Planning the Project Management Way: Efficient Planning by Effective Integration of Causal and Resource Reasoning in `RealPlan`

Biplav Srivastava, Subbarao Kambhampati, and Minh B. Do

Email: {biplav,rao,binhminh}@asu.edu
Department of Computer Science and Engineering,
Arizona State University, Tempe, AZ 85287-5406.
http://rakaposhi.eas.asu.edu/yochan.html

Abstract

In most real-world reasoning problems, planning and scheduling phases are loosely coupled. For example, in project planning, the user comes up with a task list and schedules it with a scheduling tool like Microsoft Project. One can view automated planning in a similar way in which there is 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. On the other hand, most existing automated planners studied in Artificial Intelligence 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 state of the art planners which can be overcome by leveraging the loose coupling.

Specifically, a novel planning framework called `RealPlan` is developed in which resource allocation is de-coupled from planning and is handled in a separate scheduling phase. The scheduling problem with discrete resources is represented as a Constraint Satisfaction Problem (CSP) problem, and the planner and scheduler interact either in a master-slave manner or in a peer-peer relationship. In the former, the scheduler simply tries to assign resources to the abstract causal plan passed to it by the planner and returns success. In the latter, a more sophisticated "multi-module dependency directed backtracking" approach is used where the failure explanation in the scheduler is translated back to the planner and serves as a *nogood* to direct planner search.

`RealPlan` not only preserves both the correctness as well as the quality (measured in length) of the plan but also improves efficiency. Moreover, the failure-driven learning of constraints can serve as an elegant and effective approach for integrating planning and scheduling systems. Beyond the context of planner efficiency, the current work can be viewed as an important step towards merging planning with real world problem solving where plan failure during execution can be resolved by undertaking only necessary resource re-allocation and not complete re-planning.

1 Introduction

Planning is comprised 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, 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 (i.e., planning as studied in Artificial Intelligence) can handle small plans compared to what humans already handle in the real world. In real-world problems, planning and scheduling phases are usually *loosely coupled*. The top management of a company (See Figure 1) may give strategic directions to their technology development organization which in turn develops feasible proposals. The proposals are detailed by mid to lower level managers into a manageable project for budget, approval and execution purposes.

Planning and monitoring of large-scale projects is done with a project management tool like Microsoft Project[40] by using an activity network for both planning and control. Humans come up with the Work Breakdown Structure (WBS)[39] 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-aligned to optimize on resource usage. In the Critical Path Method (CPM), activity times are assumed to be known or predictable (deterministic) while in the Program Evaluation and Review Technique (PERT), activity times are assumed to be random, with known probability distribution (probabilistic). But

[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.

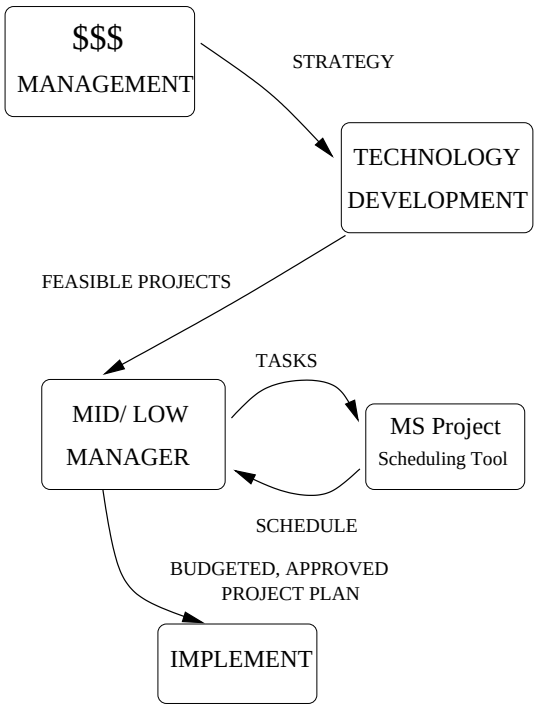

Fig. 1. Project planning in real world.

the nature of the WBS (i.e., causal plan) remains relevant irrespective of how the activities (actions) are modeled.

Most implemented AI 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 handled by logic-based planners like UCPOP[45] and Graphplan [6] in the same way as other objects in the domain. Experimental results show (in Section 1.1) that this strategy severely curtails the scale-up potential of existing planners, including such recent ones as Graphplan, Blackbox [27], FF [19], and AltAlt [58] (see Figure 2). In particular, these planners exhibit the seemingly irrational behavior of worsening in performance with increased resources. For continuous resources like time and fuel, planning systems have additionally employed time/resource map managers to ensure resource consistency (SIPE[56], IxTeT[33], IPP[32], LPSAT[57]). However, the integration of resources explodes the search space of the planner beyond the 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 the logical goals. To handle search, IPP restricts expressivity by avoiding explicit temporal modeling while other planners suffer a performance drop with slower flaw resolution.

Research Objective: The focus of this paper is on the role of resources in a planning domain, and how resource-based and causality-based reasoning can be decoupled in planning so that they are effectively integrated for planning

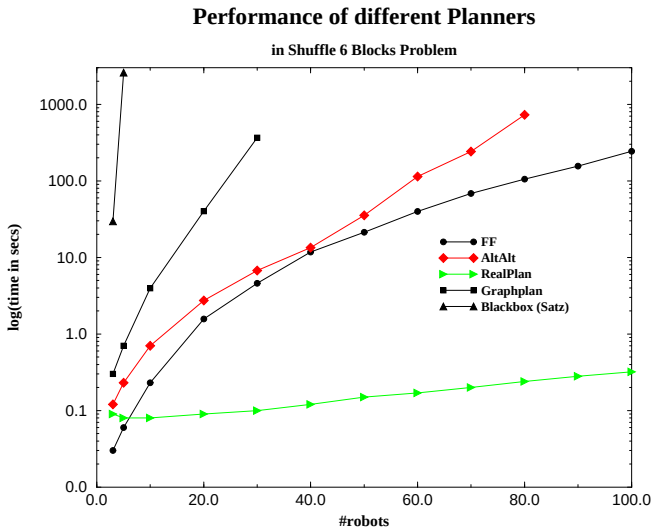

Fig. 2. Performance of Graphplan, Blackbox, FF, and AltAlt in the 6-block Shuffle problem with varying number of robots. Notice that all those planners show the counter-intuitive behavior of worsening with additional resources. The plot also shows that RealPlan scales gracefully.

efficiency. A major motivation is to scale planning algorithms to large, realistic domains where addition of more resources can reduce the planning time and not increase it.

Our contribution is a novel loosely-coupled planning approach, called `RealPlan` (explained later in Section 1.2), where causal-reasoning (planning) is used to generate an abstract plan ignoring all resource conflicts [52],[53], [54]. The abstract plan is then post-processed to allocate the required resources without altering the causal structure of the plan. As mentioned earlier, 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. A similar flow is proposed here to exploit the loose coupling – except that both planning and scheduling phases will be automated.

1.1 An Empirical Motivation

In this section, we show that the treatment of resources in all of the recent state-of-the-art planners is not scalable. We consider Graphplan-based planners, heuristic state space planners and partial-order planners.

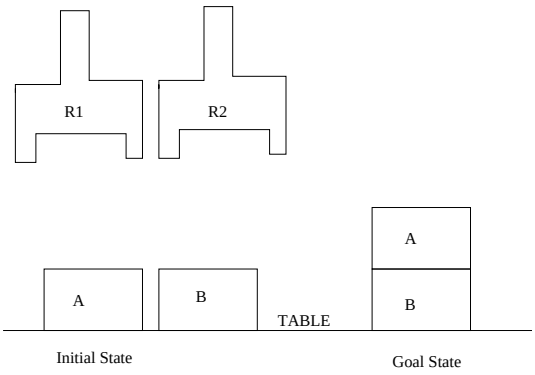

Fig. 3. A simple planning problem in blocks world domain.

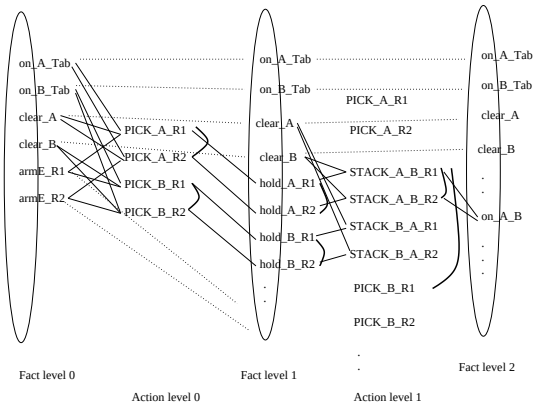

Fig. 4. The planning graph for the simple planning problem. Some details have been omitted for clarity.

1.1.1 Graphplan-based Planning

We briefly review Graphplan[6], a state-of-the-art planner, and show how its treatment of resources is not scalable. Consider a simple planning problem (See Figure 3) in the blocks world domain where blocks A and B are on a table and the goal is to achieve block A on top of block B using the available robots $R1$ and $R2$.

Graphplan performs forward state-space refinement over disjunctive partial plans [24] that corresponds to a unioned representation of the forward state-space search space. To improve pruning power in these disjunctive plans (*planning graphs* in Graphplan parlance), Graphplan infers and propagates information about disjuncts which cannot hold together in a solution (mutex relations). For the simple planning problem in Figure 3, the planning graph is shown in Figure 4. From the initial state, only actions to pick up the blocks are applicable. The mutex relations for actions and facts (predicates) are shown by curved lines. The goal state appears for the first time at the second fact level. 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.

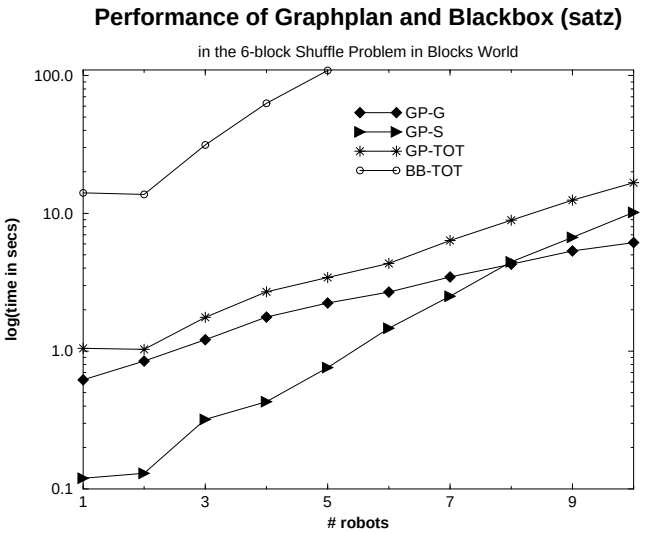

Fig. 5. Performance of Graphplan and Blackbox(satz) on the 6-block *Shuffle* problem with varying number of robots. Performance degrades with increasing number of resources.

To motivate the need for separating resource scheduling, consider the behavior of Graphplan in a modified blocks world domain that contains multiple robot hands. If Graphplan is run 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 since it is easier to resolve resource conflicts with abundant resources. Figure 5 shows the performance of Graphplan on the "*shuffle*" [2] 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! The increase in performance time is due to the increased cost in constructing the planning graph (GP-G) as well as the time for searching the planning graph (GP-S). Note that *both of them* increase with the number of robots.

One can further check if the results are consistent when the problem size is scaled independent of the number of resources. Figure 6 shows 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. Again the planning performance degrades with the increase in resources as well as domain size.

This rather counter-intuitive behavior of the planning algorithm (in Figure 6) can be deciphered once we realize that *every causal failure is being needlessly rediscovered multiple times* with different identities of the robot hands. The

[2] Shuffle problem is the multiple robots version of the 6-block *blocks_facts_shuffle* problem in the Graphplan distribution. Later k-block *shuffle* versions are also considered.

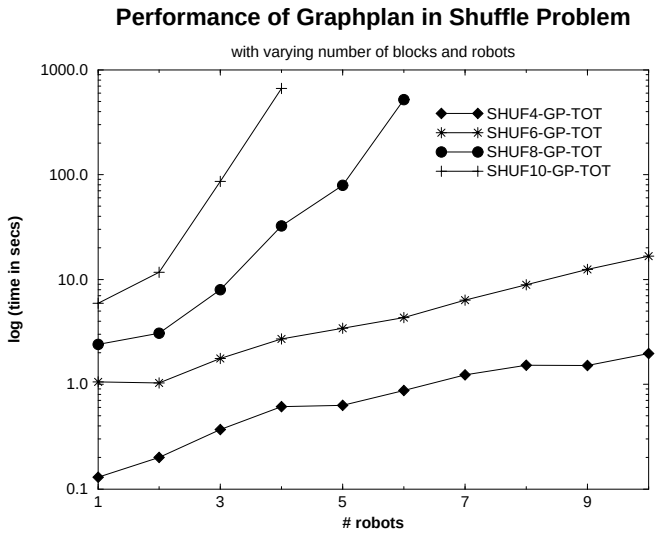

Fig. 6. Comparative performance of Graphplan in *shuffle* problem of 4, 6, 8 and 10 blocks. Performance degrades with increase in size of the domain as well as resources.

plan length and the number of steps in the plan reduce with increased resources as more resource conflicts get resolved at a level[3] (see Figure 7 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 performance degradation with the number of resources is not peculiar to Graphplan, experiments were run with Blackbox [27], which uses SAT techniques for searching the plan graph; and UCPOP [45], a traditional partial-order planner[26],[36]. Similar behavior was found in both cases. The plot titled BB-TOT in the left graph in Figure 5 illustrates the behavior of Blackbox.

