EFFICIENT PLANNING BY EFFECTIVE RESOURCE REASONING

by

Biplav Srivastava

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

ARIZONA STATE UNIVERSITY

May 2000

EFFICIENT PLANNING BY EFFECTIVE RESOURCE REASONING

by

Biplav Srivastava

has been approved

January 2000

APPROVED:

_____, Chair

_____

_____

_____

_____

Supervisory Committee

ACCEPTED:

_____

Department Chair

_____

Dean, Graduate College

ABSTRACT

Planning consists of an action selection phase where actions are selected and ordered to reach the desired goals, and a resource allocation phase where enough resources are assigned to ensure the successful execution of the chosen actions. In most real-world problems, these two phases are loosely coupled. Most existing planners do not exploit this loose-coupling and perform both action selection and resource assignment employing the same algorithm. The current work shows that the above strategy severely curtails the scale-up potential of existing planners, including such recent ones as Graphplan and Blackbox. In response, a novel planning framework was developed in which resource allocation is de-coupled from planning and is handled in a separate "scheduling" phase.

Implementing this framework raises several interesting issues regarding the role of resources in planning, the interactions between the planning and scheduling phases and the choices in selecting the methods for the two phases. During planning, resource constraints are ignored and an abstract plan is produced that can correctly achieve the goals but for the resource constraints. Next, based on the actual resource availability, the abstract plan is allocated resources to produce an executable plan. This work introduces a procedural method for inexpensive (bactrack-free) scheduling and a declarative method for posing the scheduling problem with all its complexity as a Constraint Satisfaction Problem.

The new approach not only preserves both the correctness as well as the quality (measured in length) of the plan but also improves efficiency. It is implemented on top

of Graphplan and shows impressive empirical results. This work can be viewed beyond the context of planner efficiency as how to integrate planning with real world problem solving. Specifically, when a plan fails to achieve its intended purpose during plan execution, it does not imply that the causal structure of the failed plan was incorrect but that some allocated resources were found to be unavailable. The benefit of the current approach is that it provides a framework to undertake only necessary resource re-allocation and not complete re-planning.

To my parents:

Dr. A. K. Srivastava and Smt. Shanti Srivastava

ACKNOWLEDGMENTS

I would like to thank my advisor, Subbarao Kambhampati, for giving me complete freedom in whatever I set out to do and for guiding me towards truly satisfying research and the completion of this dissertation. A big thanks also goes to my AI research team, Yochan, for very insightful discussions, support and pointers over the past 5 years. In particular, Laurie Ihrig, Terry Zimmerman, Amol Mali and BinhMinh Do have served as patient sounding boards.

On a personal level, I would like to thank my sister, Priyanka, and brother-in-law, Praveen, and last but not the least, to my wife, Vandana, for support and trying to understand planning from me.

TABLE OF CONTENTS

Page

LIST OF TABLES

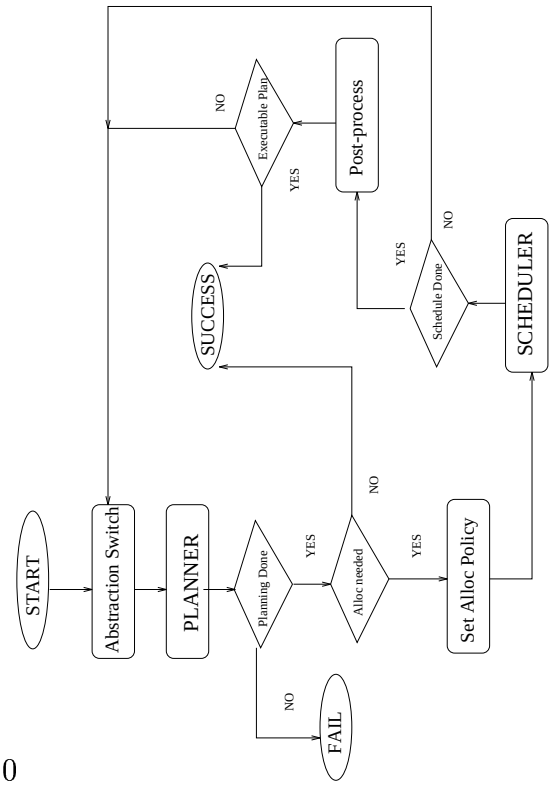
LIST OF FIGURES

Chapter 1

Introduction

Planning comprises of causal reasoning and resource reasoning. Given a domain, a set of actions to change states in the domain, an initial state and the desired goal state, the planning problem is to find a sequence of actions (also known as a plan) such that when it is executed from the initial state, a goal state can be reached. Causal reasoning ensures that for every action in the plan, its preconditions can be satisfied from the effect of another action preceding it within the plan. Causal relationships force sufficient orderings among actions to achieve the goals and furthermore, determine the extent of concurrency[1] possible in a plan. Resource reasoning ensures that all the resources needed for the execution of an action are available for allocation without any resource conflicts. A resource conflict occurs when two actions cannot be assigned the same resource, either due to resource characteristics (non-sharable resources) or due to domain characteristics (actions interfere). If resources are scarce,

[1] Borrowing from operating system terminology, concurrency refers to the potential of executing actions in parallel. The parallelism (or lack of it) exhibited in the final plan is dependent on the actual number of resources available to exploit this concurrency.



270

Figure 1.1. Unified planning-scheduling framework

the resource allocation may involve freeing and reallocating the limited resource which can add more ordering relationships among actions and effectively serialize the plan.

AI Planning can handle small plans compared to what humans can already control in the real world. In real-world problems, planning and scheduling phases are usually *loosely coupled*. Planning and monitoring of large-scale projects is done with a project management tool like Microsoft Project[27] by using a activity network for both planning and control. Humans come up with the Work Breakdown Structure (WBS)[26] to identify the different tasks at some granularity and estimate time and resources for each task. From this information, the critical bottleneck in the project is

identified and the sequence of non-critical tasks is re-alligned to optimize on resource usage. In Critical Path Method (CPM), activity times are assumed to be known or predictable (deterministic) while in Program Evaluation and Review Technique (PERT), activity times are assumed to be random, with known probability distribution (probabalistic). But the nature of WBS (i.e. causal plan) remains relevant irrespective of how activities (actions) are modeled.

Most planners do not distinguish between causal and resource reasoning and handle them within the same planning algorithm. Discrete resources (sharable or non-sharable) like robots, trucks and planes have traditionally been straightforwardly handled by logic-based planners like UCPOP[31] and Graphplan in the same way as other objects in the domain. I will show (in Section 1.1) that this strategy severely curtails the scale-up potential of existing planners, including such recent ones as Graphplan [2] and Blackbox [15]. In particular, these planners exhibit the seemingly irrational behavior of worsening in performance with increased resources. For continous resources like time and fuel, planning systems have additionally employed time/resource map managers to ensure resource consistency (SIPE[35], IxTeT[21], IPP[20], LPSAT[36]). But such an integration explodes the search space for the planner beyond action sets that are minimal with respect to the logical goals. Actions may be added to achieve the resource goals but may not be necessary for logical goals. To handle this, IPP restricts expressivity by avoiding explicit temporal modeling while other planners take a performance drop with slower flaw resolution.

In this dissertation [32],[33], I introduce a novel approach where causal-

reasoning (planning) is used to generate an abstract plan ignoring all resource conflicts. The abstract plan is then post-processed to allocate the required resources without altering the causal structure of the plan. Separating planning and scheduling is quite the normal practice in project planning scenarios in the industry, where planning is done by the humans, and scheduling is done by a variety of software packages. I am proposing a similar flow to exploit the loose coupling – except that both planning and scheduling phases will be automated.

Figure 1.1 provides a general overview of my approach. My unified framework accepts a domain description along with optional annotations for resources, finds a plan modulo the choice of resource abstraction, and then allocates resources to produce the correct final plan (if necessary). In this work, I focus on discrete reusable as well as non-sharable resources. But I argue in Chapter 8 that the approach can be extended for continous resources as well. Resources are either declared by the domain expert, or deduced through automated methods discussed later. After planning is complete, a scheduler can decide which resources to actually allocate. I have implemented this approach on top of Graphplan algorithm, and the resulting planner is not only more rational in its treatment of resources (i.e., performance does not worsen with increased resources), but also significantly outperforms Graphplan on several benchmark problems.

There are a number of technical challenges that arise in making my approach work. First, I have to identify resources in a given domain. Second, I have to decide about optimization criteria during scheduling. Third, I need to allocate resources to

an abstract plan without transferring the full complexity of planning to the scheduling phase. The planning phase produces an abstract plan of shortest length in terms of number of steps (where each step may contain several concurrent actions)[2]. The resource allocation problem is formulated as a Constraint Satisfaction Problem (CSP) and considered for scheduling based on different allocation policies including maintaining the concurrency of the plan, serializing the plan and inserting actions to free and reallocate the resources. If freeing/ reallocating actions are allowed, the problem is infact a Dynamic Constraint Satisfaction Problem (DCSP) because these actions (variables) control the normal action variables.

I introduce an intuition driven procedural method for inexpensive (bactrack-free) scheduling. My aim in procedural scheduling phase is to use the least number of resources for producing a final plan of the same length as the abstract plan and with minimum number of additional actions and causal relationships, without changing the relative positions of actions. When resource allocation is not possible without increasing the plan length, I consider it as a hard resource allocation problem and revert to declarative scheduling or traditional planning. I also introduce a declarative approach for formally specifying the resource allocation constraints and transferring the full complexity to a constraint solver. In my declarative approach, all the constraints of the scheduling problem are formulated as a Dynamic Constraint Satisfaction Problem (DCSP) and passed to a CSP solver (specifically, backjumping solver in [34]). If the declarative scheduling method fails to allocate resources in the context of given

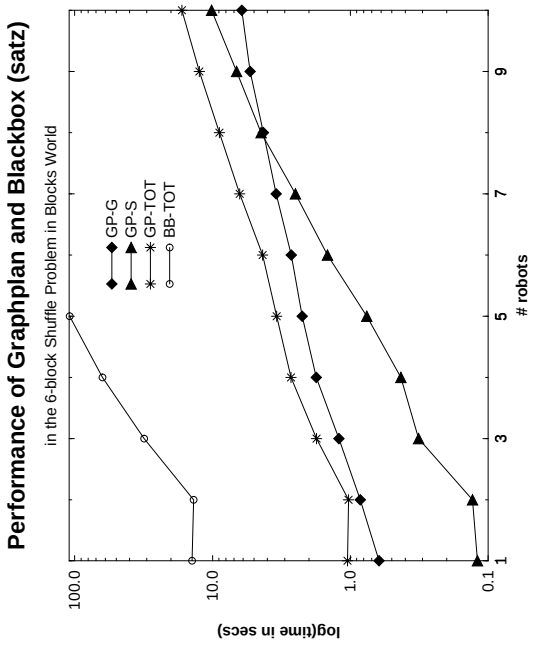
[2]Such a plan may not be optimal if actions have differing costs, but this is how Graphplan works.

resources, time limit and nature of allocation policy, the responsibility transfers to the planner to change any of the permissible parameters and try again. If the resource allocation succeeds and new free/ reallocation actions were added by the scheduler, the scheduled plan is post-processed for neccessary domain translation for executability. If all the allocation policies lead to failure or inexecutable plans in a domain, this implies that planning and scheduling were infact, *not loosely coupled*. In such a case, the framework retains the ability to switch off resource abstraction and resort to traditional planning.

1.1 An Empirical Motivation

Since the current investigation is primarily in the context of Graphplan, a state-of-the-art planner, it will help to give a brief summary. Graphplan [2] performs forward state-space refinement over disjunctive partial plans [13] that correspond to a unioned representation of the forward state-space search space tree. To improve pruning power in these disjunctive plans (*planning graphs* in Graphplan parlance), Graphplan infers and propogates information about disjuncts which cannot hold together in a solution (mutex relations). A solution is obtained by searching for a sequence of actions in the planning graph that satisfies the planning problem and mutexes help considerably in this effort. The terminology below is biased towards the Graphplan algorithm but the idea is general enough to work with most classical planners.

To motivate the need for separating resource scheduling, I will see the behavior



270

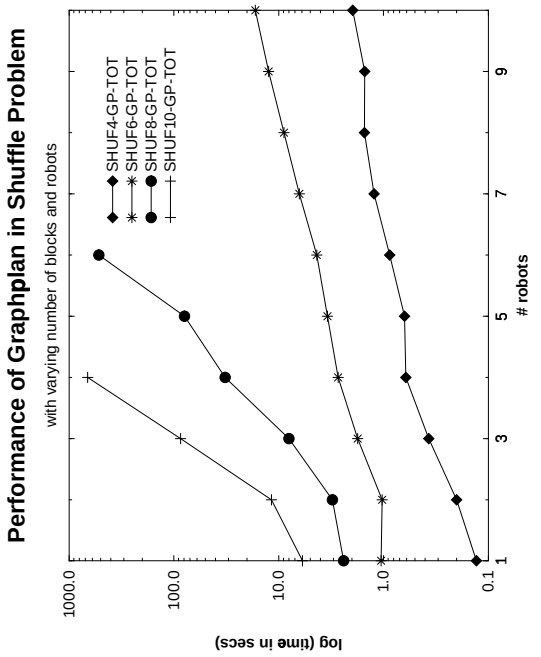
Figure 1.2. Performance of Graphplan and Blackbox(satz) on the 6-block *Shuffle* problem with varying number of robots. Performance degrades with increase in the number of resources.

of Graphplan, in a modified blocks world domain that contains multiple robot hands. If I run Graphplan multiple times on the same problem, while increasing the number of robot hands available, one would expect that the performance would improve with increased resources. Figure 1.2 shows the performance of Graphplan on the "*shuffle*"[3] problem, where a 6-block stack needs to be shuffled in a symmetric way to form a new stack. Notice that the total planning time, GP-TOT, *increases* quite steeply with the increase in the number of robots. In fact, by providing 8 robots instead of 1 robot, the planning time is slowed down by an order of magnitude! Lest the reader suspect that the increase is just due to the increased cost in constructing a planning-graph, the figure also plots the time for building the planning graph (GP-G) and the time for searching the planning graph (GP-S). I note that *both of them* increase with the number of robots.

I wanted to further check if the results are consistent when the problem size is scaled independent of number of resources. In Figure 1.3, I show the performance of Graphplan on *shuffle* problems of 4, 6, 8 and 10 blocks as the number of robots are varied from 1 to 10. I note that planning performance degrades with increase in size of the domain as well as resources.

The rather counter-intuitive behavior of the planning algorithm (in Figure 1.3) can be deciphered once we realize that *every causal failure is being needlessly rediscovered multiple times* with different identities of the robot hands. The plan length and the number of steps in the plan reduce with increased resources as more resource

[3]Shuffle problem is the multiple robots version of the 6-block *blocks_facts_shuffle* problem in the Graphplan system. Later I consider k-block *shuffle* versions also.



270

Figure 1.3. Comparative performance of Graphplan in *shuffle* problem of 4, 6, 8 and 10 blocks. Performance degrades with increase in size of the domain as well as resources.

conflicts get resolved at a level[4] (see Figure 1.4 which plots the number of steps and actions in the solution plan for *shuffle*) and stabilize after 1 and 4 robots respectively. More resources lead to the increase in the planning graph size and consequently the cost of plan search in its space. Specifically, the asymptotic cost of planning in Graphplan like planners is $O(w^l)$, where w is the width of the planning graph and l is the length of the graph. As the resources (e.g. number of robots in blocks world) increase, l tends to reduce while w increases. However, l does not reduce indefinitely, while w does increase monotonically with resource increase. Thus, the net effect is that the performance degrades with increased resources.

To ensure that this behavior is not peculiar to Graphplan, I also experimented with Blackbox [15], which uses SAT techniques for searching the plan graph; and UCPOP [31], a traditional partial-order planner[14, 23]. I found similar behavior in both cases. The plot titled BB-TOT in the left graph in Figure 1.2 illustrates the behavior of Blackbox. When the same *shuffle* blocks world problem was given to UCPOP, even the smallest problem instance with 6 blocks and 1 robot could not be solved in 10 minutes and search limit of 100000 nodes. With a smaller size problem like Sussman Anomaly which has 3 blocks (refer to Table 1.1), I found that the increase in robots does increase the planning time. The search space here gets large with increase in robots as all possible clobberers of conditions have to be considered for completeness.

[4]In Graphplan parlance, a plan step is also called an operator level. I will use the terms step and level interchangeably.

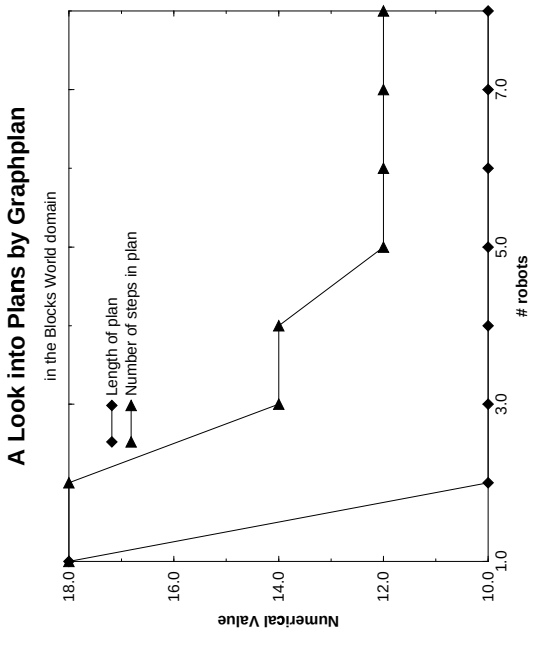


Figure 1.4. The nature of plans produced by Graphplan on a blocks world problem with varying number of robots. As more resources are added, the plan length and number of steps in the plan reduce because there are more possibilities of avoiding resource bottlenecks and so, more ways of generating parallel plan.

Robots	1	2	3	4	5
Time (in sec)	0.49	0.73	0.85	1.46	2.42

Table 1.1. Table showing the performance of UCPOP in Sussman Anomaly problem with varying number of robots.

1.2 Scheduling as a Post-planning Phase

As soon as Graphplan generates a plan graph upto the level that one solution can be extracted, it has a structure that contains all minimal solutions to the problem. Keeping disjunctive plans around leads to the explosion of the size of plan graph due to the recording of interactions among domain objects. Since the user does not care about the specifics of resources, recording interactions among multiple resources is clearly wasteful. This also degrades the backward search. One can try to reduce both graph expansion and search by abstracting (variablizing) resources needed by actions during planning and avoiding checking all interactions between them. The assumption is that they would be somehow allocated following search in a scheduling phase.

For UCPOP, considering resources during planning corresponds to additional clobbering possibilities which can degrade the planner if conflict resolution is enabled. Abstraction of resources leads to postponment of commitments on them during conflict resolution. We will focus on Graphplan planner for the rest of the dissertation but the techniques are still useful for other forms of AI planning.

