object Specification, OS_t, *associated with an object in a planning problem,* t*, is a triple containing the following components:*

1. *The current state of* t*. This is initialised from the initial state and is updated every time* t *changes its state;*
2. *The final state of* t*, taken from the goal specification. If* t *has no state specified in the goal destination this field is null.*
3. *The generic structure of* t*. This defines what kinds of generic behaviours* t *supports.*

The state of an object is the collection of properties that refer to that object in the current state of the world. Typically, the state of an object can be partitioned into collections of propositions that identify the state of the object relative to one or other of the generic behaviours the object supports. For example, a mobile object will support movement between locations and the current location is its state relative to this behaviour.

Definition 2 *An* Active Precondition, AP_o*, with respect to a set of objects* os*, is a triple containing the following components:*

1. *A proposition,* P*, involving* os*;*
2. *The object specifications, for each* $o \in os$*,* OS_o*;*
3. *The identity of the sub-solver that is responsible for satisfying* P*.*

The active precondition is a data structure that couples the original precondition of an action to the objects that are involved in the proposition and the sub-solver that can bring about the necessary condition. Often the objects that appear in a proposition are not all equally significant. For example, if the proposition is that a certain task must reach a particular stage then the task is the more important object since it is the task that must be processed in order to reach a particular stage. The precise significance of each object in an active precondition depends on the way that the sub-solver manages the process of achieving the corresponding proposition.

If we consider a planning problem that includes an instance of the generic type of processable tasks, then we can see that the object specification for a task will include the information indicating what stage the task is currently at, what stage it is to reach and whether it currently has a processor allocated to it or not. An object specification is a data structure that encapsulates the representation of the state associated with the object. The advantage of this is that, having identified a particular instance of a generic type to which the object belongs, we can select a representation for the object specification

that makes it more efficient to access the corresponding properties of the object. In the case of the task, we can store its current stage, its goal stage and whether or not there is a processor allocated to it in an internal form that is much more easily modified as the task is processed than a simple set of propositions would be.

Figure 6 shows how this abstraction is achieved with a task argument to a packing action. It can be seen that each action that has a precondition that is to be solved by a sub-solver will acquire a collection of active preconditions to replace them, along with the remaining standard preconditions. Each active precondition is responsible for handling only one proposition, but an action can have many active preconditions.

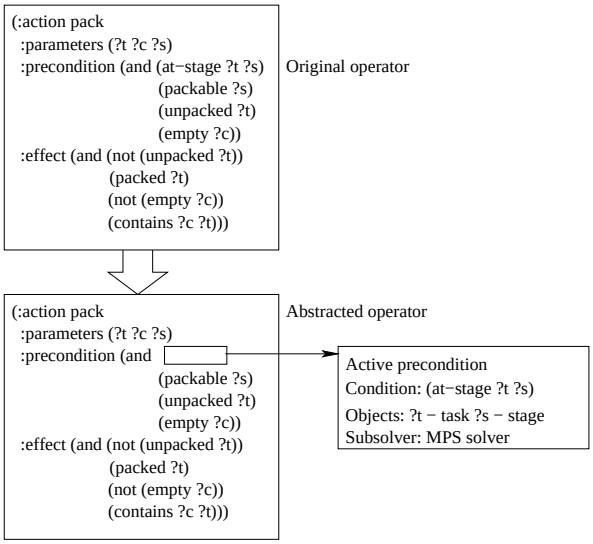


Fig. 6. The abstraction process for a packing operator. The *at-stage* precondition for the *task* t is replaced with an active precondition.

In addition to replacing the preconditions with active preconditions in this way, the abstraction process will remove the action responsible for the generic movement behaviour associated with the mobile type. In certain cases the process of asbtraction can leave a null domain description (this happens in the canonical MPS problem encoded as a planning problem [30], where there is no other structure than that of the MPS subproblem), but in general there remain components of the original problem that have to be solved by the planner. Active preconditions form one component of the mechanisms by which sub-solvers communicate. The way in which they are currently exploited is described in the next section.

3.2 Exploitation of Generic Types

In order to exploit the reformulation, our planning system is driven by a forward planning system that attempts to solve the remaining planning problem (if there is one),

using a relaxed-plan heuristic in much the same way as FF [24]. As relaxed plans are constructed in order to evaluate alternative state transitions these relaxed plans create agendas, formed from the active preconditions of actions selected in the relaxed plan. The agendas are then examined by sub-solvers to identify the goals that have been posed that fall under the remit of each of the sub-solvers. The sub-solvers then communicate to the forward planner an estimate of the cost of achieving the corresponding goals within the sub-problem for which they are responsible. Using this enriched goal distance estimate the forward planner selects the best transition by which to progress.