1.1.2 Heuristic state search Planning

Heuristic state search planners perform state-space search guided by heuristics. FF [19], one of the fastest planners at the AIPS 2000 Planning Competition[2] of this type and AltAlt [58], both use heuristics derived from a reduced plan

[3] In Graphplan parlance, a plan step is also called an operator level. The terms step and level will be used interchangeably.

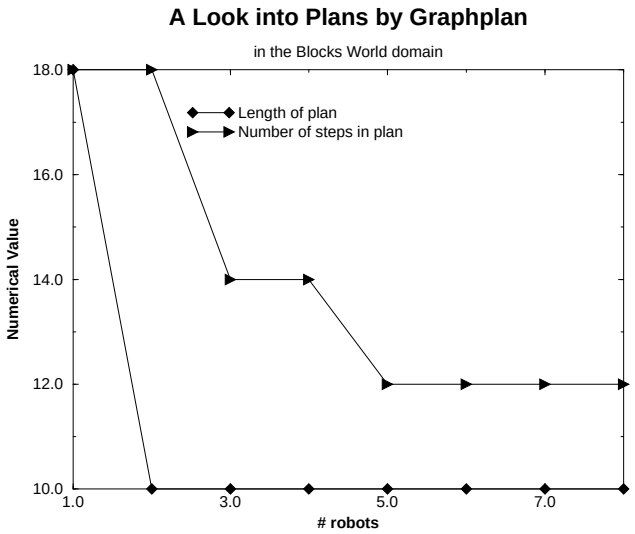

Fig. 7. The nature of plans produced by Graphplan on a blocks world problem with varying number of robots. As more resources are added, the plan length and number of steps in the plan reduce because there are more possibilities of avoiding resource bottlenecks and so, more ways of generating a parallel plan.

graph structure of the problem. Table 1 shows the result of running these planners on the multiple-robot *shuffle* problem.

We see that the performance of FF and AltAlt monotonically degrades with the increase in the number of resources (robots)[4]. Hence, resources are not handled in a scalable manner. The fact that FF also worsens with resources shows that the difficulty of handling resources in a monolithic planner is independent of the direction of the search.

1.1.3 Partial-order Planning

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 the Sussman Anomaly (See Figure 8) which has 3 blocks, the increase in robots does increase the planning time (refer to Table 2). The search space here gets large with increasing resources as all possible clobberers of conditions have to be considered for completeness. If threats are resolved at every search node before its children are explored, the increase in search space will affect the performance of UCPOP adversely.

[4] It is interesting to note that these planners can handle problems with up to 100 resources while Graphplan could only handle problems with up to 10 resources in 100 seconds.

	FF			AltAlt		
problem	time	#A	#R	time	#A	#R
shuffle_b6r3	0.03	14	3	0.12	14	3
shuffle_b6r5	0.06	12	5	0.23	12	5
shuffle_b6r10	0.23	12	5	0.70	12	5
shuffle_b6r20	1.57	12	5	2.74	12	5
shuffle_b6r30	4.61	12	5	6.74	12	5
shuffle_b6r40	11.71	12	5	13.40	12	5
shuffle_b6r50	21.40	12	5	35.41	12	5
shuffle_b6r60	39.78	12	5	114.22	12	5
shuffle_b6r70	68.40	12	5	241.88	12	5
shuffle_b6r80	105.19	12	5	729.34	12	5
shuffle_b6r90	155.46	12	5	OoM	-	-
shuffle_b6r100	244.06	12	5	-	-	-

Table 1
Performance of FF and AltAlt in the *shuffle* problem on a linux machine (PIII 500Mhz, 516MB RAM). FF runs with default option. AltAlt runs with the ADJ-SUM2 PACTION NOLEVOFF options. #A is the number of actions in the plan returned, and #R is the number of resource, robot hand in this case, used in this problem. 'OoM' means *Out of Memory*.

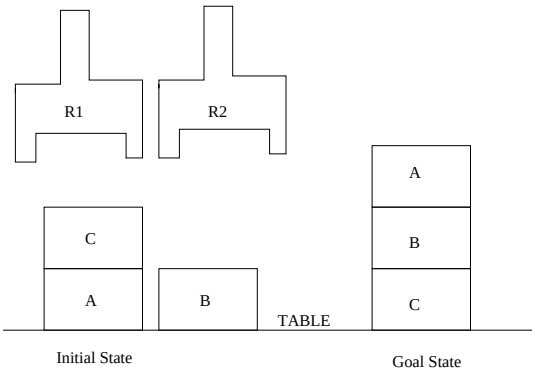

Fig. 8. The Sussman Anomaly problem (with robots).

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

Table 2
Table showing the performance of UCPOP in the Sussman Anomaly problem with varying number of robots.

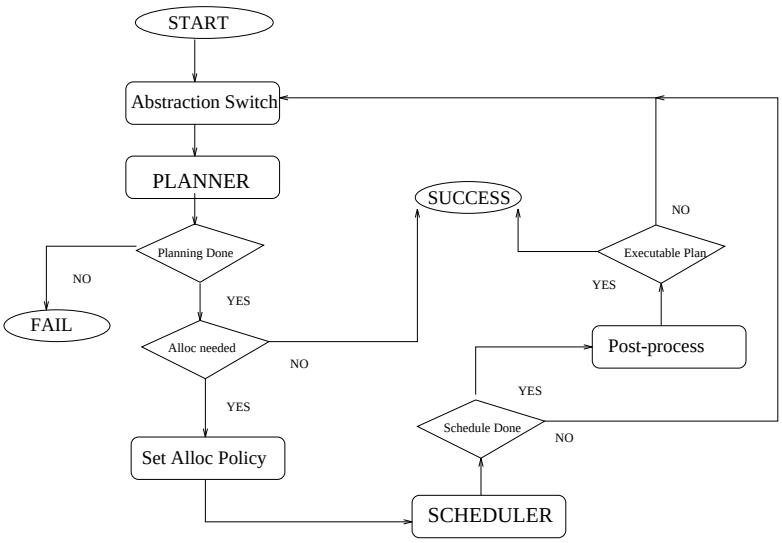

Fig. 9. `RealPlan`: A unified planning-scheduling framework.

1.2 `RealPlan` Approach

Figure 9 provides a general overview of the new approach called `RealPlan`. The 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). The focus of this work is on discrete reusable as well as non-sharable resources. But it is argued in Section 8 that the approach can be extended for continuous resources as well. After planning is complete, a scheduler interacts with the planner to decide which resources to actually allocate. The approach is implemented on top of the 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 it also significantly outperforms Graphplan on several benchmark problems. Although we use Graphplan as the base planner to focus our discussion, the general approach is by no means limited to Graphplan and can be easily adapted to other planners.

The planner-scheduler interaction is supported in the `RealPlan` framework as *master-slave* relationship (called `RealPlan-MS`, see Figure 10) with the base planners are Graphplan and GP-CSP[10]. In GP-CSP, the plan graph of Graphplan is converted into a Constraint Satisfaction Problem (CSP[4]) and solved with a standard solver. If the declarative scheduling method fails to allocate resources in the context of given resources, time limit and nature of allocation policy, the partial schedule in a failed iteration is not pursued and the responsibility transfers to the planner to change any of the these 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 necessary domain translation for executability.

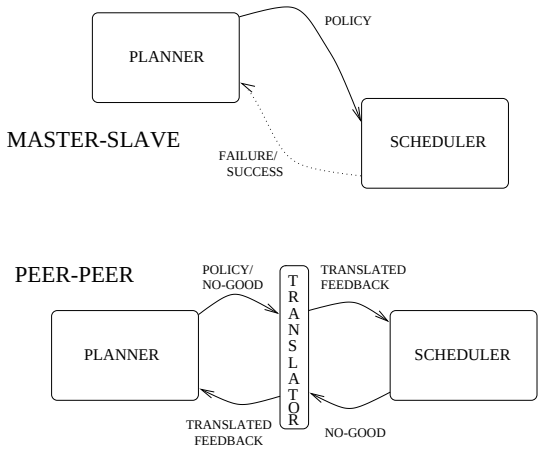

Fig. 10. *Communication relationships between the planner and the scheduler.* ***RealPlan-MS*** *refers to* master-slave *(details in Section 4) while* ***RealPlan-PP*** *refers to* peer-peer *relationship (details in Section 5).*

The framework also supports *peer-peer* relationship (called `RealPlan-PP`) between the planner and the scheduler. Using a planner like GP-CSP[10] which poses plan extraction as a CSP problem, the scheduler can inform the planner about the source of infeasibility in terms of the variables and constraints in the planner's CSP (nogoods). The planner can then use the nogoods to backtrack and select another plan, thereby handling even *moderately coupled* problems. This type of "multi-module dependency directed backtracking" approach is a variation on the hybrid planning methodology developed in [23], and is also akin to the approach used to link satisfiability and linear programming solvers in [57].

In `RealPlan-MS`, if all the allocation policies lead to failure or unexecutable plans in a domain, this implies that planning and scheduling were in fact, *not loosely coupled*. Moreover, performance considerations may not always favor the exploration of all causal plans in `RealPlan-PP`. In such cases, the framework retains the ability to switch off resource abstraction and resort to traditional planning.

There are a number of technical challenges that arise in making the new approach work. First, the resources have to be identified in a given domain. Second, one has to decide about the optimization criteria during scheduling. Third, resources have to be allocated 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) [5].

The working definition of resources is:

[5] Such a plan may not be optimal if actions have differing costs, but this is a known limitation of Graphplan.

A resource is any object for which actions may contend and which is secondary (actual identity 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 Section 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 and reallocating actions are allowed, the problem is in fact a Dynamic Constraint Satisfaction Problem (DCSP[41]) because these actions (variables) control the normal action variables.

In summary, we use a declarative approach for resource allocation where all the constraints of the scheduling problem are formulated as a Dynamic Constraint Satisfaction Problem (DCSP), compiled into a CSP problem and passed to a CSP solver (specifically, the backjumping solver in [55]). If the declarative scheduling method fails to successfully allocate available resources, given the time limit and nature of the allocation policy, the responsibility transfers to the planner to change any of the permissible parameters and try again (in `RealPlan-MS`) or learn from the failure and try with a new causal plan (in `RealPlan-PP`). If the resource allocation succeeds and new free and reallocation actions were added by the scheduler, the scheduled plan is post-processed for necessary domain translation for executability.

1.3 Scheduling as a Post-planning Phase

We further discuss the ramifications of resource reasoning within planning and how it may be leveraged for efficiency. As soon as Graphplan generates a plan graph up to 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 resources (i.e., being least committed to resources) needed by actions during planning and avoiding checking all interactions between them. The assumption is that the abstract actions in the plan would be allocated resources (following plan search) in a subsequent scheduling phase.

For UCPOP, considering resources during planning corresponds to additional clobbering possibilities which can degrade the planner performance if conflict resolution is enabled. The separation of resource reasoning from planning causes postponement of resource commitments during conflict resolution which reduces the search space. Hence abstraction of resources is an equally promising approach for UCPOP. Since Graphplan-based planners are the fastest breed of planners available today [6], we will focus on them for the rest of the paper but the techniques are still appealing for other forms of AI planning.

When resources are abstracted in Graphplan, as in `RealPlan`, the plan graph size is reduced and a maximally concurrent plan is obtained. 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 problems with initial resources $N \geq R$, the same method can be applied. As the number of resources decreases, the scheduler has to serialize the plan in line with the resource limitation. Serialization may involve moving the parallel steps to less-constrained levels and introducing steps to release unnecessary allocations and re-allocate them where needed. Figure 11 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 the proposed approach of postponing scheduling occurs 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. For `RealPlan`, 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. They were determined by modifying the search phase of Graphplan to print all the plans. 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, an alternative schedule can still make the same causal plan work.

[6] In the AIPS 1998 planning competition[37], 3 out of the 4 finalists were based on Graphplan. In AIPS 2000[2], they were still prominent.

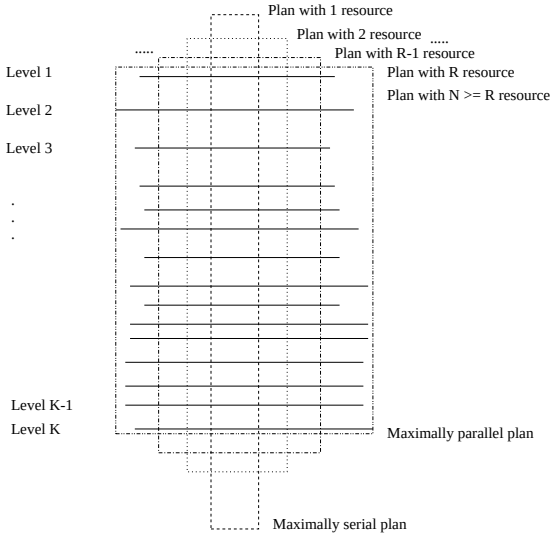

Fig. 11. When there are enough resources to overcome any resource conflict, one gets a maximally parallel plan. Otherwise, the scheduler has to serialize the plan in line with the resource availability. For the most resource constrained problems (normally 1 resource), one gets the maximally serial plan.