When I abstract resources, I not only reduce the plan graph size but also obtain a maximally concurrent plan. A successful plan ensures that all facts that do not need resources are correctly achieved by that plan. A straightforward method for resource allocation is to assign a new or freed resource to any step that is involved in a resource conflict. Suppose that this method needs R resources. Now for all

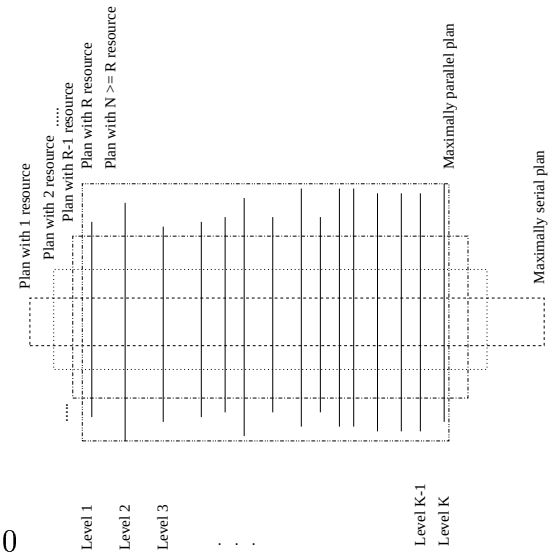


Figure 1.5. When there are enough resources to overcome any resource conflict, one gets a maximally parallel plan. Otherwise, the scheduler has to serialize the plan in line with the resource availability. For the most resource constrained problems (normally 1 resource), one gets the maximally serial plan.

problems with resources $N \geq R$, the same resource scheduling technique can be applied. As the number of resources decrease, the scheduler has to serialize the plan in line with the resource limitation. This may involve moving the parallel steps to less-constrained levels and introducing steps to release unnecessary allocations and re-allocate them where needed. Figure 1.5 gives a broad look at the type of plans obtained with different numbers of resources. Here, the same causal plan is being adapted to satisfy the resource availability constraints. In the pathological cases of the most resource constrained problems (normally 1 resource), one gets the maximally serial (i.e. serialized) plan.

Another benefit of my approach of postponing scheduling is during plan execution, when the user can easily specify the changed resource environment and obtain

new plans. In a traditional planner which performs integrated planning and scheduling, if some allocated resource becomes unavailable during plan execution, the whole plan has to be re-done or all other plans have to be enumerated until the correct plan for the new environment is found. In my case, only necessary resource re-allocation is needed because all the different plans for the problem differ only in how they allocate or de-allocate resources. To see an example, please refer to Appendix A where some of the plans for the *shuffle* problem are listed. Note that the plans differ only in how and when resources are allocated and de-allocated. If one schedule for resource management fails during plan execution, I only need to consider other alternative schedules.

1.3 Proposed Approach

In this dissertation, I focus on the role of resources in a planning domain and how resource-based reasoning can expedite planning. A major motivation is to scale planning algorithms to large, realistic domains where addition of more resources can provably reduce the planning time and not increase it. I first need to describe what resources are. *I assume that a resource is any object for which actions may contend and which is secondary (name is unimportant to the user) in a domain.* The user would want the planner to satisfy certain conditions (goals) but which resource objects are utilized to complete the plan is of no concern. Resource are identified either by the domain expert or through automated methods (see Chapter 2).

Next, the planner can be least committed about the identity of resource and postpone interactions among equivalent resources. After planning is complete, a scheduler can decide which resources to actually allocate. I have implemented this approach on top of Graphplan algorithm, and the resulting planner is not only more rational in its treatment of resources, but also achieves performance speedup. We reiterate that if all the resource allocation policies proposed by the planner lead to failure or inexecutable plans in a domain, this implies that planning and scheduling phases in this problem are infact, *not loosely coupled*. In such a case, the framework retains the ability to switch off resource abstraction and resort to traditional planning.

Here is an outline of the rest of this dissertation report. I start with a discussion about resources and their role in planning in Chapter 2. Next, I develop a planning formalism to abstract resources during planning and schedule them in a separate phase in Chapter 3. I proceed to discuss a procedural method for handling the various classes of the resource scheduling problem in Chapter 4. I also describe a declarative method for solving the resource scheduling problem as a Dynamic Constraint Satisfaction Problem in Chapter 5 and relate planner-scheduler interaction to it. The scheduled plan may need post-processing which is discussed in Chapter 6. Next I show through empirical results that my method is effective in Chapter 7. In Chapter 8, I make some observations on the overall nature of the problem and put my work in perspective with related works. Finally, I conclude in Chapter 9 with my contributions and future work.

Chapter 2

What are Resources ?

Before we discuss resources, we must formalize some terminology. All objects in a domain are instantiations of types. Types describe the entities present in a domain and their corresponding attributes. Attributes can be described by literals (binary variables) or multi-valued variables.

Recall from above that a resource is any object for which actions may contend and whose identity is unimportant to the user. For example, in the logistics domain, where there are some packages, trucks and planes at a set of places and the problems require the packages to be moved to their goal locations, the user does not care if $Plane2$ is selected instead of $Plane1$ but they will be concerned if the packet lands in *Boston* instead of *Philadelphia*. So, plane is a resource. But if the plane's identity does matter, then the plane type is not a resource[1]. The proposed definition captures the theme that scheduling should consider interactions among objects that do not

[1]Note that an object is a resource for all the actions in a domain and not per-action as has been modeled in some sytems like SIPE [35].

matter (i.e. resources) for the acceptance of the plan. Within this broad framework, resources may additionally provide domain control knowledge, suggestions for the planner about which interactions to disregard or special monitoring and execution capabilities, as is accomplished in some systems (Also see related works in Section 8).

A related issue that must be addressed is how the resources are identified in a planning domain. The most obvious approach is to let the domain expert specify the resources in the domain at the outset. I have implemented a resource specification format for this purpose. Appendix B illustrates the specification of *robot* as a resource in blocks world domain and *truck* and *airplane* in the logistics domain.

There are ways of automating resource identification process, however. We can assume that a resource is a type such that no object of that type figures explicitly in the goal specification. The motivation here is that if no objects of a type are necessarily required in the goal, these objects are secondary and will be useful only in the service of planning. The corresponding type is therefore secondary too. The definition can be easily used in the blocks world domain to detect that robot is a resource type or gripper is in gripper domain. But in more complex domains like logistics, there are multiple resources which can interact. Researchers have addressed identification of resources, for example, in TIM[9], by emulating the finite state machines implicit in the domain structure (legal operators and initial/ goal states) to automatically infer type structure of the domain, and extracting state invariants from them. Resources are objects corresponding to attribute spaces where an object can acquire or loose a property unconditionally, in contrast to a state space where corresponding objects

only exchange properties. I can easily incorporate their domain modeling techniques.

There are a number of definitions of resources in literature and corresponding resource reasoning approaches. SIPE [35] defines a resource as anything for which two actions contend when they have a harmful interaction in a nonlinear plan. The resource declaration is being used as a mechanism to specify domain control knowledge about ordering and there may be no physical mapping. For example, in the blocks world domain, blocks can be specified as "resources" and so can the robot arm. This is very unintuitive. Knoblock [18] uses similar ideas to specify a database as a resource in some action if the domain modeler wants to prevent one operator to execute in parallel with another operator when they require the same database. Though domain-specific ordering information can improve planning, we envision a broader role of resources during planning where some interactions can also be ignored.

An altogether different view of resources is taken in CIRCA[25]. Here, resources have no direct bearing on the planning domain or problem that is being solved and the goal is to come up with a plan that will also lead to optimal run-time resource usage without missing any deadline. From our perspective, this definition of resources does not help us in improving the performance of planning. But it does bring in real world constraints into planning which is a natural extension of the problem. IxTeT [21] defines a resource as any substance or set of objects whose cost or availability induces constraints on the actions that use them. The space station domain of HSTS [29] allows it to consider most of its physical components as resources and schedule for their optimal usage. Planning and resource constraints are converted to set of

common data-structures and search applied to get a plan. In these systems, planning has been extended to include specification about physical resource usage and this increases expressivity but not necessarily efficiency of planning.

Though we have only partially handled the resource modeling isses, we are in a position to tease scheduling apart from planning because modeling is an orthogonal issue and our results will still be applicable. For the rest of the dissertation, we will assume that resources have been identified either by the domain expert or by automated means like those discussed above.

Chapter 3

A New Planning Formalism

I am exploring a planning model in which resource allocation is teased apart from planning, and is handled in a separate "scheduling" phase (See Figure 3.1). I observe that a necessary condition for a schedulable plan is that it should be causally correct irrespective of the nature of resources. I can produce an abstract plan (P') which is correct sans the resource allocation and use it as a starting point for all planning problems that differ only in the number or amount of resources present. Next, based on the actual resource availability, the abstract plan will be allocated resources to produce an executable plan.

In most existing classical planning systems, sharable discrete resources are typically assumed to have infinite capacity (e.g. trucks can load any number of packages) and continous resources are assumed unlimited (e.g. fuel is available or not). The causal plan P' is also created under the most optimistic resource assumption (unlimited or infinite). While scheduling, the actual resources may be found insufficient to

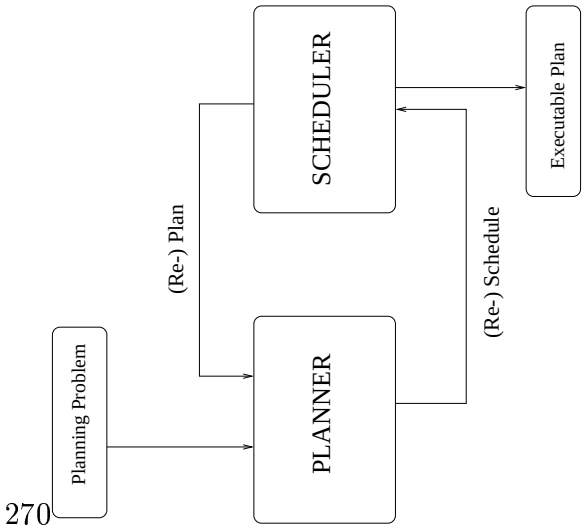


Figure 3.1. A generalized plan model for separate planning and scheduling.

assign to P' and this will force replanning to take place to honor the resource limits. The scheduler can aid the planner by informing it where re-planning is needed or re-plan itself.

3.1 Implemented Approach

Figure 3.2 summarizes my algorithm for handling this problem. In Graphplan terms, one can reduce both graph expansion and search overheads by abstracting the resources needed by actions during planning and ignoring all interactions between them, thereby obtaining a maximally concurrent plan. This plan will then be post-processed to allocate resources to actions including re-planning. See Figure 1.1 for a schematic overview of my approach.

Based on the ideas about resources from the previous chapter, I consider some

1. Identify resources: The system can recognize resources using already discussed methods.
2. If no resource information is available or resources are so low (usually one) that postponing their reasoning is counter-productive, perform conventional planning (i.e. if present, no postponement of interactions involving similar resources takes place during planning).
3. Suppose some of the objects are defined as resources. Planning proceeds as follows:
   (a) Assign dummy values to resources variables in the initial state and goal state such that equivalent resources have same dummy value.
   (b) Do not compute interference relationships (mutexes) between equivalent operators. Operators may still interfere due to other preconditions/ effects.
   (c) Complete planning.
4. Once a plan is obtained, we have to allocate resources to the actions in the plan and resolve resource conflicts using any scheduling criteria.
5. Return a valid final plan. As long as the algorithm ensures that all facts achieved during the planning phase are not undone by resource scheduling, the final plan is sound.

Figure 3.2. Synopsis of approach

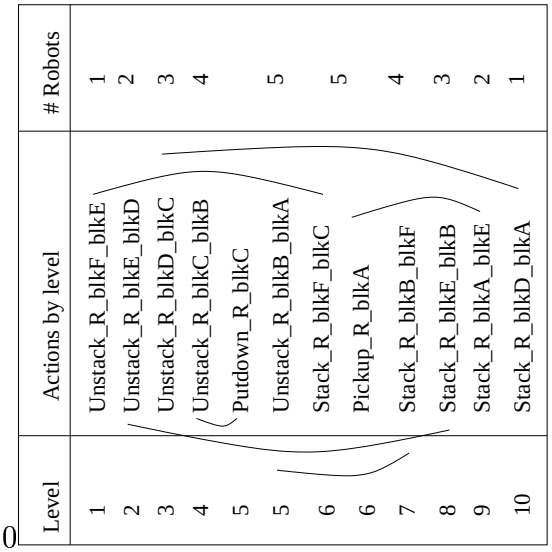


Figure 3.3. An resource-abstracted solution for *shuffle* problem. Curved lines show resource usage spans (see below). The number of resources needed at each level (which equals the number of spans crossing that level) is also shown.

object types to be identified as resources. Now, if resource abstraction switch is set, planning proceeds in the normal fashion, but with two important differences (Step 3):

- Dummy values are assigned to resource variables in the initial state such that equivalent[1] resources have the same dummy value.
- Interference relationships (mutexes in the case of Graphplan) between otherwise resource equivalent operators are ignored. Operators may still interfere due to other preconditions/ effects.

[1]In complex domains like logistics, all objects of a resource type are not equivalent (e.g. trucks in Boston are not substitutable for trucks needed in Phoenix) and this can be handled either by recognizing them as different equivalence classes within a resource type or as altogether different resource types. For now, I choose the latter and consider all objects in a resource type as equivalent.

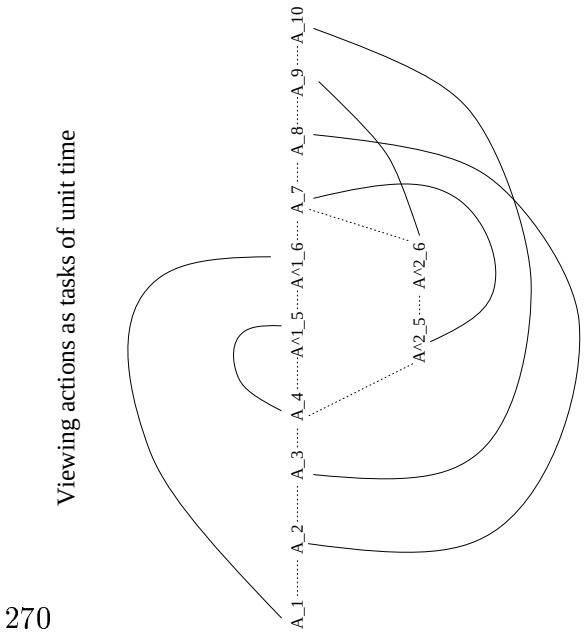


Figure 3.4. View of the resource-abstracted plan in Figure 3.3 as a task network to be scheduled. Curved lines show resource spans while dashed lines represent partial ordering constraints between tasks (actions).

If the problem is unsolvable at this stage, we know that resource scheduling is not going to make it solvable. Otherwise I give the resultant plan to the scheduler for resource allocation. An example of the plan generated for the *shuffle* problem, by disregarding inter-resource conflicts during planning, is shown in Figure 3.3. The plan consists of 10 time steps (levels) with the number of resources left allocated at each level shown in the right column (marked "#Robots"). The abstract plan is seen as a task network in Figure 3.4.

The aim of resource scheduling (Step 4) is to assign actual resources to the dummy resource variables, without undoing any causal relations established during planning. I have implemented an intuition driven procedural scheduling method (see

Chapter 4) and a declarative method where the resource allocation problem is posed as a DCSP and solved by a standard backjumping CSP solver (see Chapter 5). I begin its discussion by noting the nature of scheduling.

A straightforward method for resource allocation is to assign a new or previously freed resource to any action that is involved in a resource conflict. Suppose that this method uses a maximum of R resources. Now for all problems with resources $N \geq R$, the infinite resource assumption holds, and thus resource allocation is quite trivial. If this method fails, the allocation problem is solved through more elaborate resource management that also modifes the abstract plan. Planning phase can suggest similar intent to the scheduling phase in the form of following resource allocation policies:

1. Maintain concurrency of the plan.

2. Serialize the concurrent plan by moving actions from one step (level) to another less-constrained step.

3. Introduce actions to free unnecessary allocations and re-allocate the freed resources when needed again.

During scheduling, the aim should be to keep the cost of scheduling small enough so that the complexity of planning is not revisited in the resource scheduling phase. Note that least commitment on resources makes sense if there are multiple resources so that any resource conflict can be *potentially* overcome during scheduling by assigning different resources to the conflicting actions. But if one can detect

at the start itself that there is a single resource, resource postponement is useless in transferring planning complexity to scheduling and is infact counter-productive, because the planner is banking on concurrency in the plan while resource availability suggests a serial executable plan.

Even though we note that the pathological case of one resource can be easily detected and avoided upfront, I will persue resource postponement to illustrate how my scheduling method (specifically declarative method) also addresses this situation naturally.

3.2 Planner-Scheduler Interaction and Post-processing

The communication between planner and scheduler can be seen as suggestions (policies) by the planner about scheduling variables, their domains and constraints. The scheduler responds by flagging success or failure with the suggested parameters. If scheduling method fails to allocate resources in the context of given resources, time limit and nature of allocation policy, the responsibility transfers to the planner to change any of the permissible parameters and try again. The planner also has the option to take up non-abstracted planning at any stage. If resource allocation succeeds, the schedule and the allocation policy are used to derive an executable plan.

In summary, the different resource allocation policies supported include maintaining the concurrency of the plan, serializing the plan and inserting actions to free

and reallocate the resources. In Section 5.2, I explain the meaning of these policies in terms of scheduling variables and their values.

If resource allocation succeeds and no additional actions were inserted, the scheduled plan is executable and hence outputed. However, if new free/ reallocation actions were added by the scheduler, the scheduled plan has to be post-processed for necessary domain translation to make the final plan executable. This is discussed in detail in Chapter 6.

3.3 Comparing My Approach with Project Management

As mentioned in Chapter 1, AI Planning can handle small plans compared to what humans can already control in the real world. It would be interesting to compare planning and scheduling activities performed in project management with my approach. Recall that humans come up with the Work Breakdown Structure (WBS)[26] to identify the different tasks at some granularity and input this information to a project management tool along with estimates on time and resources for each task. Microsoft Project[27] is a standard tool used in industry for scheduling activities. Its guideline is that once the user has a task network defined, they should find the critical path in their project and compute slack time for individual tasks. Moreover, the task assignments be evaluated to identify over-allocated resources. To resolve the resource over-allocation, either the resources must be allocated differently or tasks must be

re-scheduled (euphemism for *delayed*) until the resource is available. A resource in a commercial project usually refers to people available but it can also be equipment, etc. Microsoft Project refers to "levelling" as the technique to resolve resource allocation by simply delaying certain tasks in a schedule until resource assigned to them are no longer overallocated.