If the action selected contains an active precondition then the corresponding sub-solver is given a new task: to construct an actual plan fragment which will achieve the necessary precondition from the current state prior to insertion of the newly selected action. The abstraction process ensures that all of the materials required by the sub-solver in order to achieve this are under the control of the sub-solver, so that the goal can be achieved. This process ensures efficient solution of the sub-problem and can utilise global state information about the progress of the solution in order to ensure that resources are sensibly deployed — for example, the MPS solver can ensure that loads are balanced between processors in order to achieve an efficient solution.

There are complications in this linkage that we have not space to address properly in this paper. An initial report of some of the progress we have made in handling the difficulties that arise when sub-problems are interdependent can be found in [19]. In particular, we have identified the relationships that can exist between sub-solvers and their respective responsibilities and can automatically build a dependency network that allows these relationships to be identified. Armed with this information it is possible to decompose the agenda constructed by a planner so that sub-solvers examine it in an order that allows sub-solvers to impose further goals for subsequent sub-solvers to achieve. There remain several significant challenges to resolve in order to make this approach fully general.

To give a broad idea of what can be achieved using the reformulation into sub-problems we present one data set. Data demonstrating performance in other problem sets can be found in, for example, [15, 7, 19]. The current data set shows performance on a collection of randomly generated MPS problems.

In this test we compared IPP [26], BlackBox [25], FF [24] and STAN on CMPS instances involving increasing numbers of processors and tasks. We were interested in comparing both time taken to solve the instances and makespan of the solutions. All experiments were performed on a Celeron 333 Intel processor and a machine with 256Kb of RAM, 256Kb swap space. The larger problems defeated FF primarily in the instantiation phase. It should be noted that the reformulation being carried out by TIM in these problems includes the addition of type information which is only impicit in the encodings. The instantiation by FF is improved if type information is explict, but even with this information FF cannot solve the largest half-dozen problems on this machine. The quality of the plans produced by FF is consistently poor, with all tasks being allocated to a single processor until late in the sequence when one or two alternative processors are sometimes used. This is only to be expected, since FF generates plans that IPP and BlackBox both found the optimal parallel plan for the smallest instance but had insufficient memory to solve any further instances. BlackBox attempts first to

Procs	Tasks	IPP		Blackbox		FF		STAN4		Diff	Sub
		Secs	Span	Secs	Span	Secs	Span	Secs	Span		Opt?
2	5	3.28	11	12.72	11	.01	19	.006	11	0	
2	10					.1	72	.008	38	1	
2	20					.87	233	.066	118	0	
3	10					1.49	64	.088	25	1	
3	20					1.83	211	.064	79	1	
3	30					8.92	477	.073	170	1	
4	20					53.07	215	.017	59	1	
4	30							.09	128	2	1
4	40							.121	190	2	
4	50							.141	358	3	1
10	50							.206	145	10	2
10	80							.560	300	3	
10	100							.954	548	2	
15	50							.132	97	6	2
15	100							.98	366	4	
15	150							2.48	806	4	
20	50							.153	72	3	
20	80							.504	150	10	
20	100							.894	275	5	
20	150							2.498	606	3	

Fig. 7. Results for MPS domain instances. Instances were randomly generated.

use a GraphPlan strategy and only switches to SAT-solving if this has failed to make progress when a fixed time cut-off is reached. The first instance was small enough to be solved before the cut-off so BlackBox used its Graphplan strategy in this case.

STAN is able to solve all of the instances without any search at all. When TIM has identified the MPS nature of the problem a heuristic strategy is invoked to solve the sub-problem. The first-fit decreasing heuristic produces very good quality solutions. The Diff column in the table shows the difference in load between the lightest and heaviest loaded processors. As can be seen, the difference is generally very small indicating that all processors are being allocated an approximately average load. The final column shows the extent to which STAN's solutions seem likely to deviate from optimal. By inspection of the proposed allocation it can be determined whether it is, in principle, possible to rearrange the loads to result in a shortened makespan. Whether this is actually possible or not cannot easily be determined (we would need to search for an alternative solution or compare our solutions with ones generated by alternative heuristics or approximation schemes). Only four of the solutions were possibly non-optimal and these only by very small margins.