1.4 Paper Outline

Here is an outline of the paper. We start with a discussion about resources and their role in planning in Section 2. Next, a new planning formalism is developed to abstract resources during planning and schedule them in a separate phase in Section 3. A declarative method for solving the resource scheduling problem as a Dynamic Constraint Satisfaction Problem is described in Section 4 including the `RealPlan-MS` planner-scheduler interaction in the context of scheduling policies and their interpretation in the DCSP formulation. In Section 5, `RealPlan-PP` is developed including the learning of scheduling failure to select a different causal plan. In some cases, the scheduled plan may need post-processing, and this is covered in Section 6. Empirical results in Section 7 show that the new method makes planning efficient when resource reasoning is effectively used. Section 8 puts the current work in perspective with previous work. Finally, we conclude in Section 9 with a summary of the contributions and a discussion of future work.

2 What are Resources ?

In this section, issues related to resources are discussed. Let us formalize some terminology. A planning domain consists of operators and physical objects. Operators change the state of the world. All objects in the domain are instantiations of defined 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.

2.1 Resource Identification

Recall from before 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. The problems require the packages to be moved to their goal locations, the users do not care if $Plane2$ is selected instead of $Plane1$, but they will be concerned if the package lands in *Boston* instead of *Philadelphia*. So, planes are resources [7]. The proposed definition captures the theme that scheduling should consider interactions among objects that do not matter (i.e., resources) for the acceptance of the plan. Within this broad framework, resources may additionally provide domain control knowledge, suggestions to the planner about which interactions to disregard or special monitoring and execution capabilities, as is accomplished in some systems (Also see related work in Section 8).

A related issue that must be addressed is how the resources are identified in a planning domain. While this is not a central focus of the current work, we have given it some consideration. The most obvious approach is to let a domain expert specifies the resources in the domain at the outset. In most real domains, resources will be identifiable quite naturally. A resource specification format has been implemented for this purpose (See Figure 12). Appendix A illustrates the specification of *robot* as a resource in blocks world domain and *truck* and *airplane* in the logistics domain.

In the rare cases where the domain specification is done without explicit resource identification, there are ways of automating the resource identification process. 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 to detect that a robot is a resource type in the blocks world domain or a gripper is a resource in the 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[17], by emulating the finite state machines implicit in the domain structure (legal operators and initial/

[7] But if the plane's identity does matter, then the plane type is not a resource. Note that an object is a resource for all the actions in a domain and not per-action as has been modeled in some systems like SIPE[56].

```
(resource ⟨resource-name⟩ [⟨resource-type⟩]
    (free
      (means
          (effects (⟨free-predicate⟩))
          (params {(⟨var-name⟩ VAR-TYPE)}+)
          (plans
             {(plan-nameplan-cost
                          {(step-name {(⟨action-name⟩)}+)}+)}+)))
    (unfree
       (means
           (effects (⟨unfree-predicate⟩))
           (params {(⟨var-name⟩ VAR-TYPE)}+)
           (plans
              {(plan-nameplan-cost
                           {(step-name {(⟨action-name⟩)}+)}+)}+))))
```

Fig. 12. An explicit resource specification format. Fields in ⟨ ⟩ are place holders while fields in [ ] are optional. Fields in { } scope the Kleene plus operator for one or more entires.

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 lose a property unconditionally, in contrast to a state space where corresponding objects only exchange properties. The system described in the current work can easily incorporate such domain modeling techniques because resource identification only affects the parsing of domain descriptions.

2.2 Summary

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

3 `RealPlan`: A New Planning Formalism

In this section, a planning model is explored in which resource allocation is decoupled from planning, and is handled in a separate "scheduling" phase (See

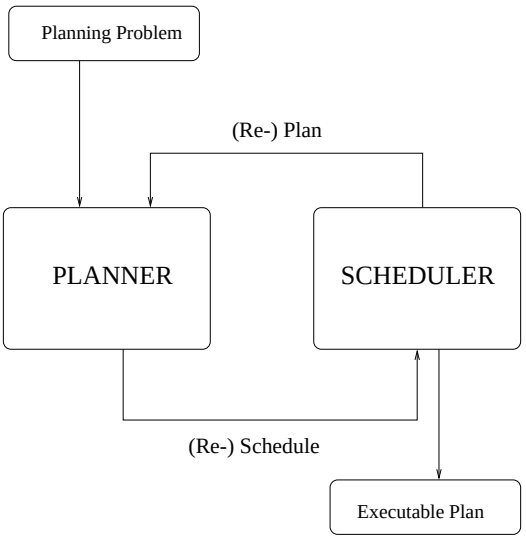

Fig. 13. A generalized model for decoupling planning and scheduling.

Figure 13). A *necessary* condition for a schedulable plan is that it should be causally correct irrespective of the nature of resources. An abstract plan (P') can be produced which is correct sans the resource allocation and it can serve as the 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 using STRIPS[14] representation, sharable discrete resources are assumed to have infinite capacity (e.g. trucks can load any number of packages) and continuous resources are assumed unlimited (e.g. fuel is available or not). The causal plan P' in our approach is also created under the most optimistic resource assumption (unlimited or infinite). During scheduling, the actual resources may be found insufficient to 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 carry out limited replanning by itself.

3.1 Decoupling Causal and Resource Reasoning

Figure 14 summarizes the new approach called `RealPlan`. In Graphplan terms, one can reduce both graph expansion and search overhead by abstracting the resources needed by actions during planning and ignoring all interactions between them, thereby obtaining a maximally concurrent causal plan. This plan will then be post-processed to allocate resources to actions. See Figure 9 for a schematic overview of `RealPlan`.

Based on the ideas about resources from the previous section, some object types are identified as resources. Now, if the resource abstraction switch is set,

(1) Identify resources: The system can recognize resources using already discussed methods in the previous section.

(2) If no resource information is available or resources are so low (usually one) that postponing their reasoning is counter-productive, perform conventional planning where interactions involving similar resources are addressed during planning.

(3) Suppose some of the objects are defined as resources. Planning proceeds as follows:
   (a) Assign dummy values to resource variables in the initial state and goal state such that equivalent resources have the same dummy value.
   (b) Do not compute interference relationships (mutexes) between resource equivalent operators. Operators may still interfere due to other preconditions/ effects.
   (c) Complete planning.

(4) Once a plan is obtained, allocate resources to the actions in the plan and resolve resource conflicts using any desirable 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.

Fig. 14. Synopsis of `RealPlan`

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 [8] resources have the same dummy value.
- Interference relationships (mutexes in the case of Graphplan) between resource equivalent operators are ignored. Operators may still interfere due to other preconditions/ effects.

If the problem is unsolvable at this stage, we know that resource scheduling is not going to make it solvable. Otherwise the resultant plan is given 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 15. The plan consists of 10 time steps (levels) with the number of resources left allocated at each level shown in the right column

[8] 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. The latter choice is used here and all objects in a resource type are considered equivalent.

Level	Actions by level	# Robots
1	Unstack_R_blkF_blkE	1
2	Unstack_R_blkE_blkD	2
3	Unstack_R_blkD_blkC	3
4	Unstack_R_blkC_blkB	4
5	Putdown_R_blkC	
5	Unstack_R_blkB_blkA	5
6	Stack_R_blkF_blkC	5
6	Pickup_R_blkA	
7	Stack_R_blkB_blkF	4
8	Stack_R_blkE_blkB	3
9	Stack_R_blkA_blkE	2
10	Stack_R_blkD_blkA	1

Fig. 15. A 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.

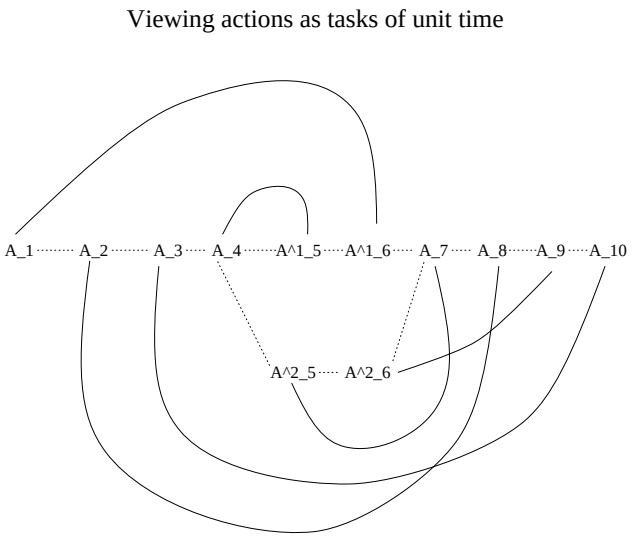

Fig. 16. View of the resource-abstracted plan in Figure 15 as a task network to be scheduled. Curved lines show resource spans while dashed lines represent partial ordering constraints between tasks (actions).

(marked "#Robots"). The abstract plan is seen as a task network in Figure 16.

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. We use a declarative method to address scheduling where the resource allocation problem is posed as a DCSP[41] problem and solved by a standard backjumping CSP solver (see Section 4). Let us consider the nature of scheduling.

A straightforward method for resource allocation is to assign a new or pre-

viously 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 modifies the abstract plan. The planning phase can suggest intent of desirable modifications to the scheduling phase in the form of following resource allocation policies:

(1) Must maintain concurrency of the plan.
(2) Allow serialization of the concurrent plan by moving actions from one step (level) to another less-constrained step.
(3) Allow introduction of actions to free unnecessary allocations and re-allocate the freed resources when needed again.

In the last policy, the scheduler is essentially allowed a limited re-planning role with the hope of efficiently producing a final plan. The idea is that if most of the resource constraints were already satisfied by simple allocation or plan serialization, the relatively small remaining conflicts could be resolved by changing only the affected portions of the plan. But if this is not the case, re-planning can degenerate into a full-fledged planning problem.

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 likely to be useless, counter-productive, because the planner is banking on concurrency in the plan while resource availability suggests the need for a serial executable plan.

Even though we note that the pathological case of one resource can (and should) be easily detected and avoided up front, resource postponement will be pursued to illustrate how the decoupled methods can also address this situation naturally.

3.2 Planner-Scheduler Interaction and Post-processing

The planner and the scheduler can communicate with each other according to a pre-determined relationship and its governing protocols. Two such relationships are shown in Figure 10: *master-slave* (`RealPlan-MS`) and *peer to peer* (`RealPlan-PP`) relationships. In the former, the planner plays the primary decision-making role while the scheduler ensures that a resource allocation is

possible. In the latter, both the planner and scheduler can make decisions as well as gainfully interpret the feedback of the other module.

`RealPlan-MS` is supported in Graphplan and GP-CSP[10], the two planners that were considered. The resource allocation policies suggested by the planner have a precise semantics for the scheduler in terms of its CSP. If the declarative scheduling method fails to allocate resources in the context of given resources, time limit and nature of allocation policy, the partial schedule in a failed iteration is not pursued and the responsibility transfers to the planner to change any of these parameters and try again. If resource allocation succeeds and no additional actions were inserted, the scheduled plan is executable and can be returned. However, if new free and 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 Section 6. If all the allocation policies lead to failure or unexecutable plans in a domain, this implies that planning and scheduling were in fact, *not loosely coupled* in this instance. In such a case, `RealPlan-MS` retains the ability to switch off resource abstraction and resort to traditional planning.

Planner-scheduler interaction is also supported as `RealPlan-PP` in GP-CSP. Here, the failure reason during scheduling is explained as a nogood in terms of scheduling CSP variables and made available to the planner in terms of the planner's CSP nogoods. The planner can use the additional nogood information to generate a different causal plan (in addition to changing the allocation policy) before calling the scheduler again.

3.3 Summary

In this section, a planning model was presented where resources are abstracted during the initial causal planning phase and later considered during the resource allocation phase. Different aspects of `RealPlan` were discussed including resource allocation and planner-scheduler interaction. Two approaches have been developed for planner-scheduler interaction: resource scheduling – `RealPlan-MS` and `RealPlan-PP`, which will be discussed in detail in the next sections. We draw parallels between `RealPlan` and project management in Section 8.

4 Scheduling as a CSP problem

In this section, we investigate casting the resource allocation problem as different forms of Constraint Satisfaction Problems (CSP). First, we pose it as

a declaratively specified Dynamic Constraint Satisfaction Problem (DCSP) problem that can be solved using any standard solver. Next, we characterize specific easier cases of the general CSP formulation and present a specialized (procedural) algorithm to handle each of the simpler classes. We also give semantics to these classes in terms of restrictions on the general DCSP problem. We conclude with a brief discussion of other scheduling methods we investigated.