Following are some of the allowed strategies for shortening a schedule in Microsoft Project and how they relate to planning:

1. Add lead (task starts before finish of predecessor) or lag time (task starts after the finish of predecessor). This refers to my scheduling policy of serializing the plan.

2. Decrease work of a resource on a task. My scheduling policy of introducing actions to free unnecessary allocations and re-allocate the freed resources when needed again, addresses this.

3. Avoid sequential order by changing the task relationships to allow more tasks to overlap or occur at the same time. In essence, the user re-plans to find a more concurrent plan of shorter length as can be done in planning.

4. Change the critical path. The user is directed to re-plan by changing the type of tasks, adding more tasks, or re-ordering the steps of the plan as can be done in planning.

5. Increase working condition i.e. capacity of each resource to accomplish more.

This is not an option in AI planning because the initial state (including the resources) are considered unchangeable.

6. Reduce the scope of a task by reducing the amount of work assigned to the task. This is not an option in AI planning because the domain operators are considered unchangeable.

7. For allocated tasks,

   (a) Increase the number of resources allocated to a task. In AI planning, actions usually need a single resource. However, it is possible to model this behavior too.

   (b) Increase resource availability by over time. Resources cannot be changed in AI planning.

One can also reduce the cost of the project schedule. Total cost of a schedule is the sum of cost of resources allocated to that project alongwith any fixed costs. One can reduce the cost of a project by re-planning or re-scheduling. Project management also allows the cost of a schedule to be decreased on an executable plan as the costs of resources (e.g. compensation of personnel) are generally different which is not usually the case in AI planning. On the other hand, scheduling by increasing the length of the plan is a viable option in AI planning but not usually allowed in commercial projects due to increased costs or loss of business opportunities.

3.4 Discussion on Plan Abstraction

Abstraction and least commitment has been widely studied in the context of planning, and therefore, my implementation has focussed on a sufficient sub-set to accentuate the resource allocation problem. Specifically, only the identity of resources is abstracted into variables and the constraints (bindings) among variables are deduced after an abstract plan is obtained, on the basis of causality and nature of resources. An actual example is shown in the next chapter in Section 4.1.

Information about variable bindings could also have been supplied during the planning process from causal dependencies as in partial-order planners like UCPOP[31]. Doing so is, however, more complex in Graphplan because the planner maintains sets of world states which leads to very few codesignation constraints (i.e. $v_i = v_j$ constraints for two variables) during the graph expansion phase. Many more bindings are found once the actual plan is known during the search phase because the supporting actions of predicates are discovered in this phase.

I have started looking in this general direction and have also enhanced graph construction with general resource variables that records forced bindings. Once codesignation constraints deduced during search are incorporated, we can have an equivalent set of resource constraints for resource allocation process. This is an incremental improvement in the representation and bookkeeping of binding constraints during planning.

Chapter 4

Scheduling as a Procedural Method

We are now ready to delve deeper into the resource scheduling phase. In this chapter, I provide an intuitive and computationally inexpensive treatment of resource allocation.

Let me state the resource allocation problem for resource R. The abstract plan has a set of action pairs $\langle A_i, A_j, \rangle \mid j \succ i$ where action A_i appears at time step i of the plan (actually written as A_i^m if it is the mth action at level i using resource R but we omit the superscript for clarity) and they constitute resource spans (S_{ij}:$\langle A_i, A_j, C\rangle$ s) that we have to allocate resources to. The effect C of action A_i is produced at level i and consumed at level j for the precondition of action A_j. I may also refer to spans by S_1, S_2, etc. if there is no need to identify its constituent actions in a given context.

4.1 Scheduling Resources: The Details

Based on the amount of resources available to the scheduler, and the way resource allocation phase interacts with the planning phase, the resource allocation problem can be classified into a variety of classes, as shown in Figure 4.1. I will start by describing the main classes briefly:

- Class INH-UNSOLV: If the problem is inherently unsolvable (for example, goals are *on_blockA_blockB* and *on_blockB_blockA* in blocks world), considering or ignoring resources during planning will not affect the solution but will help to create the planning graph faster. Hence, the problem can be handled by the planner more efficiently.

- Class INFRES: If indeed the resources are sufficient to overcome all resource conflicts, the scheduling view of the problem is the same as if there are infinite resources. For the *shuffle* problem, 5 robots are enough to overcome all resource conflicts in a plan and there is no reason why problems with 5 or more robots should take more time.

- Class FINRES: The remaining case is when the number of resources are small enough to cause resource conflicts but the problem is inherently solvable. This case can be decomposed, based on the difficulty of the resource scheduling problem, into a number of more specific sub-classes as shown in Figure 4.1. It turns out that I can handle many of these sub-classes through efficient (backtrackfree) methods.

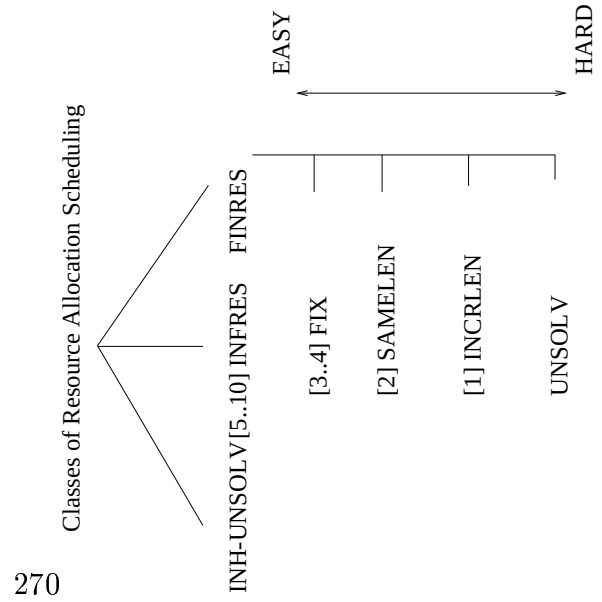


Figure 4.1. A Classification of resource allocation instances (with indication of resource quantities that make *shuffle* problem fall into each of the classes).

The complexity of resource scheduling instance, as well as the amount of modification needed to the original plan to allocate resources, increases from left to right and from top to bottom in Figure 4.1. Rather than use one general scheduling method for all classes, I cycle through scheduling methods tailored to each of the specific classes (from the easiest to the hardest). By using this approach, I am able to allocate resources with the least amount of modification to the given plan. This in turn ensures that the plans developed in our method are comparable in quality to those developed by the normal planner.

Posing the resource allocation problem: I formalize the resource allocation problem as a constraint satisfaction problem (CSP) for its simplicity even though only Class INFRES is solved here by a CSP solver. For other classes, I solve the allocation

problems by guided backtrack-free routines. In the next chapter, I augment the CSP formulation as a Dynamic CSP and solve all the classes declaratively.

I setup the resource allocation problem as a constraint satisfaction problem (CSP) by treating actions needing resources as variables and posting codesignation and non-codesignation constraints according to whether the actions can have the same resource allocated or not (resource conflicts). For this, I only have to consider facts (effects and preconditions of actions) that refer to the resource type under consideration (e.g. *arm-empty_R* or *holding_R_blockA* for ROBOT type in blocks world). This is because interactions involving facts with only non-resource objects (e.g. *clear_blockA*) would have been handled during planning itself.

I setup the CSP constraints as follows. The algorithm visits the plan level by level and determines the span (similar to a causal link in partial-order planners [23]) of allocation of resources. A span S_{ij} is a tuple $\langle A_i, A_j, C \rangle$ where the effect C of an action A_i is produced at level i and consumed at level j for the precondition of another action A_j. Actions A_i and A_j must have the same resource allocated[1]. The resource conflicts are that two actions A_j and A_k at the same level cannot have the same resources if resources are non-sharable. Moreover, any action A_k that falls within a span S_{ij} (i.e. levels i and j) and which either deletes or produces fact C must be allocated differently from actions A_i and A_j. Examples of spans in Figure 3.3 are $S_{1,6}$: $\langle A_1, A_6^1, holding\_R\_blockF \rangle$ and $S_{2,8}$: $\langle A_2, A_8, holding\_R\_blockE \rangle$.

[1]Information about variable bindings could also have been supplied by the planner from causal dependencies as elaborated in Section 3.4.

Function: *Allocate_Resources*
Parameters: Problem, AbstractPlan, ResourceType[]
Returns: Plan

- LocalPlan=AbstractPlan
- For each R_i=ResourceType,
  * LocalPool=NumberResources(ResourceType[R_i])
  * Span=GetResourceSpan(LocalPlan, R_i)
  * If Span is NIL, then continue with next ResourceType
  * Conflict=GetResourceConflict(LocalPlan, R_i)
  * For each Level L_j in LocalPlan
    • Need,RelevantSpan=Number,Cutsets of Span at L_j
    • If $Need_j$ is positive, then for each $RelevantSpan_k$
      1. If $RelevantSpan_k$ has been allocated, then continue with next RelevantSpan.
      2. Solve for the scheduling classes at Level L_j until an allocated plan (not NILPLAN) is obtained.
         (a) LocalPlan=$Solve\_INFRES$(LocalPlan, $RelevantSpan_k$, Conflict,R_i, LocalPool,L_j).
         (b) LocalPlan=$Solve\_FIX$(LocalPlan, $RelevantSpan_k$, Conflict,R_i, LocalPool,L_j).
         (c) LocalPlan=$Solve\_SAMELEN$(LocalPlan, $RelevantSpan_k$, Conflict, R_i, LocalPool,L_j).
         (d) Return OriginalPlanner(Problem).
      3. Increase LocalPool by 1 if $RelevantSpan_k$.E = Level L_j.
  * Continue with the next ResourceType
- Return LocalPlan

Figure 4.2. Pseudo-code for allocating resources

Solving the resource allocation problem: INFRES case: The resource allocation problem posed as a CSP is now ready for solving. Pseudo-code of my resource allocation algorithm is shown in Figure 4.2. See Figure 4.3 for an example. This method traverses the levels of the abstract plan and computes the spans that are relevant to a level L_j by finding spans that pass through L_j. In the *shuffle* example, the spans $S_{1,6}$ and $S_{2,8}$ are relevant at L_2. For each unallocated relevant span, it checks to see if there is a way to assign a resource. The check is made from the easiest to the hardest allocation instance in terms of the change to the abstract plan. Class INFRES is compatible with the conventional CSP formulation because no new actions (variables) are introduced. Hence, any standard CSP solver can be used in place of $Solve\_INFRES$. If the CSP problem was solved for all the levels in the abstract plan by $Solve\_INFRES$, this means that the bet made during planning (causal reasoning) that sufficient resources are available to handle all resource related interactions paid off and I have a solution. Otherwise, I have to free and reallocate resources as necessary. The scenario is similar to a Dynamic Constraint Satisfaction Problem (DCSP)[28] where activity variables (corresponding to free/ reallocating actions above) control when the normal variables be considered for value assignment. This gives me the insight to cast the complete scheduling problem declaratively as a DCSP and solve it by a standard solver, as is done in Chapter 5.

Solving the resource allocation problem: FINRES case: Based on the amount of resources, I can divide Class FINRES into a number of sub-classes as summarized in Figure 4.1. These sub-classes are currently detected during scheduling itself. The

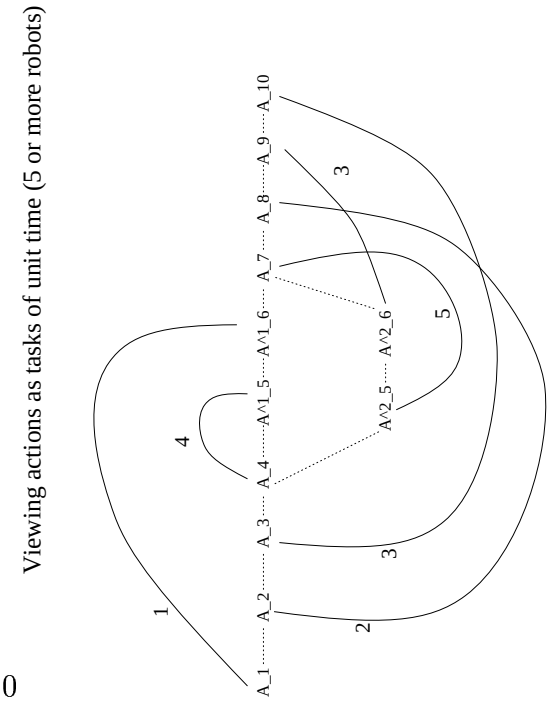


Figure 4.3. Scheduling task network of *shuffle* problem by INFRES. Curved lines show resource spans and numbers next to them are the resource allocated. Dashed lines represent ordering constraints.

general idea is that the algorithm traverses the plan level-by-level and goes on allocating the resources from the resource pool until there is a resource scarcity at some level i. A scarcity suggests a greater demand for a resource than its availability. It can be resolved, provided the number of resources are not too low, by re-arranging the resource usage pattern. Conflict resolution starts by going to a previous level (level i-1) of the plan and introducing a freeing[2] action to de-allocate a resource assigned to an action whose effect is not immediately needed. In the process, $Solve\_FIX$ shrinks the resource demand by one for all the levels from level i to level $j-2$ where the

[2]In general, adding actions to a plan can change its causal structure but I assume that there are actions or known sub-plans in the domain (e.g.*pickup, putdown* in blocks world) that can free and re-allocate resources without doing so. While there are pathological cases where the assumption may not hold, it seems to hold in most normal domains.

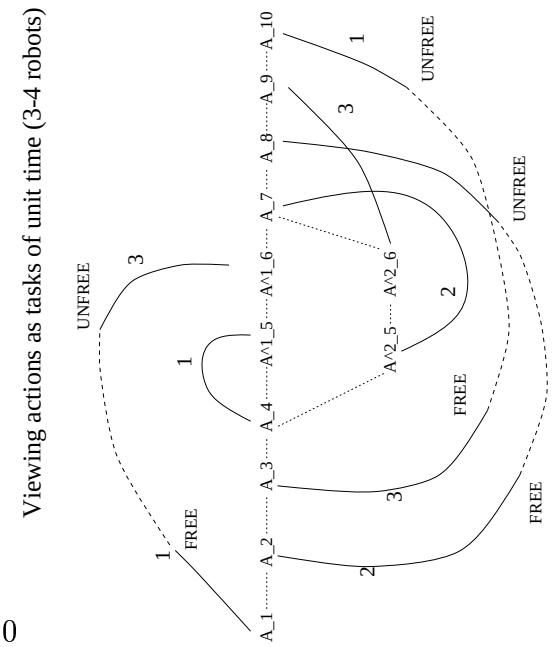


Figure 4.4. Scheduling task network of *shuffle* problem by FIX. Curved lines show resource spans and numbers next to them are the resource allocated. Dashed lines represent ordering constraints.

effect is needed at level j again (an action to reallocate the resource will be added at level j-1). See Figure 4.4 for an example. *Shuffle* problems with 3 and 4 robots can be handled by this method. In particular, with respect to Figure 3.3, the robot corresponding to span $\langle A_1, A_6^1, \mathit{holding\_R\_blockF}\rangle$ can be freed at level 2 and re-allocated at level 5. The number of robots needed at levels 3 and 4 will then reduce by 1. Problem instances solvable by this method are in the class FIX.

If resource scarcity persists, an unallocated action is moved to a subsequent level where it can be potentially allocated. But the move may force the consumers of its effects and any other actions whose effects can be clobbered by the potential move, to be moved too. Since unrestricted movement of actions can be as complex

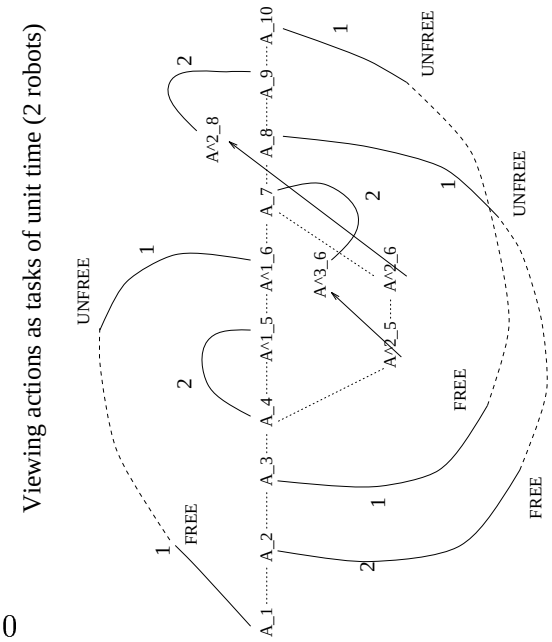


Figure 4.5. Scheduling task network of *shuffle* problem by SAMELEN. Curved lines show resource spans and numbers next to them are the resource allocated. Dashed lines represent ordering constraints. Arrows refer to the movement of actions to a lower, less-constrained level.

as planning itself, $Solve\_SAMELEN$ is not allowed to increase the plan length and is required to maintain the relative action positions. See Figure 4.5 for an example. *Shuffle* problem with 2 robots can be handled by this method. In the *shuffle* problem in Figure 3.3, the action A_5^2 can move down to level 6 to ease the scarcity. Since A_6^2 needs the $clear()$ effect of A_5^2, it then needs to move to level 8. Note that the length of the plan still remains the same. Problem instances solvable by this method are in the class $SAMELEN$ in Figure 4.1.

If the two above approaches fail, the allocation problem is in Class INCRLEN where the length of the abstract plan must be increased during scheduling (i.e. plan serialization affects plan length). *Shuffle* problem with 1 robot belongs to this class. I

give problems in this class either to the declarative scheduler discussed in next chapter or back to the original planner for solving it without any resource reasoning in the normal way. Class UNSOLV occurs when the number of resources are too small for any resource allocation to be feasible at all. If there are no resources, I can identify this class at the start of scheduling; otherwise it cannot be determined until after Class INCRLEN.

I can handle Classes INFRES, FIX and SAMELEN without backtracking in time polynomial in the length of the abstract plan (since the plan is traversed only once). For Class INCRLEN, resource abstraction is a penalty because the method goes back to the original planner and solves the problem without abstraction. However, as empirically shown later, this penalty is small and easily offset by the savings in other classes. As mentioned earlier, the reason I use a series of methods in this order is to keep the number of additional actions as small as possible while maintaining the optimal plan length.

4.2 A Mixed Declarative-procedural Scheduling Method

A different idea that I have also implemented involves a divide-and-conquer CSP phase that comes after planning is over. In this scheme, I verify that all actions that need resources can be allocated by solving a CSP problem for levels $1..N$. If this is not the case, I consider levels $1..K$ and $K..N$ and consider the CSP problem on each

portion of the plan, and so on. Once there is a solution for levels $1..K$, the scheduler can add actions to make the allocated resources available, either parallel to existing actions at levels beyond K or by inserting a new level after K. Moreover, for the $K..N$ plan portion, the scheduler has to separate it into $K..M$ and $M..N$ such that the resources need to be re-allocated after level M when they are needed.