4 Generic Types and Reformulation

So far we have discussed the use of generic types in their role as a foundation for automatic reformulation of planning problems, based on automatic recognition of fin-

gerprints within action-based models. A more recent direction of work is exploring the use of generic types as *planning domain design patterns*, analogous to the software engineering notion of *design pattern* [20]. Defining a design pattern in his seminal work, Alexander, credited with invention of design patterns, states [1] that:

> Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem in such a way that you can use this solution a million times over without ever doing it the same way twice. *Christopher Alexander*

Generic types capture precisely this generality and the fingerprints define, in a planning-domain pattern language, the necessary components and relationships between them. The characterisations provided in figures 2 and 3 have a similar status to the design patterns of programs and can play a similar role: to support the construction of planning domain descriptions using well-understood structural components that capture common behaviours. Seen in this light, the automatic recognition of generic types is an attempt to recognise and capture design patterns in use within a planning domain. The exploitation of generic types as planning domain design patterns is at an early stage and we perceive there to be opportunities in domain engineering as well as in supporting efficient planning. Some early work illustrating the use of generic types in a domain engineering role can be found in [37].

The construction of planning domains in terms of generic types *ab initio* offers some important opportunities for reformulation. All of the techniques for abstraction of sub-problems can, of course, be used to reformulate problems in the ways already described, but in addition it is possible to use the existence of generic types to reformulate problems in canonical structures, or to add control information that is associated with the existence of generic types within a problem allowing automatic reformulation for planning systems that currently rely on hand-coded control rules [33].

5 Conclusions

Reformulation has played and continues to play a vital role in planning. In this paper we have concentrated on a particular strategy for the application of reformulation, based on the observation that planning problems typically contain sub-problems that have been tackled as research problems in their own right and for which efficient heuristic solvers have been developed that will inevitably out-perform generic planning technology. This approach is still at an early stage of development, but we have demonstrated that it is both possible and effective.

Planning is not alone in being faced with a wide variety of problems that are often composed of combinations of structured sub-problems. Reformulation in order to redeploy problem solving across multiple sub-solvers, each specialising in the solution of one kind of sub-problem, would appear to be a strategy that can be applied across a much broader spectrum of combinatorial problem solving. Indeed, it seems likely that even the notion of a generic type is more general than to be applicable to planning problems alone, and its application to other areas of automatic reasoning could be a fruitful way in which to extend the power of reformulation across the whole field.

References

1. C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. *A Pattern Language*. Oxford University Press, 1977.
2. C.R. Anderson, D.E. Smith, and D.S. Weld. Conditional effects in Graphplan. In *Proc. of 4th International Conference on AI Planning Systems*, 1998.
3. F. Bacchus and F. Kabanza. Using temporal logic to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.
4. A. Blum and M. Furst. Fast Planning through Plan-graph Analysis. In *Proc. of 14th Internation Joint Conference on AI*, pages 1636–1642. Morgan Kaufmann, 1995.
5. B. Bonet and H. Geffner. Planning as heuristic search: new results. In *Proc. of 4th European Conference on Planning (ECP)*. Springer-Verlag, 1997.
6. A. Cimatti, M. Roveri, and P. Trverso. Strong planning in non-deterministic domains via model-checking. In *Proc. of 4th International Conference on AI Planning and Scheduling (AIPS'00)*, 2000.
7. M. Clark. Construction domains: a generic type solved. In *Proceedings of 20th Workshop of UK Planning and Scheduling Special Interest Group*, 2001.
8. S. Cresswell, M. Fox, and D. Long. Extending TIM domain analysis to handle ADL constructs. In L. McCluskey, editor, *Knowledge Engineering Tools and Techniques for AI Planning: AIPS'02 Workshop*, 2002.
9. M.B. Do and S. Kambhampati. Sapa: a domain-independent heuristic metric temporal planner. In *Proc. ECP-01*, 2001.
10. Minh Binh Do and S. Kambhampati. Solving planning graph by compiling it into a CSP. In *Proc. of 5th Conference on AI Planning Systems*, pages 82–91. AAAI Press, 2000.
11. B. Drabble and A. Tate. The use of optimistic and pessimistic resource profiles to inform search in an activity based planner. In *Proc. of 2nd Conference on AI Planning Systems (AIPS)*. AAAI Press, 1994.
12. S. Edelkamp. Mixed propositional and numeric planning in the model checking integrated planning system. In M. Fox and A. Coddington, editors, *Planning for Temporal Domains: AIPS'02 Workshop*, 2002.
13. M. Fox and D. Long. The automatic inference of state invariants in TIM. *Journal of AI Research*, 9:367–421, 1998.
14. M. Fox and D. Long. The detection and exploitation of symmetry in planning problems. In *Proc. of 16th Internation Joint Conference on AI*, pages 956–961. Morgan Kaufmann, 1999.
15. M. Fox and D. Long. Hybrid STAN: Identifying and Managing Combinatorial Sub-problems in Planning. In *Proc. of 17th International Joint Conference on AI*, pages 445–452. Morgan Kaufmann, 2001.
16. M. Fox and D. Long. Extending the exploitation of symmetries in planning. In *Proc. of 6th International Conference on AI Planning Systems (AIPS'02)*. AAAI Press, 2002.
17. M. Fox and D. Long. Fast temporal planning in a Graphplan framework. In M. Fox and A. Coddington, editors, *Planning for Temporal Domains: AIPS'02 Workshop*, 2002.
18. M. Fox, D. Long, S. Bradley, and J. McKinna. Using model checking for pre-planning analysis. In *AAAI Spring Symposium Series: Model-based Validation of Intelligence*. AAAI Press, 2001.
19. M. Fox, D. Long, and M. Hamdi. Handling multiple sub-problems within a planning domain. In *Proc. of 20th Workshop of UK Planning and Scheduling Special Interest Group*, 2001.
20. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of reusable software*. Addison-Wesley, 1995.
21. B.C. Gazen and C. Knoblock. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In *Proc. of 4th European Conference on Planning (ECP'97)*, 1997.