The resource allocation problem is specified separately for each resource type 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 $(S_{ij} : \langle A_i, A_j, C \rangle)$ that have to be allocated resources. The span S_{ij} just says that the effect C of action A_i is produced at level i and consumed at level j as a precondition of action A_j. Spans may be referred by S_1, S_2, etc. if there is no need to identify its constituent actions in a given context. 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.

A Constraint Satisfaction Problem (CSP[4]) consists of a set of variables, each with a finite range of values (also called the domain of the variable), and a set of constraints. The aim is to find a satisfying assignment for all the variables which is compatible with the constraint set. In a Dynamic Constraint Satisfaction Problem (DCSP[41]), there are two types of variables: activity variables and normal variables. Initially, only a subset of the variables is active, and the objective is to *find assignments for all active variables that is consistent with the constraints among those variables.* In addition, the DCSP specification also contains a set of "activity constraints." An activity constraint is of the form: "if variable x takes on the value v_x, then the variables $y, z, w...$ become active." A DCSP problem can be translated into a normal CSP problem by augmenting the domain of variables with a dummy value $\perp$ (NULL) to signify that those variable may be inactive, and modifying the constraint specification accordingly.

4.1 Declarative Scheduling

One can specify the resource allocation constraints declaratively and let a constraint solver find a satisfying assignment for resources for each action. This problem is an instance of DCSP where the activity variables (corresponding to free/ reallocating actions above) control when the normal variables are considered for value assignment. Let us discuss the necessary preparations.

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 3
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.

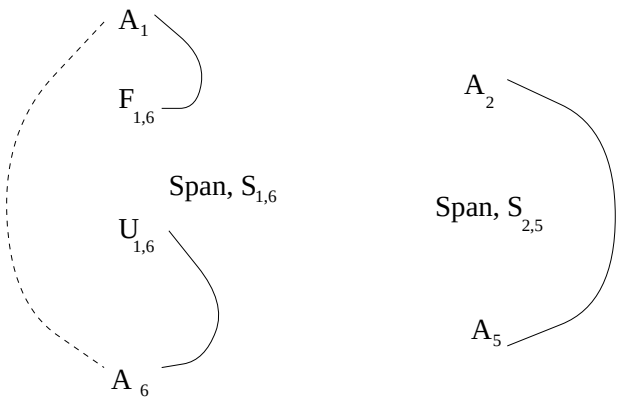

Fig. 17. Example of spans with freeing/ reallocating actions on the left and without them on the right.

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 denoted as N_i and their corresponding position variable is PN_i. Given a span $(S_{ij}{:}\langle A_i, A_j, C\rangle)$, two optional actions are associated with the span, F_{ij} for freeing the resource and U_{ij} for reallocating the resource. (See Figure 17 for illustration). The constraints on variables and their values are listed respectively in Tables 3, 4 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 include a dummy value $\perp$ (NULL) signifying that the corresponding action is potentially 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 3.

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)	If segments of two spans overlap, they cannot share resources over that segment

Table 4
Relationship among action variables.

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 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 4.

Condition	Constraint on values
$PF_{ij}^1 = PU_{ij}^1 = PF_{ij}^2 = PU_{ij}^2 = \perp$	INTERACT(PA_i^1, PA_j^1, PA_i^2, PA_j^2) $\Rightarrow RA_i^1 \neq RA_i^2$
$PF_{ij}^1 = PU_{ij}^1 = \perp$; $PF_{ij}^2, PU_{ij}^2 \neq \perp$	INTERACT(PA_i^1, PA_j^1, PA_i^2, PF_{ij}^2) $\Rightarrow RA_i^1 \neq RA_i^2$ INTERACT(PA_i^1, PA_j^1, PU_{ij}^2, PA_j^2) $\Rightarrow RA_i^1 \neq RA_j^2$
$PF_{ij}^1, PU_{ij}^1 \neq \perp$; $PF_{ij}^2 = PU_{ij}^2 = \perp$	INTERACT(PA_i^2, PA_j^2, PA_i^1, PF_{ij}^1) $\Rightarrow RA_i^2 \neq RA_i^1$ INTERACT(PA_i^2, PA_j^2, PU_{ij}^1, PA_j^1) $\Rightarrow RA_i^2 \neq RA_j^1$
$PF_{ij}^1, PU_{ij}^1, PF_{ij}^2, PU_{ij}^2 \neq \perp$	INTERACT(PA_i^1, PF_{ij}^1, PA_i^2, PF_{ij}^2) $\Rightarrow RA_i^1 \neq RA_i^2$ INTERACT(PA_i^1, PF_{ij}^1, PU_{ij}^2, PA_j^2) $\Rightarrow RA_i^1 \neq RA_j^2$ INTERACT(PU_{ij}^1, PA_j^1, PA_i^2, PF_{ij}^2) $\Rightarrow RA_j^1 \neq RA_i^2$ INTERACT(PU_{ij}^1, PA_j^1, PU_{ij}^2, PA_j^2) $\Rightarrow RA_j^1 \neq RA_j^2$

Table 5
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.

Moreover, if a resource is non-sharable, additional constraints have to be specified as summarized in Table 5. 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 $S_{1,8}$ and $S_{2,6}$ interact between levels 2 and 6. Therefore, they can not share a robot (resource) in this interval unless they can make use of the freeing/reallocation actions. For example, span $S_{1,8}$ can free its assigned resource R (from level 1) and give it to span $S_{2,6}$ at level 2. After using R for its entire duration, span $S_{2,6}$ can free it after level 6 so span $S_{1,8}$ can use it to finish its remaining duration. Freeing (and reallocating) actions will result in sub-intervals over which a robot is used by exactly one span.

Finally, in addition to the constraints in Tables 4, 5, we have certain global constraints:

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

  Sum $(A_i^m) \leq N$, $i = 1..L$, $m = 1..| A_i |$ (i.e. number of actions at level i)
- To optimize the plan, we can set the objective function as minimizing the total number of actions in the plan. This may seem strange at first glance because scheduling is usually related with minimizing resource usage. The reason is that in classical planning, any sound and complete plan is acceptable while a shorter plan is desirable. But recall that the number of resources in a problem is also 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, resource optimization requirements can also be easily handled by changing it to an optimizing CSP problem. The constraint to minimize actions is currently enforced implicitly by the CSP solver in conjunction with the scheduling policies (discussed later).

  Objective: Min Sum($| A |$) (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[55].

4.2 Specializing Scheduling into Classes

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 simpler classes, as shown in Figure 18. The main classes are 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. In fact, we will not reach the scheduling stage because the causal plan was not complete. Hence, the problem can be handled by the planner more efficiently.
- Class INFRES: If indeed the resources are sufficient to overcome all the 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 the 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 18, and discussed later in the section. It turns out that one can handle

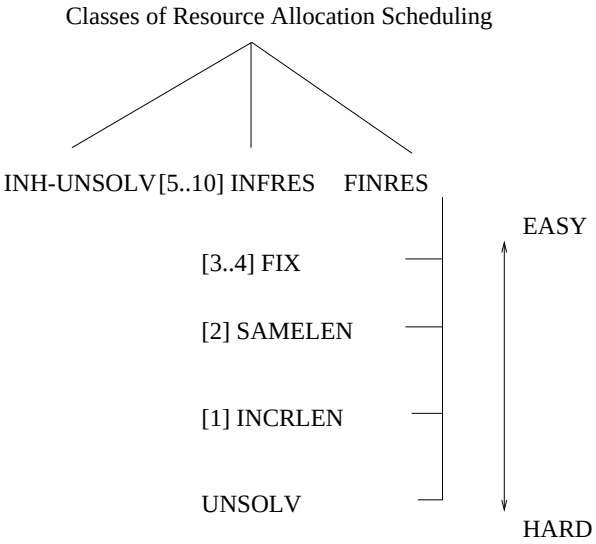

Fig. 18. A Classification of resource allocation instances (with indication of resource quantities that make *shuffle* problem fall into each of the classes).

several of these sub-classes through efficient (backtrackfree) methods.

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 18. For solving the resource allocation problem, rather than using one general scheduling method for all classes, control can cycle through scheduling methods tailored to each of the specific classes (from the easiest to the hardest). By using this approach, we can efficiently allocate resources with the least amount of modification to the given plan. The latter is important in cases where we are given a problem with excess resources. We still want to try and use as few resources as needed, while a general CSP formulation may find a plan that uses all the resources just because they are available. Iterating through classes ensures that the plans developed with the procedural algorithm is comparable in quality to those developed by the normal planner. In the rest of this section, we sketch methods for efficiently handling the special cases. We first illustrate these methods with corresponding special algorithms. Subsequently, we shall show that the classes can also be handled within the declarative CSP approach (of Sec 4.1). Special classes can be implemented by different strategies to restrict the domains of variables in specific ways.

Solving the resource allocation problem: INFRES case: Pseudo-code of a procedural resource allocation algorithm is shown in Figure 19. This algorithm 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

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

Fig. 19. Pseudo-code for allocating resources

to the abstract plan. In $Solve\_INFRES$, the span is allocated resource (for each type) from a pool of available resources which gets replenished whenever the resources are no longer needed (i.e., beyond the respective span). 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 $Solve\_INFRES$. See Figure 20 for an example.

Solving the resource allocation problem: FINRES case: Based on the amount of resources, Class FINRES can be divided into a number of sub-classes as summarized in Figure 18. These sub-classes are currently detected during scheduling itself. The 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 is 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

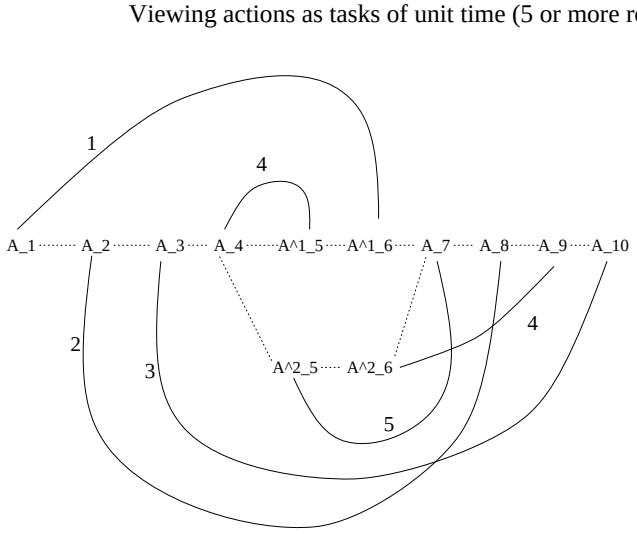

Fig. 20. Scheduling task network of *shuffle* problem by INFRES. Curved lines show resource spans and numbers next to them are the resource allocated. Dashed lines represent ordering constraints.

introducing a freeing[9] 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 effect is needed at level j again (an action to reallocate the resource will be added at level j-1). See Figure 21 for an example. *Shuffle* problems with 3 and 4 robots can be handled by this method. In particular, with respect to Figure 15, the robot corresponding to span $\langle A_1, A_6^1,$ *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 as planning itself, $Solve\_SAMELEN$ is not allowed to increase the plan length and is required to maintain the relative action positions. See Figure 22 for an example. *Shuffle* problem with 2 robots can be handled by this method. In the *shuffle* problem in Figure 15, 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

[9] In general, adding actions to a plan can change its causal structure but we 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. See a discussion in Section 6.

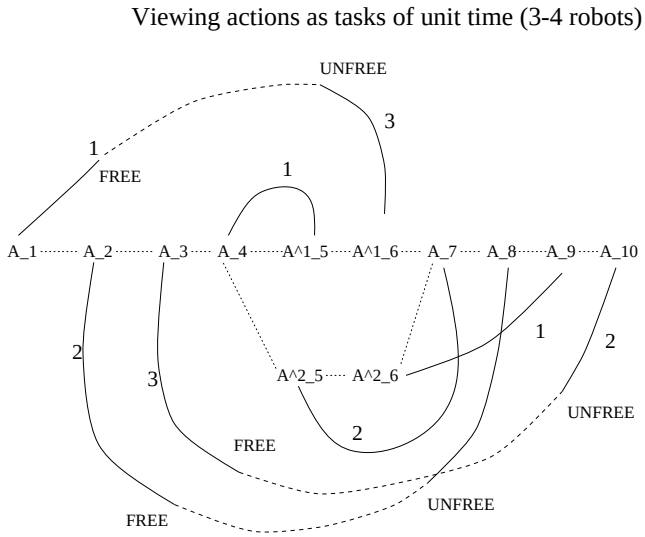

Fig. 21. Scheduling task network of *shuffle* problem by FIX. Curved lines show resource spans and numbers next to them are the resource allocated. Dashed lines represent ordering constraints.