In summary, here is an outline:

1. Suppose the plan has $1..N$ levels.
2. If CSP fails on $1..N$
   (a) Divide it into $1..K$ and $K..N$ $(K < N)$.
   (b) If CSP succeeds on $1..K$,
      i. Add actions after level K to free allocated resources but record current status of objects.
      ii. Divide $K..N$ into $K..M$ and $M..N$ such that objects are needed after M $(K < M < N)$.
      iii. Insert steps to reinstate the status of objects after M.
      iv. Recursively call self with levels of the plan as $K..M < insertedlevels > M..N$.
   (c) Else recursively call self with levels $1..K$.
3. Else return success.

In the context of the plan shown in Figure 3.3, I first consider a CSP on the complete plan (levels 1 to 10) and see if there is a consistent resource (robot) allocation. If such an allocation is found, scheduling is done. For robots 5..10, a solution is available straightaway. If the CSP fails, the algorithm looks for a solution in 1..5 and 5..10 levels with a premise that each robot used up after level 5 will be freed before it is needed again in the 5..10 levels. The algorithm continues the binary divide and conquer approach until it succeeds in every sub-region. In the extreme case of 1 robot, the (sub-) CSP is eventually set for each level.

Though I have chosen to split in the middle of a range for convenience, I could have selected any other point as well. A more informed approach would be to produce a resource profile based on resource demands over the whole plan, and decide about the split points at every level with conflict. Either way, the overall algorithm is unaffected.

The recursive algorithm outlined above can enable the extended GRAPHPLAN to solve the $shuffle$ problem with 1 to 4 robots as expected in the blocks world domain. Currently, I can solve the individual sub-allocation tasks for all the classes of the scheduling problem (if task divison is necessary) and assume successful plan merging. Plan merging needs to be procedurally implemented.

I recognized the DCSP formulation of the resource allocation problem as more interesting and present a completely declarative scheduling approach in the next chapter.

4.3 Summary and Issues

In the procedural approach, an algorithm tries to capture the desirable scheduling decisions. The steps may be intuitive but they need not guarantee optimality of solution. Here is a summary of some of the procedural methods for resource allocation discussed:

1. Assign a free resource to the action under consideration, A_i.

2. Free an immediately unnecessary resource from a span at the previous level and use its resource for allocation to A_i. The resource is reallocated before the end of the span. Note that all immediately unnecessary resources of spans must be considered for completeness.

3. The freeing of resource can occur at the current level (level i) but all unallocated A_i s will then have to be moved to lower levels. Related A_j s (e.g. the end of the span) may need to be marked *move_ready*.

4. A_i can be moved to a lower level. Related A_j s (e.g. the end of the span or actions whose effects get threatened) may need to be marked *move_ready*.

5. If an action is *move_ready* at a level, it can be moved to all lower (and perhaps upper) levels. These are backtrackable points.

As a result of too many backtrackable allocation choices, the procedural method becomes complicated, harder to verify and clearly unattractive. However,

on the pros side, failure in scheduling at any of the above stages conveniently tells the planner about the nature of the allocation problem and how the latter may change the plan (insert freeing-reallocating actions, moving actions) to make the plan schedulable. Moreover, resource allocation is backtrack-free and efficient as the abstract plan is surveyed only once.

Chapter 5

Scheduling as a declarative CSP

Let me restate the resource allocation problem for resource R. The abstract plan has a set of action pairs $\langle A_i, A_j \rangle \mid j \succ i$ where action A_i appears at time step i of the plan (actually written as A_i^m if it is the mth action at level i using resource R but I omit the superscript for clarity) and they constitute resource spans ($S_{ij} : \langle A_i, A_j, C \rangle$ s) that I have to allocate resources to. The effect C of action A_i is produced at level i and consumed at level j for the precondition of action A_j. I may also refer to spans by S_1, S_2, etc. if there is no need to identify its constituent actions in a given context.

I note that the nature of problem is such that every resource allocation choice is a backtrackable point. Moreover, actions can move to lower or upper levels if causal dependencies allow them. The problem is strictly more difficult than backward search of graphplan because actions can move across levels.

Action	Vars	Possible Values	Comments
A_i	$\langle RA_i, PA_i \rangle$	$\{1..N\}$,$\{i..L\text{-}1\}$	N is number of resources
A_j	$\langle RA_j, PA_j \rangle$	$\{1..N\}$,$\{j..L\}$	L is max length of plan
F_{ij}	$\langle RF_{ij}, PF_{ij} \rangle$	$\{\perp,1,..N\}$,$\{\perp,i{+}1..L\text{-}2\}$	$\perp \Rightarrow F_{ij}$ is not needed
U_{ij}	$\langle RU_{ij}, PU_{ij} \rangle$	$\{\perp,1,..N\}$,$\{\perp,i{+}2..L\text{-}1\}$	$\perp \Rightarrow U_{ij}$ is not needed
N_i	$\langle PN_i \rangle$	$\{i..L\}$	N_i is R insensitive

Table 5.1. Constraints on action variables and their values while scheduling for resource R. Number of resource of type R are N and the permitted length of the plan is L.

5.1 Declarative Scheduling

As an alternative to procedural scheduling, I can specify the resource allocation constraints declaratively and let a constraint solver find a satisfying assignment for resources for each action. The problem is an instance of a Dynamic Constraint Satisfaction Problem (DCSP)[28] where activity variables (corresponding to free/ reallocating actions above) control when the normal variables be considered for value assignment. I now discuss the necessary preparations.

Each action using resource R has two variables associated with it, RA_i for resource allocated and PA_i for position or level where action A_i will appear. Position of an action is also a variable because one way to allocate resources, given a resource limit, is by serializing the parallel plan. Actions that do not participate in manipulation of resources are noted as N_i and their corresponding position variable is PN_i. Given a span $(S_{ij}{:}\langle A_i, A_j, C\rangle)$, I associate two actions, F_{ij} for freeing the resource and U_{ij} to reallocate the resource. The constraints on variables and their values are

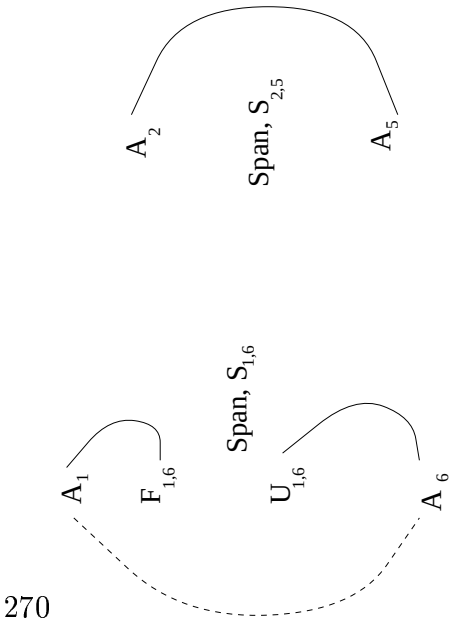


Figure 5.1. Example of spans with freeing/unfreeing actions on the left and without them on the right.

listed respectively in Tables 5.1, 5.2 and are discussed below.

The domain of a resource variable is the range of available resources R, $\{1..N\}$ and is augmented for the resource variables of freeing/ reallocating actions to includes a dummy value $\perp$ (NULL) signifying that the corresponding action is not needed. The domain of a position variable includes its current position in the plan and all the remaining valid positions (levels). For the position variables of freeing/ reallocating actions, the valid positions also includes a dummy value $\perp$ signifying that the corresponding action is not needed. The domain of all the variables are summarized in Table 5.1.

The constraints on resource values enforce that the resource used by A_i is the same as A_j unless there are freeing and reallocating actions present. If they are

Identifier	Relationship among variables	Comments
a)	$RA_i = RF_{ij} \vee$ $(RF_{ij} = \perp \wedge RA_i = RA_j)$	If freeing action is needed, it uses the same resource as span starting the action
b)	$RA_j = RU_{ij} \vee$ $(RU_{ij} = \perp \wedge RA_i = RA_j)$	If reallocating action is needed, it uses the same resource as span ending the action
c)	$RF_{ij} \neq \perp \Leftrightarrow RU_{ij} \neq \perp$	If freeing action occurs, reallocating action also occurs and vice-versa
d)	$PF_{ij} \prec PU_{ij} \vee PF_{ij} = PU_{ij} = \perp$	Position of freeing action is before position of reallocating action or both are NULL
e)	$PA_i \prec PF_{ij} \vee PF_{ij} = \perp$	Position of freeing action is after start of span or is NULL
f)	$PA_j \succ PU_{ij} \vee PU_{ij} = \perp$	Position of reallocating action is before end of span or is NULL
g)	$RF_{ij} = \perp \Leftrightarrow PF_{ij} = \perp$	If freeing action is not needed, its position is NULL and vice-versa
h)	$RU_{ij} = \perp \Leftrightarrow PU_{ij} = \perp$	If reallocating action is not needed, its position is NULL and vice-versa
i)	$PA_i \prec PA_j$	Position of action starting a span is before the action ending it
j)	$PN_i \prec PN_j$, $PN_i \prec PA_j$, $PA_i \prec PN_j$	Relative ordering of actions in the plan is maintained irrespective of resource usage
k)	Non-sharable resource constraints (see Table 5.3)	If segments of two spans overlap, they cannot share resources over that segment

Table 5.2. Relationship among action variables.

Condition	Constraint on values
$PF^1_{ij} = PU^1_{ij} = PF^2_{ij} = PU^2_{ij} = \perp$	INTERACT(PA^1_i, PA^1_j, PA^2_i, PA^2_j) $\Rightarrow RA^1_i \neq RA^2_i$
$PF^1_{ij} = PU^1_{ij} = \perp; PF^2_{ij}, PU^2_{ij} \neq \perp$	INTERACT(PA^1_i, PA^1_j, PA^2_i, PF^2_{ij}) $\Rightarrow RA^1_i \neq RA^2_i$ INTERACT(PA^1_i, PA^1_j, PU^2_{ij}, PA^2_j) $\Rightarrow RA^1_i \neq RA^2_j$
$PF^1_{ij}, PU^1_{ij} \neq \perp$; $PF^2_{ij} = PU^2_{ij} = \perp$	INTERACT(PA^2_i, PA^2_j, PA^1_i, PF^1_{ij}) $\Rightarrow RA^2_i \neq RA^1_i$ INTERACT(PA^2_i, PA^2_j, PU^1_{ij}, PA^1_j) $\Rightarrow RA^2_i \neq RA^1_j$
$PF^1_{ij}, PU^1_{ij}, PF^2_{ij}, PU^2_{ij} \neq \perp$	INTERACT(PA^1_i, PF^1_{ij}, PA^2_i, PF^2_{ij}) $\Rightarrow RA^1_i \neq RA^2_i$ INTERACT(PA^1_i, PF^1_{ij}, PU^2_{ij}, PA^2_j) $\Rightarrow RA^1_i \neq RA^2_j$ INTERACT(PU^1_{ij}, PA^1_j, PA^2_i, PF^2_{ij}) $\Rightarrow RA^1_j \neq RA^2_i$ INTERACT(PU^1_{ij}, PA^1_j, PU^2_{ij}, PA^2_j) $\Rightarrow RA^1_j \neq RA^2_j$

Table 5.3. INTERACT(a,b,c,d) = (a $\leq$ d $\wedge$ c $\leq$ b). When two sections of resource spans interact, the interacting sections cannot share the same resource. The superscript refers to the spans S_1 or S_2 for which the actions (and variables) are applicable.

present, A_i and F_{ij} have the same resource as also do U_{ij} and A_j. The constraints on position variables enforce the relative order between the actions. The position of A_i has to be before A_j while the freeing action, if present, has to be after A_i and the reallocating action, which follows a freeing action, has to be before A_j. The partial ordering of the actions in the abstract plan is also maintained irrespective of resource usage. The exact constraints on the values of variables is summarized in Table 5.2.

Moreover, if a resource is non-sharable, additional constraints have to be specified as summarized in Table 5.3. The gist of the constraints is that if any segment

of a span interacts with that of another, the two spans cannot share a resource. For example, spans $S_{1,6}$ and $S_{2,8}$ interact between levels 2 and 6. Therefore, they cannot share a robot (resource) in this interval unless their allocated robots are freed. Freeing (and reallocating) actions will result in sub-intervals over which a robot cannot be shared.

We are ready to state the CSP problem. In addition to constraints in Tables 5.2, 5.3, we have top-level constraints:

- The number of resource allocations at a level must not exceed the available resources. This constraint is implicitly stated.

  Sum $(A_i^m) \leq N$, $i = 1..L$, $m = 1..\mid A_i \mid$ (i.e. number of actions at level i)

- To optimize the plan, we can set the objective function as minimize the total number of actions in the plan. This may seem strange at first glance because scheduling is usually related with minimizing resource usage. But recall that the number of resources in a problem is part of the initial specification and there is no incentive to minimize resources in classical planning. However, since the resource allocation problem is formulated as a CSP, such a requirement can be easily handled. The constraint to minimize actions is currently enforced implicitly by the CSP solver in conjunction with the scheduling policies (discussed later). It can also be stated explicitly as an optimizing CSP problem.

  Objective: Min Sum($\mid A \mid$) (i.e. number of actions in the plan)

The CSP encodings are solved with GAC-CBJ, a CSP solver that performs generalized arc-consistency and conflict directed backjumping used by CPLAN[34]. However, if the resource allocation problem is solvable, any systematic and complete method will find it.

5.2 Policies for Planner-Scheduler Interaction

As mentioned, the communication between planner and scheduler can be seen as suggestions (policies) by the planner about scheduling variables, their domains and constraints. The scheduler responds by flagging success or failure with the suggested parameters. If scheduling method fails to allocate resources in the context of given resources, time limit and nature of allocation policy, the responsibility transfers to the planner to change any of the permissible parameters and try again. The planner also has the option to take up non-abstracted planning at any stage. If resource allocation succeeds, the schedule and the allocation policy are used to derive an executable plan.

In the previous chapter (and also [32]), the resource allocation problem was classified into a variety of classes (see Figure 5.2) on the basis of how resource allocation phase interacts with the planning phase depending on the amount of resources. The complexity of resource scheduling instance, as well as the amount of modification needed to the original plan to allocate resources, increases from left to right and from top to bottom. It was proposed that rather than using one general scheduling method for all classes, one could cycle through the scheduling methods tailored to each of the

Allocation Policy	Constraint on values
Maintain concurrency (Class INFRES)	$PA_i = i$, $PA_j = j$, $RA_i = RA_j = \{1,..N\}$ $PF_{ij} = PU_{ij} =$ $RF_{ij} = RU_{ij} = \perp$
Serialize plan	$PA_i = \{i,..L^{MAX}\text{-}1\}$, $PA_j = \{j,..L^{MAX}\}$, $RA_i = RA_j = \{1,..N\}$ $PF_{ij} = PU_{ij} =$ $RF_{ij} = RU_{ij} = \perp$
Introduce Free/ Reallocate action	(Class FINRES)
Class FIX	$PA_i = i$, $PA_j = j$, $RA_i = RA_j = \{1,..N\}$ $PF_{ij} = \{\perp,i\text{+}1\}$, $PU_{ij} = \{\perp,j\text{-}1\}$, $RF_{ij} = RU_{ij} = \{\perp, 1,..N\}$
Class SAMELEN	$PA_i = \{i,..L\text{-}1\}$, $PA_j = \{j,..L\}$, $RA_i = RA_j = \{1,..N\}$ $PF_{ij} = \{\perp,i\text{+}1,..L\text{-}2\}$, $PU_{ij} = \{\perp,j\text{-}1,..L\text{-}1\}$, $RF_{ij} = RU_{ij} = \{\perp, 1,..N\}$
Class INCRLEN	$PA_i = \{i,..L^{MAX}\text{-}1\}$, $PA_j = \{j,..L^{MAX}\}$, $RA_i = RA_j = \{1,..N\}$ $PF_{ij} = \{\perp,i\text{+}1,..L^{MAX}\text{-}2\}$, $PU_{ij} = \{\perp,j\text{-}1,..L^{MAX}\text{-}1\}$, $RF_{ij} = RU_{ij} = \{\perp, 1,..N\}$

Table 5.4. Allocation policy and restrictions on values of variables. L^{MAX} is some maximum length ($L^{MAX} \succ L$) upto which the steps of the plan can be increased.

specific classes (from the easiest to the hardest).

I can support the different classes in the form of resource allocation policies. The different policies supported include maintaining the concurrency of the plan, serializing the plan and inserting actions to free and reallocate the resources. The DCSP formulation allows the scheduler to interpret the resource allocation policy prescribed by the planner (see Figure 1.1) **in terms of constraints on the values of variables**.

Table 5.4 summarizes the different policies and what they imply in terms of legal values of variables. Maintaining concurrency of the plan corresponds to all actions A_i in the plan being immovable while no freeing/ reallocating actions are permitted. The domain of RA_i is the range of available resources. Serializing the plan implies that the action of the plan can move subject to an upper plan length, L^{MAX}, provided by the planner. Again, no freeing/ reallocating actions are permitted to be inserted. An example of L^{MAX} is the number of actions in the plan, which allows the plan to be completely serialized.

In introducing resource freeing/reallocating actions, I identify three sub-cases. If actions are considered immovable, this corresponds to Class FIX. Here, the freeing action (F_{ij}) can be introduced immediately after A_i while the reallocating action (U_{ij}) can come immediately before A_j.

The second sub-case is when the actions are allowed to move upto the length of the abstract plan, and this corresponds to Class SAMELEN. Finally, the actions are allowed to move till any upper limit L^{MAX} ($L^{MAX} \succ L$).

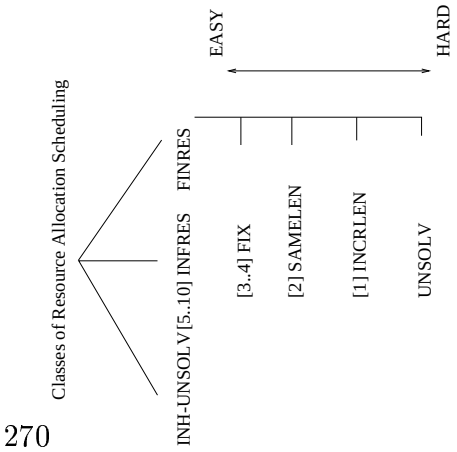


Figure 5.2. (Figure repeated for convenience) A Classification of resource allocation instances (with indication of resource quantities that make *6-shuffle* problem fall into each of the classes). INH-UNSOLV refers to causally infeasible plan for which no scheduling is needed, while UNSOLV refers to an unschedulable plan.