22. A. Gerevini and L. Schubert. Accelerating Partial Order Planners: Some Techniques for Effective Search Control and Pruning. *Journal of AI Research*, 5:95–137, 1996.
23. A. Gerevini and I. Serina. LPG: A planner based on local search for planning graphs. In *Proc. of 6th International Conference on AI Planning Systems (AIPS'02)*. AAAI Press, 2002.
24. J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of AI Research*, 14:253–302, 2000.
25. H. Kautz and B. Selman. Unifying SAT-based and graph-based planning. In *Proc. of 14th Internation Joint Conference on AI*, pages 318–325. Morgan Kaufmann, 1995.
26. J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In *Proc. of 4th European Conference on Planning, Toulouse*, pages 273–285, 1997.
27. J. Kvarnstrom and P. Doherty. TALplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence*, 30(1-4):119–169, 2000.
28. P. Laborie and M. Ghallab. Planning with sharable resource constraints. In *Proc. of 14th International Joint Conference on AI*. Morgan Kaufmann, 1995.
29. D. Long and M. Fox. Automatic synthesis and use of generic types in planning. In *Proc. of 5th Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 196–205. AAAI Press, 2000.
30. D. Long and M. Fox. Multi-processor scheduling problems in planning. In *Proc. of ICAI'01, Las Vegas*, 2001.
31. D. Long, M. Fox, L. Sebastia, and A. Coddington. An examination of resources in planning. In *Proc. of 19th UK Planning and Scheduling Workshop, Milton Keynes*, 2000.
32. D. Long and M.Fox. Planning with generic types. Technical report, Invited talk at IJCAI'01 (forthcoming Morgan-Kaufmann publication), 2001.
33. L. Murray. Reuse of control knowledge in planning domains. In L. McCluskey, editor, *Knowledge Engineering Tools and Techniques for AI Planning: AIPS'02 Workshop*, 2002.
34. D. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. SHOP: Simple hierarchical orederd planner. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1999.
35. B. Nebel. On the compilability and expressive power of propositional planning formalisms. *Journal of AI Research*, 12:271–315, 2000.
36. E. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. of 1st International Conference on Principles of Knowledge Representation and Reasoning*, pages 324–332. San Francisco, CA, Morgan Kaufmann, 1989.
37. R. Simpson, L. McCluskey, D. Long, and M. Fox. Generic types as design patterns for planning domain specification. In L. McCluskey, editor, *Knowledge Engineering Tools and Techniques for AI Planning: AIPS'02 Workshop*, 2002.
38. B. Srivastava. RealPlan: Decoupling causal and resource reasoning in planning. In *Proc. of 17th National Conference on AI*, pages 812–818. AAAI/MIT Press, 2000.
39. P. van Beek and X. Chen. CPlan: A constraint programming approach to planning. In *Proc. of 16th National Conference on Artificial Intelligence*, pages 585–590. AAAI/MIT Press, 1999.