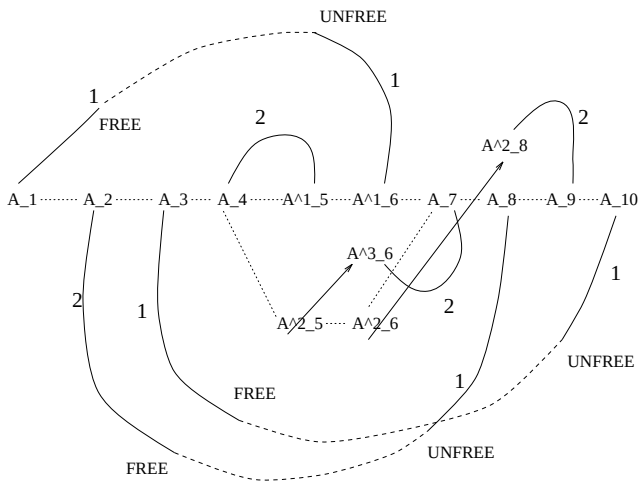

Fig. 22. Scheduling task network of *shuffle* problem by SAMELEN. Curved lines show resource spans and numbers next to them are the resource allocated. Dashed lines represent ordering constraints. Arrows refer to the movement of actions to a lower, less-constrained level.

$SAMELEN$ in Figure 18.

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. Problems in this class are given either to the full declarative scheduler 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, one can identify this class at the start of scheduling; otherwise it cannot be determined until after Class INCRLEN.

Classes INFRES, FIX and SAMELEN are handled 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 in Section 7.2.1, this penalty is small and easily offset by the savings in other classes. As mentioned earlier, the reason a series of methods is used in this order is to keep the number of additional actions as small as possible while maintaining the optimal plan length.

4.3 Viewing Classes as Policies for Planner-Scheduler Interaction (`RealPlan-MS`)

As mentioned in Section 3.2, the communication between planner and scheduler can be seen as policy suggestions 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.

The different classes described in the previous section can be interpreted 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 9) ***in terms of constraints on the values of variables***.

Table 6 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, we identify three subcases. If actions are considered immovable, this corresponds to Class FIX.

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 6
Allocation policy and restrictions on values of variables. L^{MAX} is some maximum length ($L^{MAX} \succ L$) up to which the steps of the plan can be increased.

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 up to 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} ($L^{MAX} \succ L$) in Class INCRLEN.

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.

4.4 Summary

In this section, the resource allocation problem was discussed as a Constraint Satisfaction Problem. We presented it as a declaratively specified DCSP problem and translated into a standard CSP problem. We also sub-divided it into simpler classes and investigated solving them with cheaper specialized algorithms leaving out the harder ones for the general DCSP or default planning. The planner to scheduler interaction in `RealPlan-MS` is seen as policies that have semantics in terms of domains of DCSP variables. In the next section, we discuss the `RealPlan-PP` model of planner-scheduler interaction. We follow it with how the inserted actions in the scheduled plan are translated to executable sub-plans to generate the final plan.

5 Peer to Peer Interaction Between Planner and Scheduler

Until now, the interaction between the planner and scheduler is essentially uni-directional. The scheduler tries various ways of allocating resources to the causal plan, but when it fails, the decoupled approach is abandoned and the planner goes back to solving the problem from scratch, taking both causal and resource allocation decisions into account. Although this approach does work fine in most scenarios where there is indeed loose coupling between planning and scheduling decisions, it fails in situations where the coupling is tighter. In these latter scenarios, we would like the scheduler to get back to the planner with specific information about the reasons for its failure to allocate resources

to the plan. Intuitively, we would expect that such an explanation would be in terms of some specific actions in the causal plan whose presence at their respective places makes resource allocation infeasible. Ideally, we would then like the planner to *resume* its search, armed with the "explanation" of the failure supplied to it by the scheduler, and come up with a causal plan that avoids that failure. Such a plan will then be sent to the scheduler. Of course, it is quite possible that the scheduler is unable to allocate resources to this new plan too (albeit for a different reason) – in which case the interaction cycle continues, until a schedulable plan is eventually found. Implementation of this style of interaction regime requires several steps:

Explanation Generation: The scheduler needs to be modified to provide a compact explanation of its failure to allocate resources.

Translation: The scheduler's explanation of failure needs to be "translated" into a form that makes sense for the planner's search space.

Generation of an Alternative Plan: The planner should then be able to use the translated explanation of failure to generate an alternate plan that avoids this failure.

To make the implementation of these steps uniform, we have decided to use CSP as the substrate for both planning and scheduling problems. As illustrated in Figure 23, the advantage of this uniform representation is that the interaction cycle can be thought of in terms of a 2-stage dynamic constraint satisfaction problem, with the planner's CSP making up one stage and the scheduler's CSP making the second one. Conceptually, the planner's CSP consists of top level goals as variables, and action choices as the values; while the scheduler's CSP has actions as variables, and the identities of allocatable resources as their values. The assignments made in the planner's CSP activate the variables requiring resource allocation constraints in the scheduler's CSP, while the failure in solving the scheduler's CSP can be communicated back to the planner's CSP in terms of the action choices that did not pan out in terms of resource allocation. If the planner's search is done on a CSP substrate, the scheduler's failure information can simply be used as additional declarative constraints on the planner's search.

We have already discussed how scheduling part is posed as a CSP problem (see Section 4.1). In the following subsections, we discuss the details of posing planning as a CSP problem, as well as the implementation of the three steps discussed above.

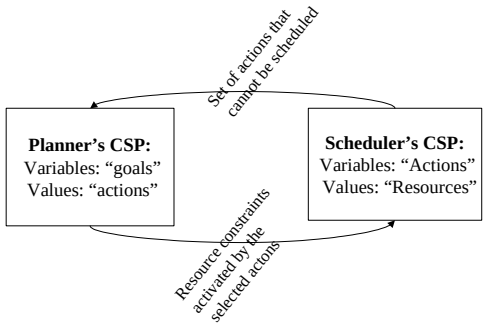

Fig. 23. Relationship among variables and values of planner and scheduler CSPs.

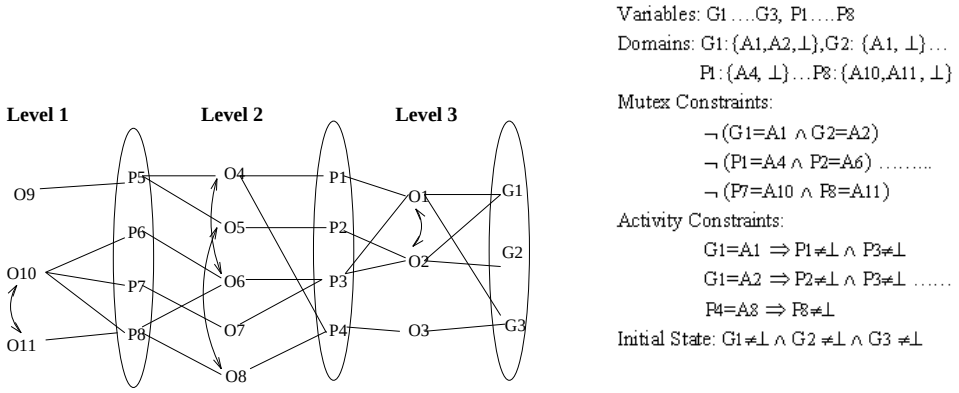

Fig. 24. Compiling a planning graph into a CSP

5.1 Posing Planning as a CSP

To pose the planning part of `Realplan` as a CSP, we adapted our work on the GP-CSP system [10], which replaces the "backward search" portion of the Graphplan planning algorithm with a CSP search regime. Specifically, the planning graph is first compiled into a CSP problem, and is then solved using standard CSP search strategies. The compilation step involves converting the propositions in the planning graph to CSP variables, and actions in the planning graph to CSP values (with the set of actions supporting a proposition in the planning graph constituting that variable's domain).

Figure 24 shows a simple planning graph and its compilation into a CSP. Since only the top-level goals and any subgoals activated by those goals need action support, the domain of every proposition variable is augmented with a null ("⊥") value, with the idea that propositions that do not need an action to support them will wind up taking the null value by default. There are two types of constraints – the mutex constraints which say that certain proposition-action combinations are infeasible, and activation constraints which state that when a proposition p takes an action a as its value, the propositions corresponding to the preconditions of a cannot then take null ("⊥") values.

If the resource allocation fails, the reason for the failure has to be extracted and communicated to the planner. The three steps are detailed next.

Explanation Generation

Generating failure explanation for the scheduler can be done in a straightforward fashion by using the explanation-based backtracking techniques [21,20]. Specifically, if we employ a conflict-directed backjumping strategy [47,21] to guide the solution of the scheduling CSP, in the event the CSP cannot be solved, the conflict set at the root of the search tree shows the subset of variables of the scheduling CSP that are causing the failure.

Explanation Translation

After we get the failure explanations from the scheduler in the form of nogoods, NG = (X_1,X_2,..,X_n), we have to transform them into the form that the planner can understand. On the face of it, this problem seems trivial – after all, as shown in Figure 10, the variables of the scheduler should correspond to actions in the planner, and thus they should make sense to the planner directly. There are, however, two complicating factors:

- Contrary to the simplified view illustrated in Figure 10, the variables of the CSP problem, as set up in Section 4.1, do not have a one-to-one correspondence with the actions in the causal plan (see Figure 23). Instead, the relation is more indirect – in terms of resource spans, positions, etc., some of which don't make direct sense to the planner. Specifically, each variable in the scheduler's CSP, and consequently in any failure explanation passed back by the scheduler, corresponds to one of the 9 types of variables listed in Table 3.
- Translation is further complicated by the fact that we have multiple resource allocation policies to help the planner in communicating the resource allocation preferences of the user to the scheduler. Different scheduling policies lead to different types of CSP encodings of the scheduling problem. As a result, we need different rules in translating the scheduler's failure explanations back to the planner. The complexity of the translation process increases along with the complexity of the different classes.

Regarding the second issue, for the simplest class, INFRES, each failure explanation (conflict set) resulting from solving the scheduling CSP can be directly converted to one set of action nodes in the planning graph of the planner. For class FIX, besides the variables corresponding to actions in the plan, the scheduling CSP also has variables representing the free/unfree actions. Therefore, we need to reason about such variables related to the spans, before we can

convert the scheduler's conflict set back to the set of actions in the planner. For the class SAMELEN, because the ranges of the possible position values for the actions in the scheduling CSP are widened, one variable in the scheduling CSP corresponds to many action nodes in the plan graph. Therefore, the translation rules for this class should be able to convert one conflict set of the scheduling CSP to many action sets in the planner CSP. Finally, for the INCRELEN class, because the ranges of the position of actions in the scheduler encoding (SE) are allowed to go beyond the highest level of the plan graph, there is no clear way to map them back to action sets in the planners. Therefore, we currently do not support the `RealPlan-PP` interactions between the two modules in this class.

To aid the translation, we start by keeping track of how the scheduler's variables are derived from the resource spans in the causal plan. Suppose there is a span $S_{i,j}$ that is started and ended by the actions A_i and A_j. We know (see Section 4.1) that this single span gives raise to the variables, RA_i, RA_j, PA_i and PA_j. To convert the variables in the scheduler's CSP to action values of the planner's CSP, we thus invert this mapping as follows:

- $RA_i, PA_i \Rightarrow A_i$
- $RA_j, PA_j \Rightarrow A_j$
- $RF_{ij}, PF_{ij}, RU_{ij}, PU_{ij} \Rightarrow A_i, A_j$

The set of actions resulting from this mapping applied to the variables in the scheduler's failure explanation is then taken as the set of actions in the causal plan that causes the scheduler's failure. In Graphplan terminology, this set of actions can be thought of as an additional n-ary action mutex constraint on the planner's CSP. We will now see that based on the specific scheduling class being used, this single translated action mutex constraint may lead to additional implied action mutex constraints.

In Section 4.2, we mentioned different scheduling classes, and how they affect the process of setting up the scheduling CSP. For class INFRES and FIX, since actions in the final plan are not allowed to move in the scheduling phase, the conflict set representing a failure in the scheduler is converted to exactly one action set in the planner. However, for class SAMELEN, because actions are allowed to *move* away from its initial position in the solution given by the planner, one conflict set of the scheduling CSP represents multiple action mutex sets of the planner. For example, if the set $(O_1, O_2, .., O_n)$ is one nogood of the scheduler, and each action O_i is allowed to take any position $s_i \leq p_i \leq e_i$ in the planner's CSP, then any one of the action sets $(O_1^{p1}, O_2^{p2}, .., O_n^{pn})$, $s_i \leq p_i \leq e_i$ in the planner's CSP will not be schedulable. Thus, for class SAMELEN, the translation process will convert each translated conflict set into multiple action mutex sets in the planner.

Generation of an Alternative Plan

Once the scheduler's failure explanation is translated into a set of nogoods for the planner, they can be used as additional action mutex constraints on the planner's CSP. We simply reinvoke the CSP solver to find another solution to the planner's CSP that respects these additional constraints, and pass them onto the scheduler. If no solution exists that satisfies the augmented set of constraints, then we know that there is no causal plan at the current plan length that can be scheduled. The planning graph is extended by another level, and the planning/ scheduling interaction is re-started.

5.3 Summary

In this section, we described how the failure information can be communicated from the scheduler back to the planner. The CSP formulation allows the planner to assimilate the feedback as additional constraints on its search. Empirical results with `RealPlan-PP` are shown in Section 7.3.