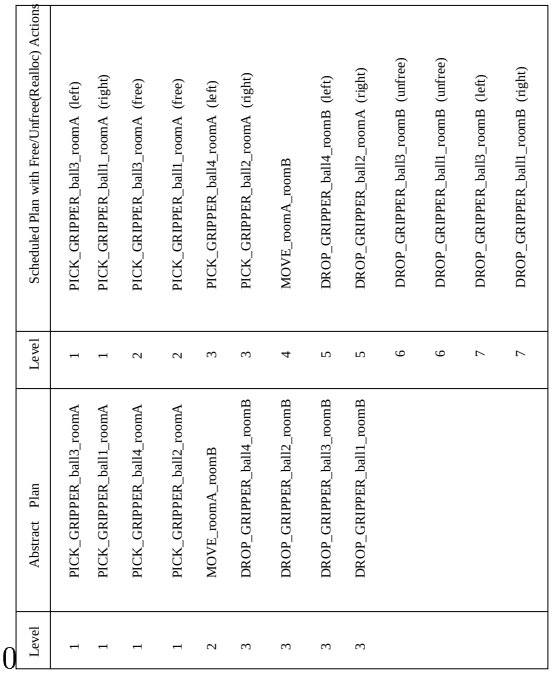
The advantage of multiple allocation policies is that it helps the planner in communicating the plan preferences of the user to the scheduler. For example, the end user may prefer plans with lower number of actions in the plan at the cost of increased plan length. Policies also make sense computationally. The complexity of the CSP problem increases with the domain size of its variables since it is $O(k^n)$ where there are n variables with average domain size of k. The idea of having multiple allocation policies is useful in guiding the scheduler towards easier resource allocation problems first.

Chapter 6

Post-processing the Scheduled Plan

If resource allocation succeeds and no additional actions were inserted, the scheduled plan is executable and hence output. However, if new free/ reallocation actions were added by the scheduler, the scheduled plan has to be post-processed for necessary domain translation to make the final plan executable.

Domain translation corresponds to replacing freeing/ reallocation actions with the corresponding actions (in general, sub-plans) in the domain that achieve similar resource-relevant effects. This information can be specified either by the user as in my implementation or derived automatically from a domain modeling tool like TIM [9]. In the blocks world domain, freeing can correspond to PUT-DOWN action which places a block on the table while reallocation (unfreeing) can correspond to PICK-UP action which holds a block again (See also Appendix B). Since there may be multiple sub-plans in general, a cost measure can be used to select which sub-plan to use. Domain translation may increase the length of the scheduled plan and introduce



Level	Abstract Plan	Level	Scheduled Plan with Free/Unfree(Realloc) Actions
1	PICK_GRIPPER_ball3_roomA	1	PICK_GRIPPER_ball3_roomA (left)
1	PICK_GRIPPER_ball1_roomA	1	PICK_GRIPPER_ball1_roomA (right)
1	PICK_GRIPPER_ball4_roomA	2	PICK_GRIPPER_ball3_roomA (free)
1	PICK_GRIPPER_ball2_roomA	2	PICK_GRIPPER_ball1_roomA (free)
2	MOVE_roomA_roomB	3	PICK_GRIPPER_ball4_roomA (left)
3	DROP_GRIPPER_ball4_roomB	3	PICK_GRIPPER_ball2_roomA (right)
3	DROP_GRIPPER_ball2_roomB	4	MOVE_roomA_roomB
3	DROP_GRIPPER_ball3_roomB	5	DROP_GRIPPER_ball4_roomB (left)
3	DROP_GRIPPER_ball1_roomB	5	DROP_GRIPPER_ball2_roomA (right)
		6	DROP_GRIPPER_ball3_roomB (unfree)
		6	DROP_GRIPPER_ball1_roomB (unfree)
		7	DROP_GRIPPER_ball3_roomB (left)
		7	DROP_GRIPPER_ball1_roomB (right)

270

Figure 6.1. The abstract plan (on left) is scheduled (on right) by inserting resource manipulating actions.

non-minimality as illustrated below.

Consider the case of gripper domain [24] where balls have to be moved between rooms. The abstract plan in Figure 6.1 was scheduled by inserting actions that assume that gripper could be freed and reallocated at different levels where needed. The resultant plan is translated based on the resource specification in Figure 6.2 to produce the left plan in Figure 6.3. This plan is non-minimal and can be post-processed to remove redundant actions. Post-processing can also check that the translated plan is executable. This check is needed because though the planner suggested to the scheduler that resource freeing/ reallocating actions are available in the domain, inserting those actions may interfere with actions already present in the plan.

```
(resource GRIPPER
    (free
      (means
          (effects (free ⟨grip⟩))
          (params (⟨grip⟩ GRIPPER) (⟨ob⟩ OBJECT)
          (⟨room⟩ ROOM))
          (plans
             (p11
                (s1 (DROP ⟨grip⟩ ⟨ob⟩ ⟨room⟩)))))))
    (unfree
       (means
          (effects (carry ⟨grip⟩ ⟨ob⟩))
          (params (⟨grip⟩ GRIPPER) (⟨ob⟩ OBJECT)
          (⟨room⟩ ROOM) (⟨to-room⟩ ROOM)
          (⟨from-room⟩ ROOM))
          (plans
             (p12
                (s1 (MOVE ⟨to-room⟩ ⟨from-room⟩))
                (s2 (PICK ⟨grip⟩ ⟨ob⟩ ⟨room⟩))
                (s3 (MOVE ⟨from-room⟩ ⟨to-room⟩)))))))
```

Figure 6.2. Resource specification of gripper including 1-step subplan (DROP) to free and 3-step subplan (MOVE, PICK, MOVE) to reallocate the gripper in a room.

Level	Domain translated plan	Level	Post-processed (minimal) plan
1	PICK_GRIPPER_ball3_roomA (left)		
1	PICK_GRIPPER_ball1_roomA (right)		
2	DROP_GRIPPER_ball3_roomA (left)		
2	DROP_GRIPPER_ball1_roomA (right)		
3	PICK_GRIPPER_ball4_roomA (left)	1	PICK_GRIPPER_ball4_roomA (left)
3	PICK_GRIPPER_ball2_roomA (right)	1	PICK_GRIPPER_ball2_roomA (right)
4	MOVE_roomA_roomB	2	MOVE_roomA_roomB
5	DROP_GRIPPER_ball4_roomB (left)	3	DROP_GRIPPER_ball4_roomB (left)
5	DROP_GRIPPER_ball2_roomB (right)	3	DROP_GRIPPER_ball2_roomB (right)
6	MOVE_roomB_roomA	4	MOVE_roomB_roomA
7	PICK_GRIPPER_ball3_roomA (left)	5	PICK_GRIPPER_ball3_roomA (left)
7	PICK_GRIPPER_ball1_roomA (right)	5	PICK_GRIPPER_ball1_roomA (right)
8	MOVE_roomA_roomB	6	MOVE_roomA_roomB
9	DROP_GRIPPER_ball3_roomB (left)	7	DROP_GRIPPER_ball3_roomB (left)
9	DROP_GRIPPER_ball1_roomB (right)	7	DROP_GRIPPER_ball1_roomB (right)

Figure 6.3. The inserted actions in the scheduled plan are translated to domain-specific actions/sub-plans (on left) and post-processed to remove non-minimal (redundant) actions (on right).

In general, adding actions to a plan is risky because this can change its causal structure and lead to interactions. But by using domain translation in a principled manner, planning can truly tap the benefits of decoupling resource reasoning from its causal reasoning phase. If all the allocation policies lead to inexecutable plans in a domain, this implies that planning and scheduling were infact, *not loosely coupled*. It may also be the case that the users' preference prevents the planner from suggesting a successful policy (like inserting new actions) to the scheduler. In such a situation, the framework retains the ability to switch off resource abstraction and resort to traditional planning.

Chapter 7

Implementation and Evaluation

The idea of decoupling causal and resource reasoning provides multiple implementation choices in both solving the abstract planning problem and in scheduling resources which I highlight in detail here. I consider planning followed by different forms of resource scheduling (procedural or declarative) and show that my approach gains in performance while being less susceptible to the quantity of resources.

7.1 Implementation Choices

By separating causal and resource reasoning, we have multiple choices in the selection of methods for abstract planning and resource scheduling which are summarized in Figure 7.1. I have developed a prototype implementation of my approach on top of Graphplan where the causal plan is obtained by Graphplan and scheduling is handled by either procedural or declarative methods. I have also used GP-CSP[5], a planner

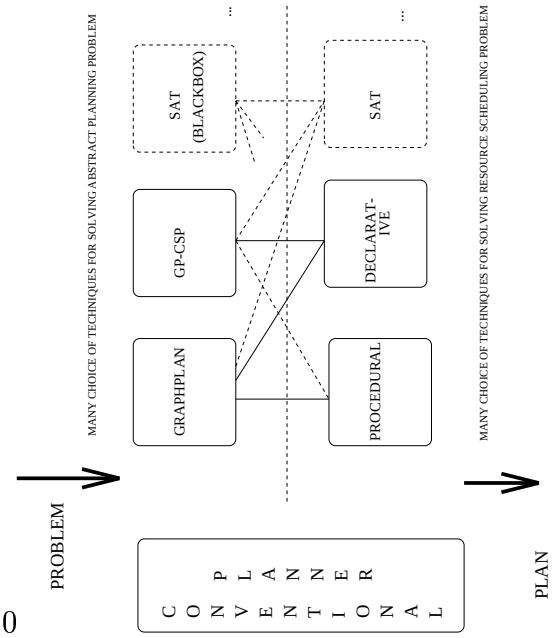


Figure 7.1. Choices for causal and resource reasoning. Boxes with solid lines show choices that have been investigated.

which converts the plan graph of Graphplan into a CSP problem and extracts plans from it with a standard CSP solver, for causal reasoning alongwith the declarative scheduler.

Since there are multiple choices in either of the two phases, the *quality* of the causal plan obtained after abstract planning and the *quality* of the schedule become important in making the selection. A plan whose actions are *perfectly justified*[8] cannot have an unnecessary action and is more desirable plan than another plan with redundant actions. Similarly, a schedule which uses lesser resources is more desirable than another schedule which uses more.

We can also try to improve the selected methods for each phase. I discuss some ideas to make the declarative solver efficient.

7.1.1 Improving CSP Performance

A simple idea to improve CSP is to narrow the domains of positional variables based on a *necessarily before* analysis of the abstract plan. I topologically sort the partial order of the actions in the plan to know their positional upper limit given a L^{MAX}. This seems to help the CSP but not in a major way in my experiments. One reason for it may be that higher payoffs will occur when the abstract plan is quite serial and the time limit is very high. But in such cases, the need to increase the length of the plan may not be usually there.

Another interesting performance issue is related to how the the length of the plan is increased during scheduling. One can either increase the length linearly or exponentially (e.g. doubling it) to arrive at a schedulable L^{MAX}. But this can lead to wasted search time if the length is too low in the first case and too high in the second case. I select a mixed approach where a growth factor controls how much (say 10%) of the length is increased linearly from the current plan length before doubling it. The idea is to increase length exponentially for smaller plan lengths and linearly for higher plan lengths to converge at the sufficient plan length fast with reduced wastage.

I have also investigated how to improve the encoding of CSP constraints. To this effect, I have tried a number of alternative formulations of constraints with lower arity but. One of the challenges we face is that since the GAC-CBJ solver only calls a constraint checker if a variable in that constraint is assigned, there can be a situation where the solution is obtained but it violates some constraint as the corresponding

checker was not called! The art is in ensuring that all the constraints are of lower arity while ensuring all sufficient checkers are called at each stage.

7.2 Solving Problems

I now compare the performance of my prototype to standard Graphplan, as I vary the amount of resources. First, I consider the *blocks world* (where the number of robot hands is varied) and the *logistics* domain (where the number of trucks at different cities are varied). I consider planning followed by different forms of resource scheduling (procedural or declarative) and show that my approach gains in performance while being less susceptible to the quantity of resources. I prefer declarative methods in further experimentation, thereafter.

Next, I investigate the relationship between the nature of resources (sharable v/s non-sharable) and scheduling time. I consider the *rocket* domain where the sharable rocket can be used to transport items between location. I also consider the inter-play between these type of resources in a domain I created called the *shuttle* domain. In this domain, there are sharable shuttles and non-sharable cranes to move boxes between inter-stellar bodies (e.g. Earth and Moon). I consider problems where the number of both of these resources are varied independently. Sharable resources pose lesser scheduling conflicts compared with non-sharable resources because they allow overlap in resource spans and it is not obvious if problems with them will also benefit from my approach. I show that my approach is still quite useful.

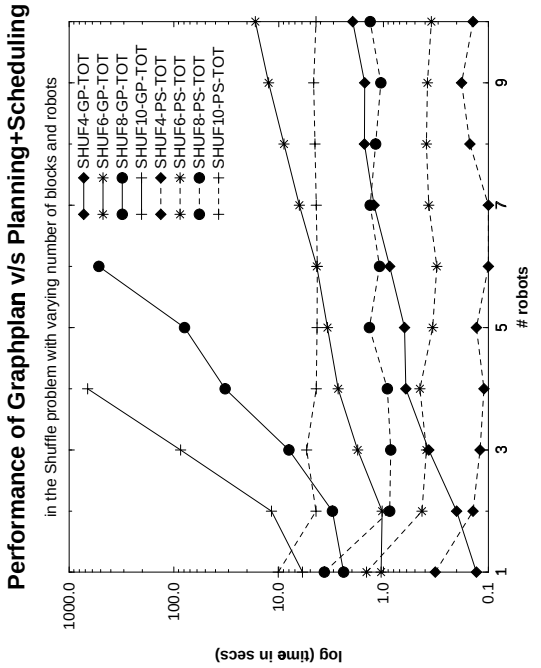
7.2.1 Planning and Procedural Scheduling

Figure 7.2 shows the results for the *shuffle* problems with 4, 6, 8 and 10 blocks as the number of robots are varied from 1 to 10. The plots clearly show that planning followed by scheduling (PS-TOT) is significantly better than original planning in the presence of resources (GP-TOT). Let us consider the 6-block *shuffle* problem in detail.

In my method, the planning time is constant and the scheduling time is dependent on the specific class (in Figure 4.1) that the problem falls into. In the *shuffle* case, problems with 5 to 10 robots are in class INFRES, problems with 3 and 4 robots are in class FIX, and problems with 2 robots are in class SAMELEN. The first class needs no modifications to the plan, the second class requires insertion of new actions, while the third also requires movement of actions across levels (steps). *Shuffle* problems with 1 robot are in class INCRLEN, and are sent back to the planner. This is reflected by the dip in the plot (SHUF6-PS-TOT) after 1 robot case.

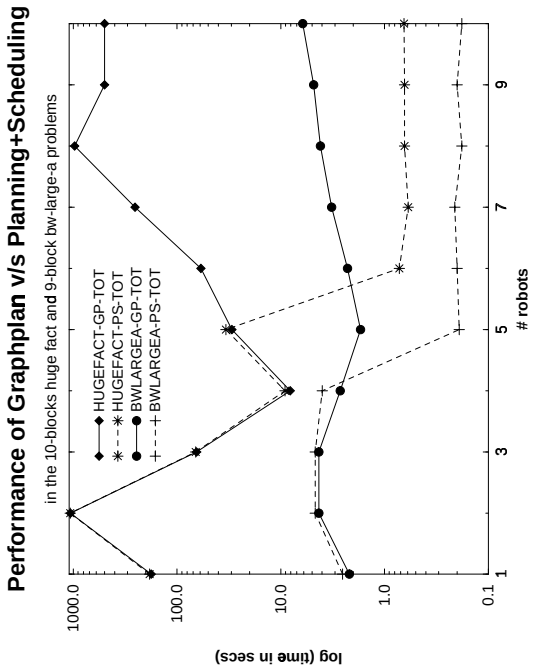
Although not shown in the plots, the length of the plan with our approach is the same as with original Graphplan in terms of the number of levels and actions. As the number of resources (here robots) increase, my approach takes almost constant time whereas the performance of Graphplan is adversely impacted to a significant extent.

In Figure 7.4, we see the performance of my method on a different blocks world problem, namely, the 8-block inversion problem. Problems with robots 2 to 10 are in Class INFRES and the one with robot 1 is in Class INCRLEN. The plot clearly



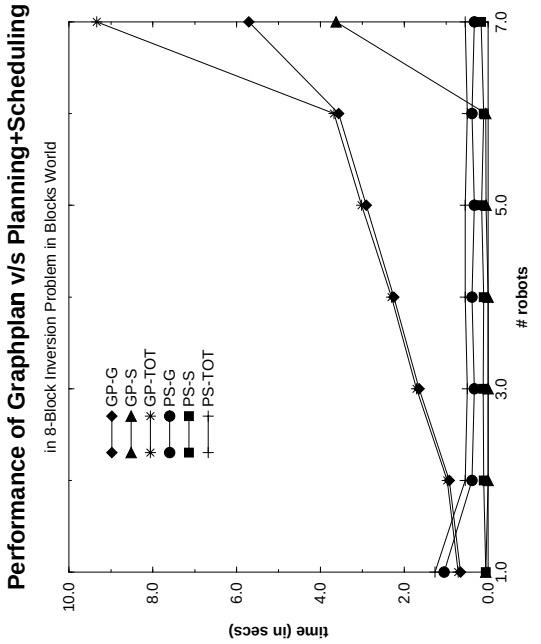
270

Figure 7.2. Comparative performance on shuffle problems of 4, 6, 8 and 10 blocks with my approach and Graphplan (Total: 80 problems).



270

Figure 7.3. Comparative performance on huge-fact (10 blocks) and bw-large-a (9 blocks) problems with my approach and Graphplan (Total: 40 problems).



270

Figure 7.4. Plot showing the performance of Graphplan alongwith Planning followed by Scheduling method for 8-block inversion problem (Total: 20 problems).

# Trucks/city	Normal GP	GP+Sched
1	1.0	2.7
2	2.4	1.0
3	4.6	1.6
4	10.0	1.5
10	500.0	1.0

Table 7.1. Runtime results from experiments in the logistics domain (in cpu sec). GP refers to Graphplan while GP+Sched refers to my approach.

riterates the above result.

In Figure 7.3, we see the performance of my method on the 10-block *huge-fact* problem and the 9-block *bw-large-a* problem. For *huge-fact*, problems with robots 1 to 5 are in Class INCRLEN and the remaining problems are in Class INFRES. Within Class INCRLEN, the search time of the original planner first increases with resources and then falls as the resource scarcity is eased. But across classes, the performance of the original planner degrades with resources. My method relies on the original planner for Class INCRLEN and thus suffers a minor penalty in those instances, but it shows remarkable improvement later on. For *bw-large-a*, problems with robots 1 to 4 are in Class INCRLEN and the remaining problems are in Class INFRES. I obtain results similar to those in the *huge-fact* case. Notice that the amount of resources at which the algorithm transitions from one class to another depends on the problem. This is why the algorithm in Figure 4.2 cycles through all the methods for each problem.