6 Post-processing the Scheduled Plan

In this section, we discuss post-processing steps that may be needed by 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 and reallocation actions were added by the scheduler, the scheduled plan has to be post-processed to replace generic freeing/reallocation actions with real actions in the domain.

6.1 Translating Inserted Actions

Domain translation corresponds to replacing resource freeing and reallocating 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 the current implementation or derived automatically from a domain modeling tool like TIM [17]. 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 A). 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 non-minimality as illustrated below.

```
(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⟩)))))))
```

Fig. 25. 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	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)

Fig. 26. The abstract plan (on left) is scheduled (on right) by inserting resource manipulating actions.

Consider the case of gripper domain [37] where balls have to be moved between rooms. The resource specification for gripper is given in Figure 25. It states that there is a 1-step plan to free a gripper and a 3-step plan to reallocate a gripper.

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)

Fig. 27. 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).

The left column in Figure 26 shows a causal plan for moving 4 balls from roomA to roomB. The abstract plan is scheduled (in right column of Figure 26) by inserting actions that assume that the gripper could be freed and reallocated at different levels where needed. The resultant plan is translated based on the resource specification of gripper in Figure 25 to produce the left plan in Figure 27. This plan is non-minimal and can be further post-processed with known justification techniques [15] to remove redundant actions. A justified plan is one that does not contain actions which are not necessary for achieving a goal. Post-processing also ensures that the translated plan is executable. This check is needed because though the planner suggested to the scheduler that resource freeing and reallocating actions are available in the domain, inserting those actions may interfere with actions already present in the plan.

In general, adding actions to a plan is risky because this can change its causal structure and lead to harmful 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. Domain translation as well as plan justification (specifically, *backward justification[15]*) have been fully implemented.

7 Implementation and Evaluation

The `RealPlan` approach of decoupling causal and resource reasoning provides multiple implementation choices in both solving the abstract planning problem and in scheduling resources which are highlighted in detail in this sec-

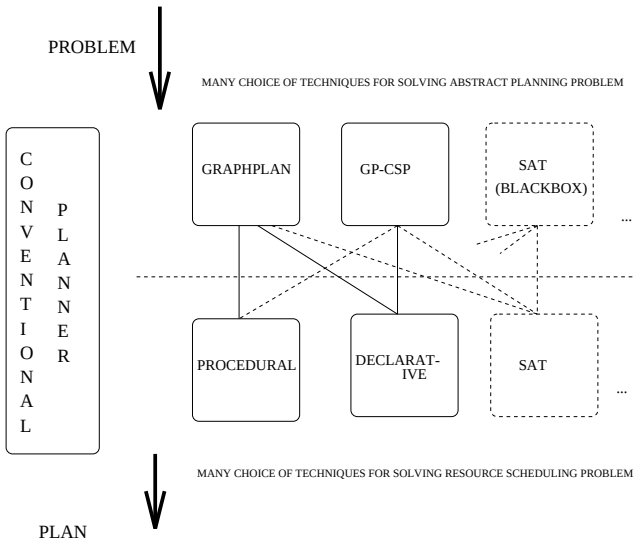

Fig. 28. Choices for causal and resource reasoning. Boxes with solid lines show choices that have been investigated.

tion. We consider `RealPlan-MS` where we investigate Graphplan planning followed by different forms of resource scheduling (procedural as in Section 4.2 or declarative as in Section 4.1) and show that `RealPlan-MS` gains in performance while being less sensitive to the quantity of resources. We also consider `RealPlan-PP` where GP-CSP is used along with declarative scheduling and show that `RealPlan-PP` can gain in performance over `RealPlan-MS` if the planner-scheduler interaction is appropriately controlled. Finally, we compare `RealPlan` with heuristic-based planners.

7.1 Implementation Choices

By separating causal and resource reasoning, multiple choices are available in the selection of methods for abstract planning and resource scheduling which are summarized in Figure 28. A prototype implementation of `RealPlan` has been completed on top of Graphplan where the causal plan is obtained by Graphplan and scheduling is handled by either procedural or declarative methods. A different planner has also been used, namely GP-CSP[10], which converts the plan graph of Graphplan into a CSP problem and extracts plans from it with a standard CSP solver, for causal reasoning along with 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*[15] 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.

7.2 Solving Problems with `RealPlan-MS`

Our evaluation mechanism is to compare the performance of the prototype with standard Graphplan, as the amount of resources is varied. First, 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) are used. We consider planning followed by different forms of resource scheduling (procedural or declarative) and `RealPlan` is shown to gain in performance while being less susceptible to the quantity of resources. Declarative methods are preferred in further experimentation, thereafter.

Next, the relationship between the nature of resources (sharable *vs.* non-sharable) and scheduling time is investigated. We first consider the *rocket* domain where a sharable rocket can be used to transport items between locations. To study the interplay between these types of resources in a domain, a new planning domain was 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). We focus on 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 a de-coupled approach. `RealPlan-MS` is found to be still quite useful.

7.2.1 Planning and Procedural Scheduling

In `RealPlan`, the planning time is constant and the scheduling time is dependent on the specific class (in Figure 18) that the problem falls into. The allocation classes are iterated in the following order: class INFRES, class FIX and class SAMELEN. Recall that allocation problems which need backtracking and class INCRLEN are solved by returning to normal planning.

Figure 29 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 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

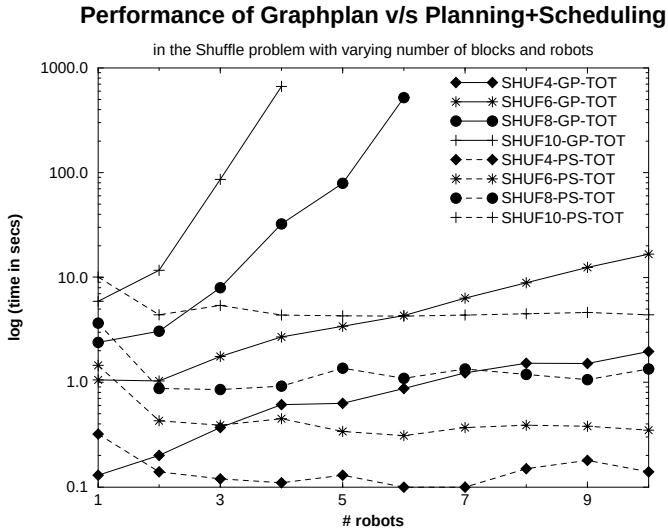

Fig. 29. Comparative performance on shuffle problems of 4, 6, 8 and 10 blocks with `RealPlan-MS` and Graphplan (Total: 80 problems).

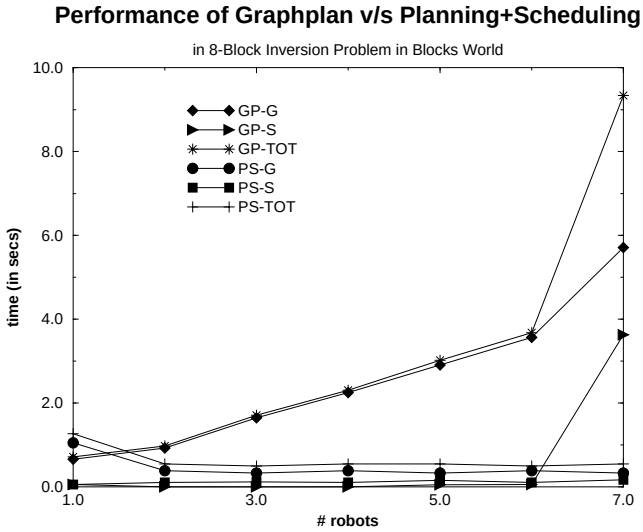

Fig. 30. Plot showing the performance of Graphplan along with Planning followed by Scheduling method for 8-block inversion problem (Total: 20 problems).

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, `RealPlan-MS` takes almost constant time whereas the performance of Graphplan is adversely impacted to a significant extent.

In Figure 30, we see the performance of the new 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 reiterates results in Figure 29.

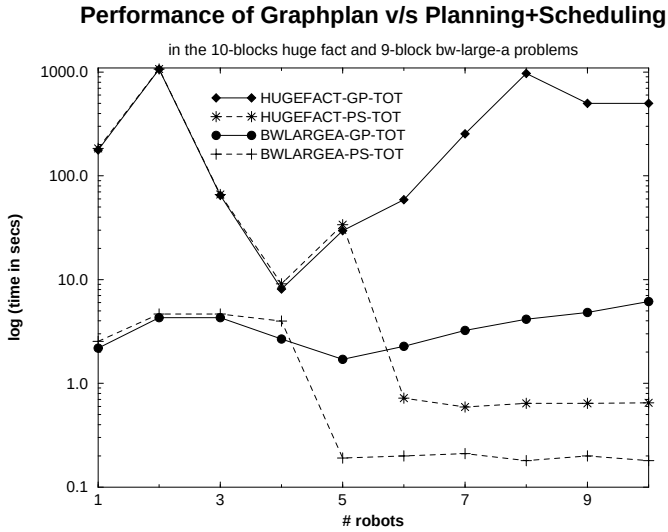

Fig. 31. Comparative performance on huge-fact (10 blocks) and bw-large-a (9 blocks) problems with `RealPlan-MS` and Graphplan (Total: 40 problems).

# 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
Runtime results from experiments in the logistics domain (in cpu sec). GP refers to Graphplan while GP+Sched refers to `RealPlan-MS`.

In Figure 31, we see the performance of the new 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. The new 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. Results similar to those in the *huge-fact* case are obtained. 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 19 cycles through all the methods for each problem.

Multiple resources & the Logistics domain: Note that our approach can

handle domains with multiple 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 in 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. In the declarative scheduling approach, we do not explicitly tell the CSP solver the order in which different types of resources should be scheduled but let it figure this out by itself as part of the CSP search process. To illustrate the multi-resource case, consider the results of experiments in the logistics domain, shown in Table 7. 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

Declarative scheduling approach has also been fully implemented on top of Graphplan and the early results are promising. Again we compare the performance of `RealPlan-MS` 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[55]. First, 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) are used.

In the new method, the causal reasoning time is constant and the resource reasoning time is dependent on the specific allocation policy (in Table 6) 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.

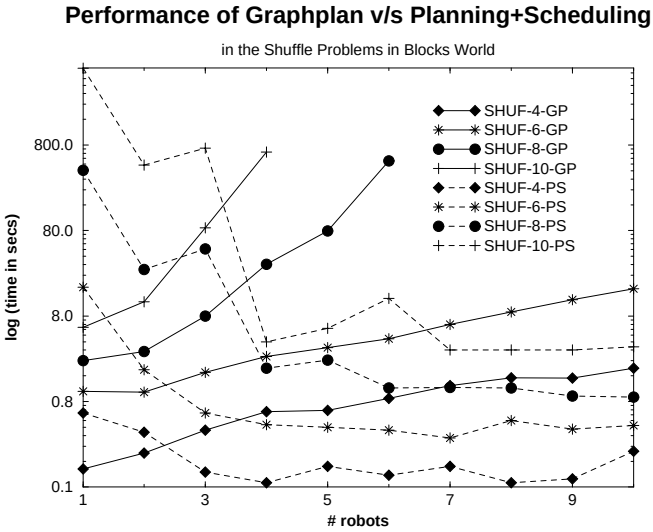

Fig. 32. Comparative performance of `RealPlan-MS` v/s Graphplan in *shuffle* problem of 4, 6, 8 and 10 blocks (Total: 80 problems).

Figure 32 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 the *6-shuffle* case, problems with 5 to 10 robots 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).

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 up front.

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 following the above order, but takes 81 minutes when class INCRLEN was specified up front.

In Figure 33, we see the performance of the new method on the 10-block

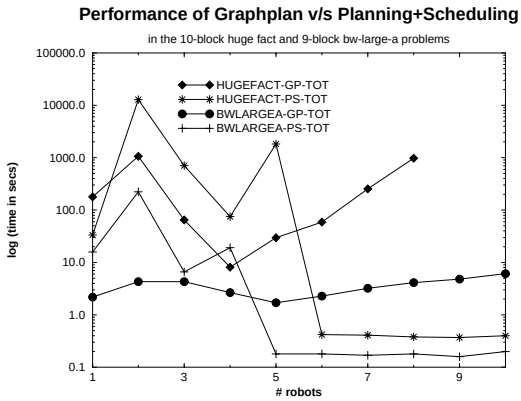

Fig. 33. Comparative performance on *huge-fact* (10 blocks) and bw-large-a (9 blocks) problems with Graphplan (Total: 40 problems).

# 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 8
Runtime results from experiments in the logistics domain (in cpu sec). GP refers to Graphplan while GP+Sched refers to `RealPlan-MS`.

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, the plan length is allowed to be increased linearly up to 10% of the current plan length before it is doubled. Across classes, we see that decoupling planning and scheduling leads to better results.

To illustrate the multi-resource case, consider the results of experiments in the logistics domain (described in Section 7.2.1), shown in Table 8. 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.