Multiple resources & the Logistics domain: Note that the procedural algorithm in Figure 4.2 and declarative method of Chapter 5 can handle domains with multi-

ple resources. Since a valid plan must be allocated resources with respect to all the resource types in the domain, the abstract plan can either be iteratively scheduled with respect to the resources of each type, as is done is procedural scheduling, or all the constraints for the different resource types can be declared together and solved simultaneously by the CSP solver. The order in which the resources should be scheduled in may be important for procedural scheduling efficiency but not correctness or optimality of the final plan, and this is reflected in the declarative scheduling methodology. To illustrate the multi-resource case, consider the results of my experiments in the logistics domain, shown in Table 7.1. The problem here involves 3 Packages at 3 cities which need to be delivered to cities other than the originating city using 3 airplanes. The number of trucks (t) at each city is varied as shown. The resource declaration makes a truck at each city equivalent to other trucks at the same city. This ensures that trucks in different cities are not considered inter-changeable. The total number of trucks (n)) in the domain is $3t$ (thus the total number of trucks in the domain in the largest problem, $t = 10$, is 30). The algorithm plans by abstracting all the trucks first. The procedural resource allocation algorithm will then solve the CSP for the three resource types one after another, each corresponding to the allocation of trucks at a specific city. In declarative scheduling, the CSP for the three resource types will be solved together. We note that separating planning from scheduling is again a very good idea in this domain too – leading to significant speedups as the number of resources (trucks/city) increases.

7.2.2 Planning and Declarative Scheduling

I have also fully implemented my declarative scheduling approach on top of Graphplan and the early results are promising. I now compare the performance of my approach to standard Graphplan, as one varies the amount of resources. Recall that the CSP encodings are solved with GAC-CBJ, a CSP solver that performs generalized arc-consistency and conflict directed backjumping used by CPLAN[34]. I consider the blocks world (where the number of robot hands is varied) and the logistics domain (where the number of trucks at different cities are varied).

Figure 7.5 shows the results for the *shuffle* problems with 4, 6, 8 and 10 blocks as the number of robots are varied from 1 to 10. The plots clearly show that planning followed by scheduling (SHUF--PS) is significantly better than original planning in the presence of resources (SHUF--TOT). The time-axis is on log scale. The plot shows that total time is relatively flat as the number of resources increase in contrast to the performance of Graphplan. Let us consider the 6-block *shuffle* problem in detail.

In my method, the causal reasoning time is constant and the resource reasoning time is dependent on the specific allocation policy (in Table 5.4) that successfully allocated the resources. For fair comparison, since Graphplan only looks for shorter length of the plan while the serializing allocation policy prefers both shorter length as well as fewer number of actions in the plan, this policy is disabled. The allocation policies are iterated in the following order: class INFRES, class FIX, class SAMELEN and finally class INCRLEN. In the *6-shuffle* case, problems with 5 to 10 robots

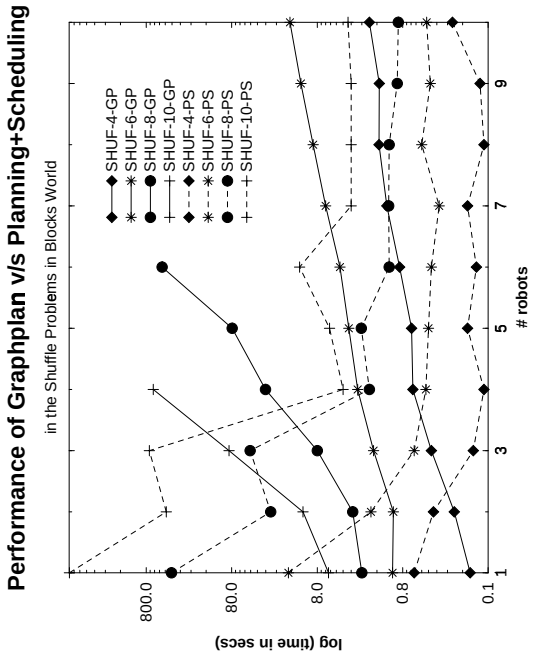
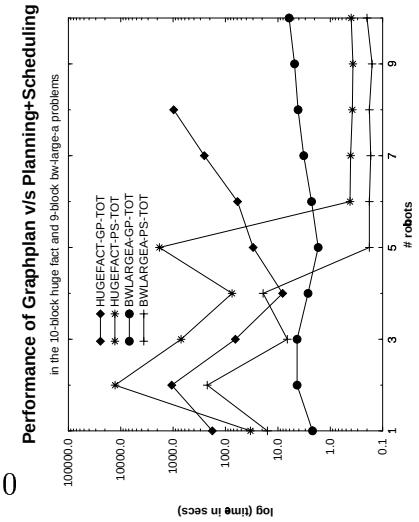


Figure 7.5. Comparative performance of my approach of decoupling causal and resource reasoning v/s Graphplan in *shuffle* problem of 4, 6, 8 and 10 blocks (Total: 80 problems).

are solved in class INFRES, problems with 3 and 4 robots are solved in class FIX, and problem with 2 robots is solved in class SAMELEN. The first class needs no modifications to the plan, the second class requires insertion of new actions, while the third also requires movement of actions across levels (steps).

In Figure 7.6, we see the performance of my method on the 10-block *huge-fact* problem and the 9-block *bw-large-a* problem. The time-axis is on log scale. For *huge-fact*, problems with robots 1 to 5 are in Class INCRLEN while for *bw-large-a*, problems with robots 1 to 4 are in Class INCRLEN. The performance of the scheduler, as noted before, is dependent on how the length of the plan is increased following resource allocation failure. In the experiments, I allow 10% of the length to



270

Figure 7.6. *Comparative performance on huge-fact (10 blocks) and bw-large-a (9 blocks) problems with Graphplan.*

be increased linearly from the current plan length before it is doubled. Across classes, we see that decoupling planning and scheduling leads to better results.

Utility of scheduling classes: The idea of progressively increasing the domain sizes of variables is very useful in practice. For example, the 10-shuffle problem with 4 robots was solved in 4 sec in class FIX while following the above order, but it took 81 minutes when class INCRLEN was specified upfront.

All *k-shuffle* problems with 1 robot can only be solved in class INCRLEN, and are handled straightforwardly, albeit with higher effort (it is reflected by the dip in the plot SHUF--PS after 1 robot case). As noted before, this is a pathological case because least commitment on resources during causal reasoning makes sense only if there are multiple resources so that any resource conflict can be *potentially* overcome during scheduling by assigning different resources to the conflicting actions. It could have been easily detected and avoided upfront.

# Trucks/city	Normal GP	GP+Sched
1	1.0	0.66
2	2.4	0.88
3	4.6	1.11
4	10.0	1.14
10	500.0	1.26

Table 7.2. Runtime results from experiments in the logistics domain (in cpu sec). GP refers to Graphplan while GP+Sched refers to my approach.

My method can also handle domains with multiple resources and sharable resources. Since a valid plan must be allocated resources with respect to all the resource types in the domain, all the constraints for the different resource types can be declared together and solved simultaneously by the CSP solver. To illustrate the multi-resource case, consider the results of my experiments in the logistics domain, shown in Table 7.2. The problem here involves 3 Packages at 3 cities which need to be delivered to cities other than the originating city using 3 airplanes. The number of trucks (t) at each city is varied as shown. The resource declaration makes a truck at each city equivalent to other trucks at the same city. This ensures that trucks in different cities are not considered inter-changeable. The total number of trucks (n)) in the domain is $3t$ (thus the total number of trucks in the domain in the largest problem, $t = 10$, is 30). We note that separating planning from scheduling is again a very good idea in this domain too – leading to significant speedups as the number of resources (trucks/city) increases.

# Rockets	Normal GP	GP+Sched	GP-CSP+Sched
2	0.13	3.05	0.48
3	0.31	2.97	0.28
4	0.15	2.99	0.31
5	0.23	2.99	0.28
6	0.40	2.96	0.30
7	0.40	2.99	0.29
8	0.55	2.98	0.31

Table 7.3. Runtime results from experiments in the rocket domain (in cpu sec). GP refers to Graphplan, GP+Sched refers to Graphplan for abstract planning followed by declarative scheduling. In GP-CSP+Sched, the planner is changed to GP-CSP.

7.2.3 The Effect of Nature of Resources on Scheduling

I now investigate the relationship between the nature of resources (sharable v/s non-sharable) and (declarative) scheduling time. I consider the *rocket* domain where the sharable rocket can be used to transport items between location. Table 7.3 shows the result of my experiments in the $rocket\_facts\_obj10$ problem in Graphplan distribution where 10 objects have to be moved from one location to another. The number of rockets are varied in each of the runs. We see that planning with Graphplan is completed in a fraction of second and it does not change much with the number of sharable resources. On the other hand, the planning time with my approach is much higher. It turns out that the causal reasoning in the space of abstracted plans takes an average of 2.38 sec (note that causal reasoning is constant for my approach) while average scheduling time is mere 0.03 sec. This suggests that either the specific planner (i.e. Graphplan) is not handling the abstracted planning problem efficiently or that

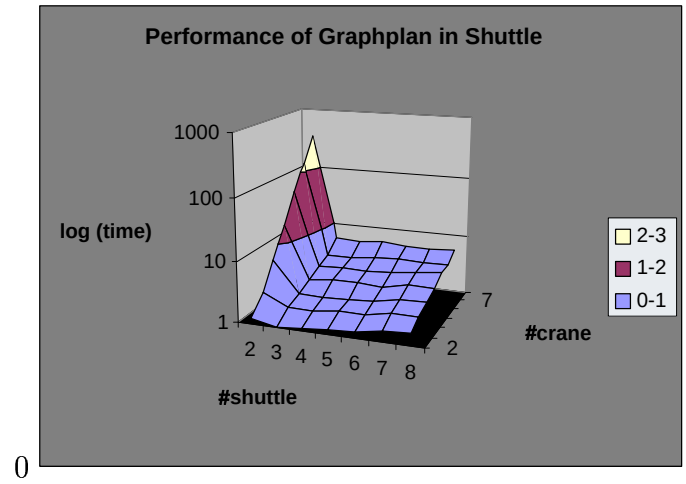
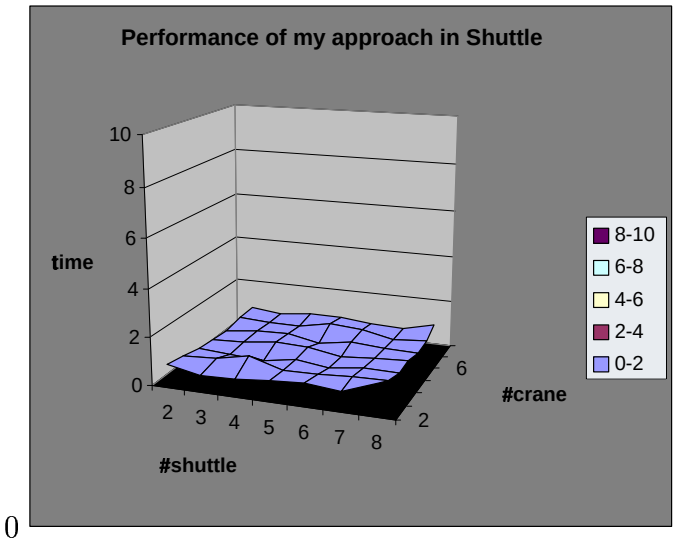


Figure 7.7. Comparative performance of Graphplan in *shuttle* problems of 2..8 cranes and 2..8 shuttles. (Total: 49 problems)

decoupling causal and resource reasoning is not beneficial for sharable resources for overall planning efficiency even though the scheduling time is still very small.

I focussed on GP-CSP[5], a planner which converts the plan graph of Graphplan into a CSP problem and solves it with a standard solver. The third column in Table 7.3 shows the result of using GP-CSP for solving the abstracted planning problem and performing scheduling thereafter. We see that the overall performance is in line with Graphplan confirming that the specific abstracted planning problem was not being solved by Graphplan efficiently while decoupling causal and resource reasoning itself does not degrade performance for sharable resources.

The scenario is highlighted if there are non-sharable resources in addition to sharable resources. To study the inter-play between these types of resources, I created a domain called the *shuttle* domain. In this domain, there are sharable shuttles and non-sharable cranes to move boxes between inter-stellar bodies (e.g. Earth and



0

Figure 7.8. Comparative performance of my approach in *shuttle* problems of 2..8 cranes and 2..8 shuttles (Total: 49 problems).

Moon). I consider problems where the number of both of these resources are varied independently. In Figure 7.7, we see the performance of Graphplan which degrades sharply with the number of non-sharable cranes and lesser so with the number of sharable shuttles. In Figure 7.8, I plot the performance of my approach. We note that run-time is fairly constant and much lesser than Graphplan with varying number of non-sharable cranes and sharable shuttles.

7.3 Lessons Learned

As we see from above discussion, decoupling causal and resource reasoning in planning is a promising idea for overall planning efficiency. The performance improvement is more marked for non-sharable resources and when planning and scheduling are *loosely*

coupled. This approach also provides us choices for selecting different reasoners for causal and resource parts. We suggested that plan and schedule quality can be used in deciding the specific reasoners for assembling an efficient planner.

Chapter 8

Discussion and related work

My work is an example of a planning model in which resource allocation is teased apart from planning, and is handled in a separate "scheduling" phase (See Figure 8.1). One may observe that a necessary condition for a schedulable plan is that it should be causally correct irrespective of the allocation of resources. Once an abstract plan (P') is produced which is correct sans the resource allocation, we can use it as a starting point for all planning problems that differ only in the number or amount of resources present. Most planners do not distinguish between these two forms of reasoning and handle them within the same planning algorithm. Indeed, the work on O-Plan [4, pp. 73], has identified the inefficiency of combining resource scheduling with planning (although, to my knowledge, no specific steps were taken to address that inefficiency in the O-Plan work).

In my approach, the planner and scheduler communicate in terms of policies that the scheduler interprets in terms of its variables, their domains and constraints.

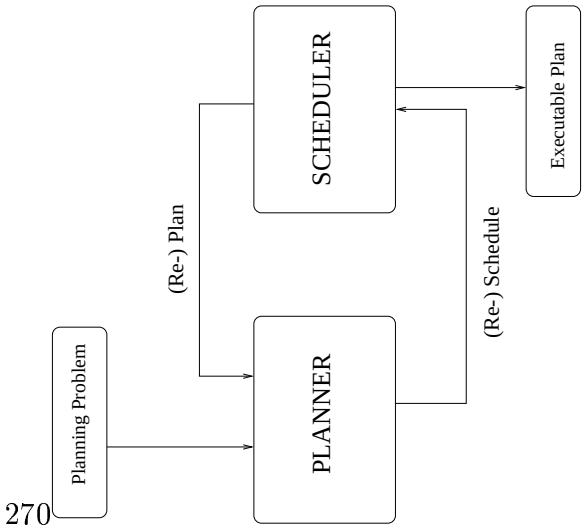


Figure 8.1. (Figure repeated for convenience) A generalized plan model for separate planning and scheduling.

The failure of the scheduler to allocate resources while following an allocation policy, informs the planner to try another policy. The partial schedule in a failed iteration is not pursued further and destroyed. This makes intuitive sense also since I am interested in *loosely coupled* scheduling instances.

Another way of looking at the planner-scheduler interaction is by having the scheduler "explain" the reason for its failure to allocate sufficient resource. This explanation, which is in terms of resource limitations as well as the plan structure, is then regressed to just the restrictions on the plan structure. The regressed explanation can then be passed on to the planner, to be used in re-starting its search. This type of "multi-module dependency directed backtracking" approach is a variation on the hybrid planning methodology developed in [12], and is also akin to the approach used to link satisfiability and linear programming solvers in [36].

My work can be seen as abstraction of resources from planning phase. From this angle, my idea of keeping the structure of the causal plan intact during resource allocation phase is akin to the enforcement of ordered monotonicity property in ALPINE[17]. An important difference however is that our work is not dependent on the availability of strong abstractions, but is rather motivated by the desire to exploit the loose-coupling between planning and scheduling in most real world domains. If the abstract plan cannot be scheduled, I support interaction between the scheduler and planner to arrive at a schedulable plan.

Among planners that have considered resources, in SIPE[35], domain-specific operator ordering can be provided by defining what are resource objects in the domain. Work more closer to mine is by El-Kholy and Richards[6] and Cesta and Cristiano[3] who perform temporal and resource reasoning after a plan is obtained. They however do not consider the interactions between resource allocation and planning phases. The recent LPSAT planner by Wolfman and Weld[36] distinguishes between discrete and continous state variables, pushing the assignment of continous ones to an LP solver. Note that discrete/continous distinction is really orthogonal to resource/non-resource distinction. Abstraction of resources can be applied to both continous and discrete parts of LPSAT. A natural extension of resource scheduling will be handling of metric constraints which is useful for real-world tasks like resource planning, temporal planning and optimization[19],[36].

Fox and Long[10] have described a way of utilizing symmetry in domains to speedup planning. Symmetric domain objects are by definition functionally similar

and cannot be usefully distinguished. The insight here is that any one of the symmetric objects is sufficient during solution verification to avoid equivalent failures. They keep track of symmetric objects during planning while I abstract out resources.

There exist methods for improving the performance of Graphplan by removing irrelevant literals from the problem specification (c.f. [30]). Such methods however are not applicable for my problem as resources – however many of them there may be – are never irrelevant and infact facilitate the state transition of desired objects.

Explanation-based learning (EBL) and dependency directed backtracking (DDB) techniques have been used by Kambhampati[11] to expedite Graphplan. Though these methods capture some of the regularities of the domain/problem, I found that they are still not competitive with my approach. Finally, the complexity of changing plans for scheduling and parallelization has also been studied by Backstrom [1]. While he focuses on parallelizing a complete and correct plan, I start with a maximally parallel resource-abstracted plan and add or shift actions across levels to handle resource constraints.

Chapter 9

Concluding Comments and Future Work

My work is motivated by the desire to exploit the loose-coupling between planning and scheduling in real world domains. I have introduced a novel planning framework in which resource allocation is teased apart from planning, and is handled in a separate "scheduling" phase. The aim is to make planning efficient and scale it to large domains containing multiple resources. I described resources and using infinite resource assumption, showed that disregarding resources during planning and subsequently scheduling resources can lead to increased performance. I provided a procedural method and a declarative method to schedule resources. In the procedural method, I discussed how resources can be allocated to an abstract plan, which is obtained after causal reasoning, on the notion of plan length optimality. In the declarative method, all the constraints of the resource allocation problem are expli-

cated and given to a CSP solver. By doing so, I can handle the resource scheduling problem in its full complexity and can provide a computational characterization of the different scheduling classes. The runtime of my approach is much less sensitive to the resource quantity. It thus admits the paradigm of *plan once and schedule anytime*. If some allocated resource becomes unavailable during plan execution, the approach can handle the exception through resource re-allocation.

In future, work on a tighter integration of planner and scheduler can be done by allowing the scheduler to suggest modifications in the abstracted plan structure. I have already used GP-CSP[5], a planner which converts the plan graph of Graphplan into a CSP problem. Using such a planner, the scheduler can inform the former about the source of infeasibility in terms of the variables and constraints in the planner's CSP. This will allow true multi-module dependency directed backtracking to handle even *tightly coupled* problems.

One can also incorporate a domain modeling tool (e.g. TIM[9]) to automatically identify freeing and unfreeing sub-plans in a domain. The user may, however, still be needed if different sub-plans have varying costs. Finally, as identified in [32], decoupling planning and scheduling can benefit not only Graphplan-style state-space planning but also goal-directed planning as in UCPOP[31], and also form the framework for planning with metric and continuous resources. We can investigate these directions in future.

My resource scheduling phase currently only aims to generate the shortest length plan – equating, in effect, the plan cost with plan length. While this is con-

sistent with the current practice in systems like Graphplan and Blackbox, real world domains would need more general cost metrics that are a function of both the plan length and resource costs. I have handled discrete sharable and non-sharable resources until now. One can incorporate continous resources by modeling linear constraints. We can concentrate on these more general issues in future.

References

[1] Backstrom, C. *Computational Aspects of Reordering Plans.* JAIR Vol.9, 99-137. 1998.

[2] Blum, A., and Furst, M. *Fast Planning through Planning Graph Analysis.* Proc. IJCAI-95, 1636-1642. 1995.

[3] Cesta, A. and Cristiano, S. *A Time and Resource Problem in Planning Architectures.* Proc. ECP-96. 1996.

[4] Currie, K. and Tate, A. *O-Plan: the Open Planning Architecture.* AI, Vol 52, 49-86. 1991.

[5] Do, B., and Kambhampati, S. *Solving Planning Graph by Compiling it into CSP.* Technical Report, Arizona State University. 1999.

[6] El-Kholy, A. and Richards, B. *Temporal and Resource Reasoning in Planning: the parcPlan Approach.* Proc. ECAI-96. 1996.

[7] Fikes, R., and Nilsson, N. STRIPS*: A New Approach to the Application of Theorem Proving to Problem Solving.* Readings in Planning. Morgan Kaufmann Publ., San Mateo, CA. 1990.

[8] Fink, E., and Yang, Q. *Formalizing Plan Justifications.* Proc. CSCSI-92, 9-14. 1992.

[9] Fox, M. and Long, D. *The Automatic Inference of State Invariants in TIM.* Journal of AI Research, Volume 9, pages 367-421. 1999.

[10] Fox, M., and Long, D. *The Detection and Exploitation of Symmetry in Planning Domains.* Proc. IJCAI-99. 1999.

[11] Kambhampati, S. *EBL and DDB for Graphplan.* Proc. IJCAI-99. 1999.

[12] S. Kambhampati, M.R. Cutkoksy, J.M. Tenenbaum and S. Lee. *Integrating General Purpose Planners and Specialized Reasoners: Case Study of a Hybrid Planning Architecture.* IEEE Trans. on Systems, Man and Cybernetics, Special issue on Planning, Scheduling and Control, Vol. 23, No. 6, November/December, 1993). (An earlier version appears in Proc. AAAI-91). 1993.

[13] Kambhampati, S.; Parker, E.; and Lambrecht, E. *Understanding and Extending Graphplan.* Proc. ECP. 1997.

[14] Kambhampati, S., and Srivastava, B. *Universal Classical Planning: An Algorithm for Unifying State Space and Plan Space Planning Approaches.* New Directions in AI Planning: EWSP 95, IOS Press. 1995.

[15] Kautz, H., and Selman, B. *BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving.* Workshop Planning as Combinatorial Search, AIPS-98, Pittsburgh, PA, 1998. 1998.

[16] Kautz, H. and Selman, B. *Pushing the Envelope: Planning, Propositional Logic and Stochastic Search.* Proc. AAAI 96. 1996.

[17] Knoblock, C. A. *Automatically Generating Abstractions for Planning.* AI Journal, 68(2). 1994.

[18] Knoblock, C. A. *Generating Parallel Execution Plans with a Partial-order Planner.* Proc. AIPS, Morgan Kaufmann Pub., San Mateo, CA. 1994.

[19] Koehler, J. *Planning under Resource Constraints.* Proc. ECAI-98. 1998.

[20] Koehler, J; Nebel, B.; Hoffmann, J.; and Dimopoulos, Y. *Extending Planning Graphs to an* ADL *Subset.* Proc. ECP-97. 1997.

[21] Laborie, P., and Ghallab, M. *Planning with Sharable Resource Constraints.* Proc. IJCAI-95. 1995.

[22] Li, C.M., and Anbulagan. *Heuristics Based on Unit Propagation for Satisfiability Problems.* Proc IJCAI-97. 1997.

[23] McAllester, D., and Rosenblitt, D. *Systematic* Nonlinear Planning. Proc. 9th NCAI-91, 634-639. 1991.

[24] McDermott, D. *AIPS-98 Planning Competition Results.* At ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html. 1998.

[25] McVey, C. B.; Atkins, E. M.; Durfee, E. H.; and Shin, K. G. *Development of Iterative Real-time Scheduler to Planner Feedback.* Proc. IJCAI. 1997.

[26] Moder, J. J., and Phillips, C. R. *Project Management with CPM and PERT.* Reinhold Publ., Chapman & Hall Ltd., London. 1964.

[27] Microsoft. *Microsoft Project Version 4.0 User Guide.* Microsoft Press. 1998. 1998.

[28] Mittal, S., and Falkenhainer, B. *Dynamic Constraint Satisfaction Problems.* Proc. AAAI-90. 1990.

[29] Muscettola, N. *Toward Real-world Science Mission Planning.* Proc. AAAI Fall Symposium. 1994.

[30] Nebel, B.; Dimopoulos, Y.; and Koehler, J. *Ignoring Irrelevant Facts and Operators in Plan Generation.* Proc. ECP-97. 1997.

[31] Penberthy, J., and Weld, D. UCPOP*: A Sound, Complete, Partial Order Planner for ADL.* Proc. AAAI-94, 103-114. 1994.

[32] Srivastava, B., and Kambhampati, S. *Scaling up Planning by Teasing out Resource Scheduling.* Proc. ECP-99. 1999.

[33] Srivastava, B. *Decoupling Causal and Resource Reasoning in Planning.* ASU TR 99-007. 1999.

[34] van Beek, P., and Chen, X. *CPlan: A Constraint Programming Approach to Planning.* Proc. AAAI-99. 1999.

[35] Wilkins, D. E. *Practical planning: Extending the Classical* AI *Planning Paradigm.* Morgan Kaufmann Pub., San Mateo, CA. 1988.

[36] Wolfman, S., and Weld, D. *The* LPSAT *Engine and its Application to Resource Planning.* Proc. IJCAI-99. 1999.

APPENDIX A

ALL PLANS FOR THE *SHUFFLE* PROBLEM

These 4 plans are among the 98,657 plans returned from graphplan for 3 robot run on *shuffle* problem. Note that they only differ in how robots are freed and unfreed.

```
Plan 1 :
  1 UNSTACK_rob1_blockF_blockE
  2 UNSTACK_rob2_blockE_blockD
  3 PUT-DOWN_rob2_blockE              free blockE
  3 UNSTACK_rob3_blockD_blockC
  4 UNSTACK_rob2_blockC_blockB
  5 PUT-DOWN_rob2_blockC
  6 STACK_rob1_blockF_blockC
  6 UNSTACK_rob2_blockB_blockA
  7 STACK_rob2_blockB_blockF
  7 PICK-UP_rob1_blockE               unfree blockE
  8 STACK_rob1_blockE_blockB
  8 PICK-UP_rob2_blockA
  9 STACK_rob2_blockA_blockE
  10 STACK_rob3_blockD_blockA


Plan 2 :
  1 UNSTACK_rob3_blockF_blockE
  2 UNSTACK_rob2_blockE_blockD
  2 PUT-DOWN_rob3_blockF              free blockF
  3 PUT-DOWN_rob2_blockE              free blockE
  3 UNSTACK_rob3_blockD_blockC
  4 PUT-DOWN_rob3_blockD              free blockD
```

```
  4 UNSTACK_rob2_blockC_blockB

  5 PUT-DOWN_rob2_blockC

  5 PICK-UP_rob3_blockF              unfree blockF

  6 STACK_rob3_blockF_blockC

  6 UNSTACK_rob2_blockB_blockA

  7 STACK_rob2_blockB_blockF

  7 PICK-UP_rob3_blockE              unfree blockE

  8 STACK_rob3_blockE_blockB

  8 PICK-UP_rob2_blockA

  9 STACK_rob2_blockA_blockE

  9 PICK-UP_rob3_blockD              unfree blockD

  10 STACK_rob3_blockD_blockA


Plan 3 :

  1 UNSTACK_rob3_blockF_blockE

  2 PUT-DOWN_rob3_blockF             free blockF

  2 UNSTACK_rob1_blockE_blockD

  3 PUT-DOWN_rob1_blockE             free blockE

  3 UNSTACK_rob3_blockD_blockC

  4 PUT-DOWN_rob3_blockD             free blockD

  4 UNSTACK_rob1_blockC_blockB

  5 PICK-UP_rob3_blockF              unfree blockF

  5 PUT-DOWN_rob1_blockC

  6 UNSTACK_rob1_blockB_blockA

  6 STACK_rob3_blockF_blockC

  7 PICK-UP_rob3_blockE              unfree blockE
```

```
  7 STACK_rob1_blockB_blockF

  8 PICK-UP_rob1_blockA

  8 STACK_rob3_blockE_blockB

  9 STACK_rob1_blockA_blockE

  9 PICK-UP_rob3_blockD              unfree blockD

  10 STACK_rob3_blockD_blockA


Plan 4 :

  1 UNSTACK_rob2_blockF_blockE

  2 PUT-DOWN_rob2_blockF             free blockF

  2 UNSTACK_rob1_blockE_blockD

  3 UNSTACK_rob2_blockD_blockC

  3 PUT-DOWN_rob1_blockE             free blockE

  4 UNSTACK_rob1_blockC_blockB

  4 PUT-DOWN_rob2_blockD             free blockD

  5 PICK-UP_rob2_blockF              unfree blockF

  5 PUT-DOWN_rob1_blockC

  6 UNSTACK_rob1_blockB_blockA

  6 STACK_rob2_blockF_blockC

  7 PICK-UP_rob2_blockE              unfree blockE

  7 STACK_rob1_blockB_blockF

  8 PICK-UP_rob1_blockA

  8 STACK_rob2_blockE_blockB

  9 STACK_rob1_blockA_blockE

  9 PICK-UP_rob2_blockD              unfree blockD

  10 STACK_rob2_blockD_blockA
```

APPENDIX B

EXPLICIT RESOURCE SPECIFICATION

The format to specify resources in the blocks world domain.

```
# robot
(resource ROBOT
 (free
   (means
     (effects (arm-empty <rob>))
       (params  (<rob> ROBOT)
                (<ob> OBJECT)
                (<underob> OBJECT))
        (plans
           (p1  1
             (s1  (PUT-DOWN  <rob> <ob>)))
           (p2  4
             (s1  (STACK  <rob> <ob>
                       <underob>)))))))
 (unfree
   (means
     (effects (holding <rob> <ob>))
       (params  (<rob> ROBOT)
                (<ob> OBJECT)
                (<underob> OBJECT))
        (plans
           (p1  1
             (s1  (PICK-UP  <rob> <ob>)))
           (p2  4
             (s1  (UNSTACK  <rob> <ob>
```

```
                    <underob>)))))))
```

The similar format in logistics domain will be:

```
# truck
(resource TRUCK
  (free
    (means
      (effects (at <truck> <loc-to>))
      (params  (<truck> TRUCK)
               (<loc-from> LOCATION)
               (<loc-to> LOCATION)
               (<city> CITY))
       (plans
          (p1  1
            (s1  (DRIVE-TRUCK <truck>
                         <loc-from>
                         <loc-to>
                         <CITY>)))))))
  (unfree
    (means
      (effects (at <truck> <loc-from>))
      (params  (<truck> TRUCK)
               (<loc-from> LOCATION)
               (<loc-to> LOCATION)
               (<city> CITY))
       (plans
          (p1  1
```

```
            (s1  (DRIVE-TRUCK <truck>
                              <loc-from>
                              <loc-to>
                              <CITY>)))))))


#airplane
(resource AIRPLANE
  (free
    (means
      (effects (at <airplane> <loc-to>))
      (params  (<airplane> AIRPLANE)
               (<loc-from> AIRPORT)
               (<loc-to> AIRPORT))
       (plans
          (p1  1
            (s1  (FLY-AIRPLANE <airplane>
                               <loc-from>
                               <loc-to>))))))
  (unfree
    (means
      (effects (at <airplane> <loc-from>))
      (params  (<airplane> AIRPLANE)
               (<loc-from> AIRPORT)
               (<loc-to> AIRPORT))
       (plans
          (p1  1
```

```
(s1  (FLY-AIRPLANE <airplane>
                   <loc-from>
                   <loc-to>)))))))
```

APPENDIX C

DOMAINS AND PROBLEM SPECIFICATION

The domains and problem instances mentioned in the dissertation are listed here. Problem sets are obtained by varying the number of resources in a problem instance.

C.1 Blocks world domain

C.1.1 Domain operator file

```
#old-style
(resource ROBOT
 (free
   (means
     (effects (arm-empty <rob>))
       (params  (<rob> ROBOT)
                (<ob> OBJECT)
                (<underob> OBJECT))
        (plans
```

```
          (p1  1
            (s1  (PUT-DOWN  <rob> <ob>)))
          (p2  4
            (s1  (STACK  <rob> <ob>
                      <underob>))))))
 (unfree
   (means
     (effects (holding <rob> <ob>))
       (params  (<rob> ROBOT)
                (<ob> OBJECT)
                (<underob> OBJECT))
        (plans
           (p1  1
             (s1  (PICK-UP  <rob> <ob>)))
           (p2  4
             (s1  (UNSTACK  <rob> <ob>
                      <underob>)))))))

(operator
  PICK-UP
  (params (<rob> ROBOT) (<ob> OBJECT))
  (preconds
(clear <ob>) (on-table <ob>)
        (arm-empty <rob>))
  (effects
(holding <rob> <ob>)))

(operator
  PUT-DOWN
  (params (<rob> ROBOT) (<ob> OBJECT))
  (preconds
(holding <rob> <ob>))
  (effects
(clear <ob>) (arm-empty <rob>)
        (on-table <ob>)))

(operator
 STACK
 (params (<rob> ROBOT) (<ob> OBJECT)
         (<underob> OBJECT))
 (preconds
       (clear <underob>)
       (holding <rob> <ob>))
 (effects
       (arm-empty <rob>) (clear <ob>)
```

```
      (on <ob> <underob>)))

(operator
 UNSTACK
 (params (<rob> ROBOT) (<ob> OBJECT)
         (<underob> OBJECT))
 (preconds
       (on <ob> <underob>) (clear <ob>)
       (arm-empty <rob>))
 (effects
       (holding <rob> <ob>)
       (clear <underob>)))
```

C.1.2 *shuffle* problem with 3 robots

```
(blockA OBJECT)
(blockB OBJECT)
(blockC OBJECT)
(blockD OBJECT)
(blockE OBJECT)
(blockF OBJECT)
(rob1 ROBOT)
(rob2 ROBOT)
(rob3 ROBOT)

(preconds
(on-table blockA)
(on blockB blockA)
(on blockC blockB)
(on blockD blockC)
(on blockE blockD)
(on blockF blockE)
(clear blockF)
(arm-empty rob1)
(arm-empty rob2)
(arm-empty rob3))

(effects
(on blockD blockA)
(on blockA blockE)
(on blockE blockB)
(on blockB blockF)
(on blockF blockC))
```

C.1.3 Hugefact problem with 3 robots

```
(blockA OBJECT)
(blockB OBJECT)
(blockC OBJECT)
(blockD OBJECT)
(blockE OBJECT)
(blockF OBJECT)
(blockG OBJECT)
(blockH OBJECT)
(blockI OBJECT)
(blockJ OBJECT)
(rob8 ROBOT)
(rob9 ROBOT)
(rob10 ROBOT)

(preconds
(on-table blockA)
(on blockB blockA)
(on blockC blockB)
(on-table blockD)
(on blockE blockD)
(on blockF blockE)
(on-table blockG)
(on blockH blockG)
(on blockI blockH)
        (clear blockC)
        (clear blockJ)
        (clear blockF)
(on blockJ blockI)
        (arm-empty rob8)
        (arm-empty rob9)
        (arm-empty rob10))

(effects
(on-table blockA)
(on blockD blockA)
(on blockG blockD)
(on-table blockJ)
(on blockF blockJ)
(on blockI blockC)
(on-table blockC)
(on blockH blockE)
(on blockE blockB)
(on-table blockB))
```

C.1.4 BW-large-a problem with 3 robots

```
(blockA OBJECT)
(blockB OBJECT)
(blockC OBJECT)
(blockD OBJECT)
(blockE OBJECT)
(blockF OBJECT)
(blockG OBJECT)
(blockH OBJECT)
(blockI OBJECT)
(rob8 ROBOT)
(rob9 ROBOT)
(rob10 ROBOT)

(preconds
        (arm-empty rob8)
        (arm-empty rob9)
        (arm-empty rob10)
(on blockC blockB)
(on blockB blockA)
(on-table blockA)
(on blockE blockD)
(on-table blockD)
(on blockI blockH)
(on blockH blockG)
(on blockG blockF)
(on-table blockF)
        (clear blockC)
        (clear blockE)
        (clear blockI))

(effects
(on blockA blockE)
(on-table blockE)
(on blockH blockI)
(on blockI blockD)
(on-table blockD)
(on blockB blockC)
(on blockC blockG)
(on blockG blockF)
(on-table blockF)
```

```
        (clear blockA)
        (clear blockH)
        (clear blockB))
```