7.2.3 The Effect of Nature of Resources on Scheduling

Let us now investigate the relationship between the nature of resources (sharable *vs.* non-sharable) and (declarative) scheduling time. Consider the *rocket* do-

# 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 9
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.

main where the sharable rocket can be used to transport items between location. Table 9 shows the result of 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 `RealPlan-MS` (implemented in Graphplan) 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 `RealPlan-MS`) while average scheduling time is a mere 0.03 sec. This suggests that either the specific planner (i.e. Graphplan) is not handling the abstracted planning problem efficiently or that decoupling causal and resource reasoning is not beneficial for sharable resources for overall planning efficiency even though the scheduling time is still very small.

A different planner was also used, namely GP-CSP[10], which converts the plan graph of Graphplan into a CSP problem and solves it with a standard solver. The third column in Table 9 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 impact of the nature of resources on scheduling is further highlighted if there are non-sharable resources in addition to sharable resources. To study the interplay between these types of resources, a new domain was 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

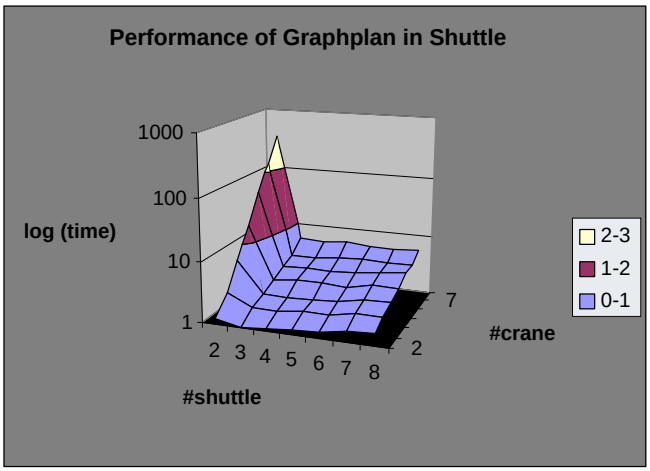

Fig. 34. Comparative performance of Graphplan in *shuttle* problems of 2..8 cranes and 2..8 shuttles. (Total: 49 problems)

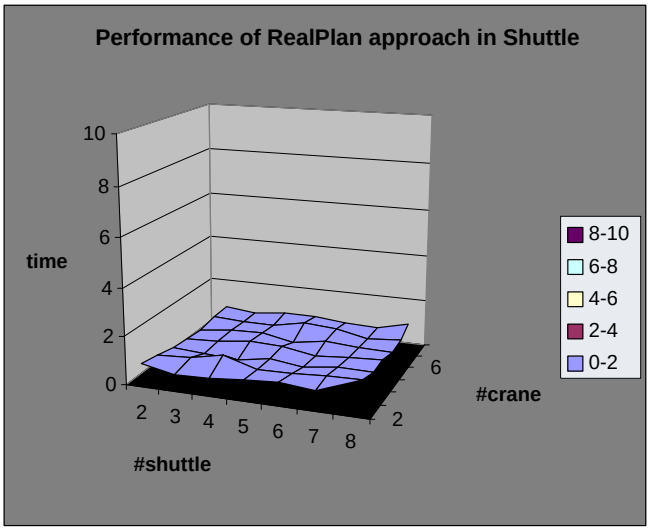

Fig. 35. Comparative performance of `RealPlan-MS` in *shuttle* problems of 2..8 cranes and 2..8 shuttles (Total: 49 problems).

Moon). We focus on problems where the number of both of these resources are varied independently. In Figure 34, the performance of Graphplan degrades sharply with the number of non-sharable cranes and less so with the number of sharable shuttles. In Figure 35, the performance of `RealPlan-MS` is plotted. We note that the run-time is fairly constant and much less than that of Graphplan with varying number of non-sharable cranes and sharable shuttles.

In this section, we demonstrate that the RealPlan-PP approach of supporting bidirectional interaction between the planner and the scheduler can serve as an effective approach for integrating planning and scheduling systems. Specifically, as our results show, it can be more effective than RealPlan-MS as well as normal planning.

Table 10 shows the comparison results of using the RealPlan-PP and two alternative methods. We tested problems from the multi-robot, multi-block blocksworld domain (with *r#* giving the number of robots and *b#* giving the number of blocks). The column titled *RealPlan-PP* shows the Class (INFRES, FIX or SAMELEN) that the experiment is conducted with, the time needed to solve the problem using the RealPlan-PP interaction framework, and the number of backtracks between the scheduler and the planner until we get the schedulable plan. The next column titled *BTPOF* (Back To Planner On Failure) shows the time in seconds if we do not use the interaction framework, but instead go back to the planner after the first time we fail and try to handle the resource constraints inside the planner by grounding all actions. The next column titled *RealPlan-MS* shows the result if we do not use the interaction approach, but let the scheduler run with its default setting. Thus, it will try all the scheduling classes (INFRES, FIX, SAMELEN, INCRLEN) in the mentioned order, and try to gradually increase the scheduling horizon in the INCRLEN class until the solution is found. Note that RealPlan-MS here uses declarative scheduling in the context of GP-CSP. The two subcolumns titled *class* and *time* show the last class that the default setting used to find the solution, and the running time in seconds. The next two columns show the speedups of the RealPlan-PP framework compared with the BTPOF and RealPlan-MS approaches. The last two columns titled *#robot* show the number of robots used in that tested problem compared with the number of robots needed to solve the scheduling problem without any interaction (i.e. in Class INFRES [10]). All the running times are in seconds. The planner-scheduler encodings and nogoods used for experiments in Table 10 follows the discussion in Section 5.

We see in Table 10 that RealPlan-PP has large speedup over the BTPOF approach (*BTPOF* column) and over RealPlan-MS (*RealPlan-MS* column) in almost all of the solved problems. The results also show that when resources are *critical*, i.e., there are not enough resources to assign to actions in the first

[10] The number of resources needed in Class INFRES depends upon the nature of the causal plan used to drive scheduling. Since GP-CSP is used for causal planning in these experiments while Graphplan was mainly used in Section 7.2, the causal plan may be different causing these numbers to be different from those of previous results.

	RealPlan-PP			BTPOF	RealPlan-MS		Speedup		#robot	
problem	class	time(s)	#bktrs	no interaction	class	time(s)	BTPOF	RealPlan-MS	used	limit
shuffle_b6r2	INFRES	-	>30	2.99	INCRLEN	39.06	-	-	2	3
shuffle_b6r2	FIX	0.29	2	2.98	INCRLEN	39.06	10.3x	135x	2	3
shuffle_b6r2	SAMELEN	1.70	1	3.00	INCRLEN	39.06	1.76x	23x	2	3
shuffle_b8r4	INFRES	2.59	4	>3 hrs	SAMELEN	9373	>4170x	3619x	4	5
shuffle_b8r4	FIX	8.74	1	>3 hrs	SAMELEN	9373	>1236x	1072x	4	5
shuffle_b8r3	INFRES	202	20	6858	INCRLEN	>3 hrs	33.95x	53.46	3	5
shuffle_b8r3	FIX	2.34	1	6861	INCRLEN	>3 hrs	2932x	>4615x	3	5
shuffle_b8r3	SAMELEN	94	1	6862	INCRLEN	>3 hrs	73x	>115x	3	5
shuffle_b8r2	INFRES	-	>30	>3 hrs	INCRLEN	>3 hrs	-	-	2	5
shuffle_b8r2	FIX	3.66	12	>3 hrs	INCRLEN	>3 hrs	>2951x	>2951x	2	5
shuffle_b8r2	SAMELEN	79	5	>3 hrs	INCRLEN	>3 hrs	>137x	>137x	2	5
huge-fact_r6	INFRES	4.53	4	976	INCRLEN	11056	215x	2441x	6	7
huge-fact_r6	FIX	17.48	3	991	INCRLEN	11056	56.7x	632x	6	7
huge-fact_r6	SAMELEN	20.59	2	983	INCRLEN	11056	47.7x	537x	6	7
huge-fact_r5	INFRES	33.93	19	701	INCRLEN	704	20.66x	20.75x	5	7
huge-fact_r5	FIX	6.95	6	700	INCRLEN	704	101x	101x	5	7
huge-fact_r5	SAMELEN	15.61	5	701	INCRLEN	704	44.9x	45.1x	5	7
bw-largea_r4	INFRES	-	>30	1.15	INCRLEN	0.47	-	-	4	5
bw-largea_r4	FIX	-	>30	1.15	INCRLEN	0.47	-	-	4	5
bw-largea_r4	SAMELEN	1.22	5	1.17	INCRLEN	0.47	0.96x	0.39x	4	5

Table 10

`RealPlan-PP` experiments. The "`RealPlan-PP`" columns show the class, the running time and the number of backtrackings between the planner and the scheduler for the `RealPlan-PP` framework. The BTPOF shows the running time if our policy is "Back to Planner On Failure" for that class. The "`RealPlan-MS` columns show the results for the default setting of running all the scheduling classes consecutively. The "class" column shows the first class in which a particular problem could be solved and the "time" column shows the running time in seconds. The "Speedup" column shows the speedup of using `RealPlan-PP` compared with BTPOF and `RealPlan-MS` approaches. The last column titled "#robot" shows the number of robot used in a particular problem, and the number of robot that are needed to solve the first solution from GP-CSP without interaction (i.e. Class INFRES).

plan given by the planner, it is efficient to fail at the earliest, go back to the planner with the reason of the failure, and find another (hopefully, easier to schedule) plan. Specifically, by not trying to increase the scheduling horizon and assigning resources for a very hard plan, the problem can be solved much faster by going back and finding another plan that can be handled by simpler classes. The speedup results over the `RealPlan-MS` approach can be up to more than 4900 times. Moreover, the interaction framework still retains the advantage of decoupling the resource assignment from the planning task. This fact is confirmed by the big speedup over the BTPOF approach by up to more than 4170 times.

The comparison results between using the `RealPlan-PP` interaction framework for different scheduling classes shows no clear cut winner between INFRES, FIX, and SAMELEN. As expected, due to the lesser flexibility of the simpler scheduling encoding strategy, class INFRES always requires more inter-module backtracks than class FIX, which in turn requires more than class SAMELEN in most of the cases. As a result, within the limit of 30 backtracks, class INFRES is only able to get the schedulable plan in 4 of the 7 cases. However, despite the higher number of interactions, class INFRES finds a solution in the fastest time in 2 of the 4 cases it can solve. Using class FIX, we can solve 6 of the 7 cases, and it is always faster than class SAMELEN. The number of backtracks in class FIX are also only slightly higher than class SAMELEN. Moreover, even when the number of backtracks needed in class FIX is more than 2 times higher than in class SAMELEN (12 to 5) in problem *shuffle_b8r2*, it still manages to be about 20 times faster, thanks to the simpler CSP encoding. Overall, even though class SAMELEN allows us to solve the most number of problems and class INFRES gives the fastest running times in some problems, class FIX seems to be the best fit for `RealPlan-PP`.

The only remaining exception in `RealPlan-PP` is that when the resources become very critical, then we may need too many interactions between the planner and scheduler before we get to the plan that can be schedulable. For example, if we try to solve the problem *huge-fact*, which needs 7 robots to schedule the first plan, with 1 or 2 robots, then our planner will go back and forth between two modules several hundreds times before finally run out of memory before finding the first schedulable plan. One possible solution to this problem is to limit the number of interactions between planner and scheduler. Currently, we use the upper bound of 30 for our experiments.

7.4 Heuristic State Search Planners and `RealPlan`

We next see the performance of two heuristic state search planners, FF and AltAlt, in comparison to `RealPlan`. All results are on a linux machine (PIII

	FF			AltAlt			Realplan-MS		
problem	time	#A	#R	time	#A	#R	time	#A	#R
shuffle_b6r3	0.03	14	3	0.12	14	3	0.09	16	3
shuffle_b6r5	0.06	12	5	0.23	12	5	0.08	12	5
shuffle_b6r10	0.23	12	5	0.70	12	5	0.08	12	5
shuffle_b6r20	1.57	12	5	2.74	12	5	0.09	12	5
shuffle_b6r30	4.61	12	5	6.74	12	5	0.10	12	5
shuffle_b6r40	11.71	12	5	13.40	12	5	0.12	12	5
shuffle_b6r50	21.40	12	5	35.41	12	5	0.15	12	5
shuffle_b6r60	39.78	12	5	114.22	12	5	0.17	12	5
shuffle_b6r70	68.40	12	5	241.88	12	5	0.20	12	5
shuffle_b6r80	105.19	12	5	729.34	12	5	0.24	12	5
shuffle_b6r90	155.46	12	5	OoM	-	-	0.28	12	5
shuffle_b6r100	244.06	12	5	-	-	-	0.32	12	5

Table 11
Performance of FF and AltAlt in the *shuffle* problem. `RealPlan-MS` is insensitive to the number of resources.

500Mhz, 516MB RAM) [11].

In Table 11, we see the performance of FF, AltAlt and `RealPlan-MS` planners in the *shuffle* problem. We see that `RealPlan-MS` performs the best and its result is insensitive to the number of resources.