C.2 Logistics domain

C.2.1 Domain operator file

```
#old-style
(resource la-TRUCK shares
  (free
    (means
      (effects (at <la-truck> <loc-to>))
      (params  (<la-truck> la-TRUCK)
               (<loc-from> LOCATION)
               (<loc-to> LOCATION)
               (<city> CITY))
       (plans
          (p1  1
            (s1  (la-DRIVE-TRUCK <la-truck>
                         <loc-from>
                         <loc-to>
                         <city>))))))
  (unfree
    (means
      (effects (at <la-truck> <loc-from>))
      (params  (<la-truck> la-TRUCK)
               (<loc-from> LOCATION)
               (<loc-to> LOCATION)
               (<city> CITY))
       (plans
          (p1  1
            (s1  (la-DRIVE-TRUCK <la-truck>
                         <loc-from>
                         <loc-to>
                         <city>)))))))


(resource bos-TRUCK shares
  (free
    (means
      (effects (at <bos-truck> <loc-to>))
```

```
      (params  (<bos-truck> bos-TRUCK)
               (<loc-from> LOCATION)
               (<loc-to> LOCATION)
               (<city> CITY))
       (plans
          (p1  1
            (s1  (bos-DRIVE-TRUCK <bos-truck>
                          <loc-from>
                          <loc-to>
                          <city>))))))
  (unfree
    (means
      (effects (at <bos-truck> <loc-from>))
      (params  (<bos-truck> bos-TRUCK)
               (<loc-from> LOCATION)
               (<loc-to> LOCATION)
               (<city> CITY))
       (plans
          (p1  1
            (s1  (bos-DRIVE-TRUCK <bos-truck>
                          <loc-from>
                          <loc-to>
                          <city>)))))))

(resource pgh-TRUCK shares
  (free
    (means
      (effects (at <pgh-truck> <loc-to>))
      (params  (<pgh-truck> pgh-TRUCK)
               (<loc-from> LOCATION)
               (<loc-to> LOCATION)
               (<city> CITY))
       (plans
          (p1  1
            (s1  (pgh-DRIVE-TRUCK <pgh-truck>
                          <loc-from>
                          <loc-to>
                          <city>))))))
  (unfree
    (means
      (effects (at <pgh-truck> <loc-from>))
      (params  (<pgh-truck> pgh-TRUCK)
               (<loc-from> LOCATION)
               (<loc-to> LOCATION)
               (<city> CITY))
```

```
      (plans
         (p1  1
           (s1  (pgh-DRIVE-TRUCK <pgh-truck>
                        <loc-from>
                        <loc-to>
                        <city>)))))))


(operator la-LOAD-TRUCK
  (params (<obj> OBJECT)
          (<la-truck> la-TRUCK)
          (<loc> LOCATION)
          (<city> CITY))
  (preconds
        (at <la-truck> <loc>)
        (at <obj> <loc>)
        (loc-at <loc> <city>))
  (effects
        (la-in <obj> <la-truck>)
        (at <la-truck> <loc>)
        (loc-at <loc> <city>)))


(operator bos-LOAD-TRUCK
  (params (<obj> OBJECT)
          (<bos-truck> bos-TRUCK)
          (<loc> LOCATION)
          (<city> CITY))
  (preconds
        (at <bos-truck> <loc>)
        (at <obj> <loc>)
        (loc-at <loc> <city>))
  (effects
        (bos-in <obj> <bos-truck>)
        (at <bos-truck> <loc>)
        (loc-at <loc> <city>)))


(operator pgh-LOAD-TRUCK
  (params (<obj> OBJECT)
          (<pgh-truck> pgh-TRUCK)
          (<loc> LOCATION)
          (<city> CITY))
  (preconds
        (at <pgh-truck> <loc>)
        (at <obj> <loc>)
        (loc-at <loc> <city>))
```

```
  (effects
        (pgh-in <obj> <pgh-truck>)
        (at <pgh-truck> <loc>)
        (loc-at <loc> <city>)))

(operator LOAD-AIRPLANE
  (params (<obj> OBJECT)
          (<airplane> AIRPLANE)
          (<loc> LOCATION))
  (preconds
        (at <obj> <loc>)
        (at <airplane> <loc>))
  (effects
        (in <obj> <airplane>)
        (at <airplane> <loc>)))

(operator la-UNLOAD-TRUCK
  (params (<obj> OBJECT)
          (<la-truck> la-TRUCK)
          (<loc> LOCATION)
          (<city>  CITY))
  (preconds
        (at <la-truck> <loc>)
        (la-in <obj> <la-truck>)
        (loc-at <loc> <city>))
  (effects
        (at <obj> <loc>)
        (at <la-truck> <loc>)
        (loc-at <loc> <city>)))

(operator bos-UNLOAD-TRUCK
  (params (<obj> OBJECT)
          (<bos-truck> bos-TRUCK)
          (<loc> LOCATION)
          (<city>  CITY))
  (preconds
        (at <bos-truck> <loc>)
        (bos-in <obj> <bos-truck>)
        (loc-at <loc> <city>))
  (effects
        (at <obj> <loc>)
        (at <bos-truck> <loc>)
        (loc-at <loc> <city>)))

(operator pgh-UNLOAD-TRUCK
```

```
  (params (<obj> OBJECT)
          (<pgh-truck> pgh-TRUCK)
          (<loc> LOCATION)
          (<city>  CITY))
  (preconds
        (at <pgh-truck> <loc>)
        (pgh-in <obj> <pgh-truck>)
        (loc-at <loc> <city>))
  (effects
        (at <obj> <loc>)
        (at <pgh-truck> <loc>)
        (loc-at <loc> <city>)))

(operator UNLOAD-AIRPLANE
  (params (<obj> OBJECT)
          (<airplane> AIRPLANE)
          (<loc> LOCATION))
  (preconds
        (in <obj> <airplane>)
        (at <airplane> <loc>))
  (effects
        (at <obj> <loc>)
        (at <airplane> <loc>)))

(operator la-DRIVE-TRUCK
  (params (<la-truck> la-TRUCK)
          (<loc-from> LOCATION)
          (<loc-to> LOCATION)
          (<city> CITY))
  (preconds
        (at <la-truck> <loc-from>)
        (loc-at <loc-from> <city>)
        (loc-at <loc-to> <city>))
  (effects
        (at <la-truck> <loc-to>)
        (loc-at <loc-from> <city>)
        (loc-at <loc-to> <city>)))

(operator bos-DRIVE-TRUCK
  (params (<bos-truck> bos-TRUCK)
          (<loc-from> LOCATION)
          (<loc-to> LOCATION)
          (<city> CITY))
  (preconds
        (at <bos-truck> <loc-from>)
```

```
        (loc-at <loc-from> <city>)
        (loc-at <loc-to> <city>))
  (effects
        (at <bos-truck> <loc-to>)
        (loc-at <loc-from> <city>)
        (loc-at <loc-to> <city>)))

(operator pgh-DRIVE-TRUCK
  (params (<pgh-truck> pgh-TRUCK)
          (<loc-from> LOCATION)
          (<loc-to> LOCATION)
          (<city> CITY))
  (preconds
        (at <pgh-truck> <loc-from>)
        (loc-at <loc-from> <city>)
        (loc-at <loc-to> <city>))
  (effects
        (at <pgh-truck> <loc-to>)
        (loc-at <loc-from> <city>)
        (loc-at <loc-to> <city>)))


(operator FLY-AIRPLANE
  (params (<airplane> AIRPLANE)
          (<loc-from> AIRPORT)
          (<loc-to> AIRPORT))
  (preconds
        (at <airplane> <loc-from>))
  (effects
        (at <airplane> <loc-to>)))
```

C.2.2 Logistics problem

```
(package1 OBJECT)
(package2 OBJECT)
(package3 OBJECT)
(pgh-truck1 pgh-TRUCK)
(pgh-truck2 pgh-TRUCK)
(pgh-truck3 pgh-TRUCK)
(bos-truck1 bos-TRUCK)
(bos-truck2 bos-TRUCK)
(bos-truck3 bos-TRUCK)
(la-truck1 la-TRUCK)
(la-truck2 la-TRUCK)
```

```
(la-truck3 la-TRUCK)
(airplane1 AIRPLANE)
(airplane2 AIRPLANE)
(bos-po LOCATION)
(la-po LOCATION)
(pgh-po LOCATION)
(bos-airport AIRPORT)
(bos-airport LOCATION)
(pgh-airport AIRPORT)
(pgh-airport LOCATION)
(la-airport AIRPORT)
(la-airport LOCATION)
(pgh CITY)
(bos CITY)
(la CITY)

(preconds
 (at package1 pgh-po)
 (at package2 bos-po)
 (at package3 la-po)
 (at airplane1 pgh-airport)
 (at airplane2 pgh-airport)
 (at bos-truck1 bos-po)
 (at bos-truck2 bos-po)
 (at bos-truck3 bos-po)
 (at pgh-truck1 pgh-po)
 (at pgh-truck2 pgh-po)
 (at pgh-truck3 pgh-po)
 (at la-truck1 la-po)
 (at la-truck2 la-po)
 (at la-truck3 la-po)
 (loc-at pgh-po pgh)
 (loc-at pgh-airport pgh)
 (loc-at bos-po bos)
 (loc-at bos-airport bos)
 (loc-at la-po la)
 (loc-at la-airport la))

(effects
 (at package1 bos-po)
 (at package2 la-po)
 (at package3 bos-po))
```

C.3 Shuttle domain

C.3.1 Domain operator file

```
#old-style
(resource CRANE
  (free
    (means
      (effects (arm-empty <crane>))
      (params  (<crane> CRANE)
               (<ob> OBJECT)
               (<underob> OBJECT))
       (plans
          (p1  1
            (s1  (PUT-DOWN <crane> <ob>)))
          (p2  4
            (s1  (STACK <crane> <ob>
                        <underob>))))))
    (unfree
      (means
(effects (holding <crane> <ob>))
        (params  (<crane> CRANE)
                 (<ob> OBJECT)
                 (<underob> OBJECT))
         (plans
             (p1  1
               (s1  (PICK-UP <crane> <ob>)))
             (p2  4
               (s1  (UNSTACK <crane> <ob>
                             <underob>)))))))


(resource SHUTTLE shares nofree
 (free
   (means
     (effects (at <ob> <place>))
       (params  (<ob> OBJECT)
                (<shuttle> SHUTTLE)
                (<place> PLACE))
       (plans
          (p1  1
            (s1  (UNLOAD <ob> <shuttle>
                         <place>))))))
```

```
 (unfree
   (means
     (effects (in <ob> <place>))
       (params  (<ob> OBJECT)
                (<shuttle> SHUTTLE)
                (<place> PLACE))
       (plans
          (p1  1
            (s1  (LOAD <ob> <shuttle>
                        <place>)))))))

(operator
 PICK-UP
 (params (<crane> CRANE) (<ob> OBJECT))
 (preconds
       (clear <ob>) (on-ground <ob>)
       (arm-empty <crane>))
 (effects
       (holding <crane> <ob>)))

(operator
 PICK-FROM-CONVEYOR
 (params (<crane> CRANE) (<ob> OBJECT)
         (<place> PLACE))
 (preconds
       (clear <ob>)
       (on-conveyor <ob> <place>)
       (arm-empty <crane>))
 (effects
       (holding <crane> <ob>)
       (at <ob> <place>)))

(operator
 PUT-DOWN
 (params (<crane> CRANE) (<ob> OBJECT))
 (preconds
       (holding <crane> <ob>))
 (effects
       (clear <ob>) (arm-empty <crane>)
       (on-ground <ob>)))

(operator
 PUT-ON-CONVEYOR
 (params (<crane> CRANE) (<ob> OBJECT)
         (<place> PLACE))
```

```
 (preconds
       (holding <crane> <ob>)
       (at <ob> <place>))
 (effects
       (clear <ob>) (arm-empty <crane>)
       (on-conveyor <ob> <place>)))

(operator
 STACK
 (params (<crane> CRANE) (<ob> OBJECT)
         (<underob> OBJECT))
 (preconds
       (clear <underob>)
       (holding <crane> <ob>))
 (effects
       (arm-empty <crane>) (clear <ob>)
       (on <ob> <underob>)))

(operator
 UNSTACK
 (params (<crane> CRANE) (<ob> OBJECT)
         (<underob> OBJECT))
 (preconds
       (on <ob> <underob>) (clear <ob>)
       (arm-empty <crane>))
 (effects
       (holding <crane> <ob>)
       (clear <underob>)))

(operator
 LOAD-IN-SHUTTLE
 (params (<ob> OBJECT) (<shuttle> SHUTTLE)
         (<place> PLACE))
 (preconds
       (at <shuttle> <place>)
       (on-conveyor <ob> <place>)
       (clear <ob>))
 (effects
       (in <ob> <shuttle>)
       (at <shuttle> <place>)))

(operator
 UNLOAD-FROM-SHUTTLE
 (params (<ob> OBJECT) (<shuttle> SHUTTLE)
         (<place>  PLACE))
```

```
 (preconds
       (at <shuttle> <place>)
       (in <ob> <shuttle>))
 (effects
       (on-conveyor <ob> <place>)
       (at <shuttle> <place>)
       (clear <ob>)))

(operator
 MOVE
 (params (<shuttle> SHUTTLE) (<from> PLACE)
         (<to> PLACE))
 (preconds
        (has-fuel <shuttle>)
        (at <shuttle> <from>))
 (effects
        (at <shuttle> <to>)))
```

C.3.2 Shuttle problem with 4 cranes and 6 shuttles

```
(boxA OBJECT)
(boxB OBJECT)
(boxC OBJECT)
(boxD OBJECT)
(crane1 CRANE)
(crane2 CRANE)
(crane3 CRANE)
(crane4 CRANE)
(Earth PLACE)
(Moon PLACE)
(shuttle1 SHUTTLE)
(shuttle2 SHUTTLE)
(shuttle3 SHUTTLE)
(shuttle4 SHUTTLE)
(shuttle5 SHUTTLE)
(shuttle6 SHUTTLE)

(preconds
        (at shuttle1 Earth)
        (at shuttle2 Earth)
        (at shuttle3 Earth)
        (at shuttle4 Earth)
        (at shuttle5 Earth)
```

```
        (at shuttle6 Earth)
        (has-fuel shuttle1)
        (has-fuel shuttle2)
        (has-fuel shuttle3)
        (has-fuel shuttle4)
        (has-fuel shuttle5)
        (has-fuel shuttle6)
        (at boxA Earth)
        (at boxB Earth)
        (at boxC Earth)
        (at boxD Earth)
(on-ground boxA)
(on boxB boxA)
(on boxC boxB)
(on boxD boxC)
(clear boxD)
(arm-empty crane1)
(arm-empty crane2)
(arm-empty crane3)
(arm-empty crane4))

(effects
        (at boxA Moon)
        (at boxB Moon)
        (at boxC Moon)
        (at boxD Moon)
(on boxD boxA)
(on boxA boxB)
(on boxB boxC))
```

C.4 Rocket domain

C.4.1 Domain operator file (PRODIGY style)

```
(resource ROCKET shares nofree
 (free
   (means
     (effects (at <object> <place>))
       (params  (<object> CARGO)
                (<rocket> ROCKET)
                (<place> PLACE))
```

```
       (plans
          (p1  1
            (s1  (UNLOAD <object> <rocket>
                         <place>))))))
 (unfree
   (means
     (effects (in <object> <place>))
       (params  (<object> CARGO)
                (<rocket> ROCKET)
                (<place> PLACE))
        (plans
           (p1  1
             (s1  (LOAD <object> <rocket>
                        <place>)))))))

(operator
 LOAD
 (params (<object> CARGO)
         (<rocket> ROCKET)
         (<place> PLACE))
 (preconds
       (at <rocket> <place>)
       (at <object> <place>))
 (effects
       (in <object> <rocket>)
       (del at <object> <place>)))

(operator
 UNLOAD
 (params (<object> CARGO)
         (<rocket> ROCKET)
         (<place>  PLACE))
 (preconds
       (at <rocket> <place>)
       (in <object> <rocket>))
 (effects
       (at <object> <place>)
       (del in <object> <rocket>)))

(operator
 MOVE
 (params (<rocket> ROCKET)
         (<from> PLACE) (<to> PLACE))
 (preconds
       (has-fuel <rocket>)
```

```
      (at <rocket> <from>))
 (effects
      (at <rocket> <to>)
      (del has-fuel <rocket>)
      (del at <rocket> <from>)))
```

C.4.2 Rocket problem with 3 rockets

```
(London PLACE)
(Paris PLACE)
(JFK PLACE)
(OHARE PLACE)
(r1 ROCKET)
(r2 ROCKET)
(r3 ROCKET)
(mxf CARGO)
(avrim CARGO)
(alex CARGO)
(jason CARGO)
(pencil CARGO)
(paper CARGO)
(april CARGO)
(michelle CARGO)
(betty CARGO)
(lisa CARGO)

(preconds
(has-fuel r1)
(has-fuel r2)
(has-fuel r3)
(at r1 London)
(at r2 OHARE)
(at r3 London)
(at mxf    London)
(at avrim  London)
(at alex   London)
(at jason  London)
(at pencil London)
(at paper    OHARE)
(at michelle OHARE)
(at april    OHARE)
(at betty    OHARE)
(at lisa     OHARE))
```

```
(effects
(at mxf    Paris)
(at avrim  Paris)
(at alex   Paris)
(at jason  Paris)
(at pencil Paris)

(at paper    JFK)
(at michelle JFK)
(at april    JFK)
(at betty    JFK)
(at lisa     JFK))
```

C.5 Gripper domain

C.5.1 Domain operator file

```
#old-style
(resource GRIPPER
 (free
  (means
    (effects (free <grip>))
      (params  (<grip> GRIPPER)
               (<ob> OBJECT)
               (<room> ROOM))
       (plans
         (p1  1
           (s1  (DROP <grip> <ob>
                      <room>))))))
 (unfree
   (means
    (effects (carry <grip> <ob>))
      (params  (<grip> GRIPPER)
               (<ob> OBJECT)
               (<room> ROOM)
       (<from-room> ROOM)
               (<to-room> ROOM))
       (plans
        (p1  1
          (s1  (MOVE <to-room>
```

```
                    <from-room>))
          (s2  (PICK <grip> <ob>
                    <room>))
          (s3  (MOVE <from-room>
                    <to-room>)))))))

(operator
  PICK
  (params (<grip> GRIPPER)
          (<ob> OBJECT)
          (<room> ROOM))
  (preconds
(at <ob> <room>)
        (at-robby <room>)
        (free <grip>))
  (effects
(carry <grip> <ob>)
        (at-robby <room>)))

(operator
  DROP
  (params (<grip> GRIPPER)
          (<ob> OBJECT)
          (<room> ROOM))
  (preconds
(carry <grip> <ob>)
        (at-robby <room>))
  (effects
(free <grip>) (at <ob> <room>)
        (at-robby <room>)))

(operator
 MOVE
 (params (<from-room> ROOM)
         (<to-room> ROOM))
 (preconds
(at-robby <from-room>))
 (effects
       (at-robby <to-room>)))
```

C.5.2 Gripper problem with 2 grips

```
(roomA ROOM)
```

```
(roomB ROOM)
(ball1 OBJECT)
(ball2 OBJECT)
(ball3 OBJECT)
(ball4 OBJECT)
(left  GRIPPER)
(right GRIPPER)

(preconds
(at-robby roomA)
(at ball1 roomA)
(at ball2 roomA)
(at ball3 roomA)
(at ball4 roomA)
(free left)
(free right))

(effects
(at ball1 roomB)
(at ball2 roomB)
(at ball3 roomB)
(at ball4 roomB))
```