In Table 12, we see the performance of these planners in the *shuttle* domain. Recall from Section 7.2.3 that the difference between the *shuffle* and the shuttle problem is that there are two types of resources (instead of one) in the latter. We see that `RealPlan-MS` performs the best while AltAlt scales better than FF in this domain.

Finally, we consider the problems from an external testbench [12] made available from parcPlan group [44]. In this blocks world domain, the table has finite block holding capacity causing the causal and resource reasoning phases to work closely together (tightly coupled). We compare FF and AltAlt with `RealPlan-PP` in Table 13. Note that FF scales quite well in this domain but its time still increases with the number of resources. A possible reason for its

[11] FF runs with default option. AltAlt runs with the ADJ-SUM2 PACTION NOLEVOFF options. OoM means *Out of Memory*.
[12] Available at *http* : *//www.icparc.ic.ac.uk/parcPlan/bw_pddl.tar.gz*.

	FF				AltAlt				Realplan-MS			
problem	time	#A	#C	#S	time	#A	#C	#S	time	#A	#C	#S
shuttle_f4c2s2	0.52	28	2	2	0.86	26	2	2	0.16	25	2	1
shuttle_f4c3s3	2.64	31	3	3	1.76	26	3	2	0.17	25	3	1
shuttle_f4c4s4	18.16	30	4	4	3.31	26	4	2	0.14	25	4	1
shuttle_f4c5s5	159.41	30	4	4	5.88	26	4	2	0.14	25	4	1
shuttle_f4c6s6	OoM	-	-	-	10.07	26	5	2	0.14	25	4	1
shuttle_f4c7s7	-	-	-	-	16.39	26	5	2	0.14	25	4	1
shuttle_f4c8s8	-	-	-	-	55.37	26	5	2	0.14	25	4	1
shuttle_f4c9s9	-	-	-	-	105.81	26	5	2	0.14	25	4	1
shuttle_f4c10s10	-	-	-	-	669.27	26	5	2	0.14	25	4	1
shuttle_f4c11s11	-	-	-	-	OoM	-	-	-	0.14	25	4	1

Table 12
Performance of FF and AltAlt in the Shuttle problems.

	FF			AltAlt			Realplan-PP		
problem	time	#A	#R	time	#A	#R	time	#A	#R
b6x6prob1_3r	0.02	12	3	0.25	12	3	0.34	14	3
b6x6prob1_5r	0.02	12	3	0.51	12	4	0.34	14	3
b6x6prob1_10r	0.03	12	3	1.60	12	4	0.34	14	3
b6x6prob1_20r	0.06	12	3	7.55	12	4	0.36	14	3
b6x6prob1_30r	0.10	12	3	58.92	12	4	0.37	14	3
b6x6prob1_40r	0.13	12	3	234.84	12	4	0.39	14	3
b6x6prob1_50r	0.16	12	3	974.01	12	4	0.42	14	3
b6x6prob1_60r	0.18	12	3	OoM	-	-	0.45	14	3
b6x6prob1_100r	0.33	12	3	-	-	-	0.61	14	3

Table 13
Result in the parcPlan blocks world domain with multiple robots and limited block positions on the table.

good result is that its "helpful action" technique may help to remove a lot of irrelevant actions. The performance of `RealPlan-PP` is almost constant while AltAlt also slows with more resources.

7.5 Summary and Lessons Learned

In this section, decoupling of causal and resource reasoning in planning was empirically shown as a promising idea for overall planning efficiency. The performance improvement in `RealPlan-MS` is more marked for non-sharable resources and when planning and scheduling are *loosely coupled*. Moreover, we demonstrated that using failure-driven learning of constraints as in `RealPlan-PP` can be more effective than `RealPlan-MS` as well as BTPOF approach. `RealPlan` also provides us choices for selecting different reasoners for causal and resource parts. We suggested that performance, and plan and schedule quality can be used in deciding the specific reasoners for assembling an efficient planner.

8 Discussion and related work

In this section, `RealPlan` is put in the context of project management activities in the commercial world, as well as planning and scheduling literature. For the most part, project management in the commercial world is performed with a standard tool like Microsoft Project[40] which is primarily an event scheduling tool. There is a rich body of work in classical planning [36], [45], [26], [6], [25]. In classical planning[1], time is atomic, actions are observable and deterministic, resources are not modeled separately from other objects in the domain and there is no reward for resource optimization. The primary focus is on obtaining a sequence of actions to achieve the goal state.

Scheduling has been studied widely in Operations Research (OR)[46] and Artificial Intelligence (AI)[61]. In AI, the resource allocation approaches are constraint based as in systems like OPIS[50], ISIS[16] and MICRO-BOSS [48] with very limited action selection choices, if any. Resources are richly modeled (in terms of the capacity of resources, the rate at which metric constraints are consumed, etc.) and resource optimization is the main objective.

8.1 Resource Definitions in Literature

There are a number of definitions of resources in the literature and corresponding resource reasoning approaches. SIPE [56] defines a resource as anything for which two actions contend. 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. The distinction between resource and non-resource objects is not clear. Knoblock [30] uses similar ideas

to specify a database as a resource for some action if the domain modeler wants to prevent one operator from executing 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[38]. 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 execution time resource constraints do bring real world constraints into planning which is a natural extension of the problem. IxTeT [33] 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 [42] allows it to consider most of its physical components as resources and schedules them for their optimal usage. Planning and resource constraints are converted to a set of common data-structures and search is 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.

8.2 Comparing `RealPlan` with Project Management

As mentioned in Section 1, AI Planning can handle plans that are small compared to what humans themselves handle in the real world. It would be interesting to compare planning and scheduling activities performed in project management with `RealPlan`. Recall that humans devise the Work Breakdown Structure (WBS)[39] 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[40] 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 the resources assigned to them are no longer overallocated.

Following are some of the allowed strategies for shortening a schedule in Mi-

crosoft Project and how they relate to the policies implemented in our (automated) approach:

(1) Add lead time (task starts before the predecessor finishes) or lag time (task starts after the predecessor finishes). This refers to our resource allocation policy of serializing the plan.
(2) Decrease unnecessary work of a resource on a task. The resource allocation 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 `RealPlan`.
(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 `RealPlan`.
(5) Increase working condition i.e. capacity of each resource to accomplish more. This is not an option in `RealPlan` because the initial state (and initial resources) are considered unchangeable.
(6) Reduce the scope of a task by reducing the amount of work assigned to the task. This could potentially be implemented in `RealPlan` by introducing non-primitive operators (also called Hierarchical Task Network planning[13]) where a non-primitive operator can be decomposed in multiple ways. It is not an option in `RealPlan` with primitive operators because the domain operators are considered unchangeable.
(7) For allocated tasks,
    (a) Increase the number of resources allocated to a task. In the examples we considered, actions usually have a fixed (single) resource requirements.
    (b) Increase resource availability over time. Resources cannot be changed in `RealPlan`.

One can also reduce the cost of the project schedule. Total cost of a schedule is the sum of the cost of resources allocated to that project along with 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 since the costs of resources (e.g., compensation of personnel) are generally different which is not usually the case in `RealPlan`. On the other hand, scheduling by increasing the length of the plan is a viable option in `RealPlan` but not usually allowed in commercial projects due to increased costs or loss of business opportunities.

8.3 Mixed Planning and Scheduling Systems

According to a recent survey on planning and scheduling[51], difficult practical problems in automated reasoning lie in between the scope of the two areas and are far from resolved. As mentioned in the introduction, planning systems have incorporated continuous resources like time and fuel by additionally employing time and resource map managers to ensure resource consistency (SIPE[56], IxTeT[33], IPP[32], LPSAT[57]). But such an integration explodes the search space for the planner beyond action sets that are minimal with respect to the logical goals because actions may be added to achieve the resource goals that are not necessary for logical goals. To control search, either expressivity is restricted or performance degradation due to slower flaw resolution is tolerated.

The recent LPSAT planner by Wolfman and Weld[57] distinguishes between discrete and continuous state variables, pushing the assignment of continuous ones to an simplex-based LP solver like Cassowary [5]. Note that discrete/continuous distinction is really orthogonal to resource/non-resource distinction. Abstraction of resources can be applied to both continuous and discrete parts of LPSAT. A natural extension of resource scheduling will be the handling of metric constraints which is useful for real-world tasks like resource planning, temporal planning and optimization[31], [57].

8.4 Planner Scheduler Integration in `RealPlan`

`RealPlan` envisages resource allocation to be de-coupled from planning, and is handled in a separate "scheduling" phase. 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 causal plan 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 [8, pp. 73], has identified the inefficiency of combining resource scheduling with planning (although, to our knowledge, no specific steps were taken to address that inefficiency in the O-Plan work).

A crucial factor for `RealPlan` is planner-scheduler interaction. In `RealPlan-MS`, the planner and scheduler communicate in terms of policies that the scheduler interprets in terms of its variables, their domains and constraints. 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.

Another way of looking at the planner-scheduler interaction is by having the scheduler "explain" the reason for its failure to allocate sufficient resource. In `RealPlan-PP`, we essentially perform a type of "multi-module dependency directed backtracking" approach that is a variation on the hybrid planning methodology developed in [23], and is also akin to the approach used to link satisfiability and linear programming solvers in [57].

In Sadeh et al[49], a blackboard based architecture called IP3S is presented for integrating process planning and production scheduling. However, the system relies on the user to select different process plan in the event of scheduling failure while in `RealPlan-PP`, the planner can automatically generate a different plan.

8.5 Distributed Constraint Satisfaction Problems and `RealPlan`

Distributed Constraint Satisfaction Problem (Dist-CSP) is a CSP-based technique for communication between distributed agents. The integration between planner and scheduler in our problem is different from (and is orthogonal to) Dist-CSP [59],[60], in the way we set up the separate CSP encodings. In the Dist-CSP, the variables and constraints of a large system are distributed between agents. The encoding for each agent is defined up front, and we have to get a consistent set of solutions for all local CSPs. However, in our system, the CSP encoding for the scheduling problem can *only* be defined after we get the solution from the planner. Notice that we can also use the Dist-CSP structure to encode our planning-scheduling problem. To do so, the planner's and scheduler's CSP encodings (PE and SE) are built up front, with the logical variables and constraints being included in the PE, and the resource-related variables and constraints being put into the SE. However, because we do not know which set of actions will be logically consistent (and can be a potential solution of the PE), we have to suppose that any action in the graph can appear in the solution, and have to include the resource-related variables and constraints for it in the SE.

There are certain advantages of choosing our current framework of generating the CSP encoding for the scheduling problem upon the plan given by the planner over the Dist-CSP approach. First, normally, the number of actions in the final plan is much smaller than the number of actions in the plan graph. Therefore, the CSP encoding generated from the solution given by the planner will be much smaller than the one generated in the Dist-CSP approach. Second, our experiments show that the planning and scheduling problems are actually *loosely coupled*, meaning that there will likely not be many inter-

actions between the two modules. Therefore, we will not get much benefit from the Dist-CSP approach in saving the time of generating different CSP encoding for the scheduling problem. Third, naturally, the relations between the variables in two modules (planner and scheduler) are not equivalent. The *variables* in the scheduler correspond to the *values* in the scheduler, but it is not even a one-to-one relation. One variable in the SE corresponds to a set of values of different variables in the PE. This fact makes it more reasonable to generate and solve the planning part first before we go to the scheduling part.

8.6 Separating Resource Reasoning for Efficiency

Among planners that have considered resources, in SIPE[56], domain-specific operator ordering can be provided by defining what are resource objects in the domain. Work more closer to `RealPlan` is by El-Kholy and Richards[12] and Cesta and Cristiano[7] who perform temporal and resource reasoning after a plan is obtained. They however do not consider the interactions between resource allocation and planning phases. In [35], planning has been separated into action selection and action sequencing activities, and the latter is expanded to scheduling. In contrast, we consider causal reasoning as planning, and resource reasoning as scheduling. Specifically, the causal plan has selected actions along with sequencing information that is independent of resource considerations whereas resource reasoning adds additional sequencing constraints.

Fox and Long[18] 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 resources are abstracted out in the presented approach.

There exist methods for improving the performance of Graphplan by removing irrelevant literals from the problem specification (c.f. [43]). Such methods however are not applicable for us since resources – however many of them there may be – are never irrelevant and in fact facilitate the state transition of desired objects.

Explanation-based learning (EBL) and dependency directed backtracking (DDB) techniques have been used by Kambhampati[22] to expedite Graphplan. Though these methods capture some of the regularities of the domain/problem, we found that they are still not competitive with `RealPlan`. Finally, the complexity of changing plans for scheduling and parallelization has also been studied by Backstrom [3]. While he focuses on parallelizing a complete and correct plan, `RealPlan` starts with a maximally parallel resource-abstracted plan and

add or shift actions across levels to handle resource constraints.

8.7 Relation to Plan Abstraction

Abstraction and least commitment has been widely studied in the context of planning [13], [25]. In the context of making planning efficient, `RealPlan` can be seen as the abstraction of resources from planning to the 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 was shown in Section 4.1.

From the abstraction angle, the 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[29]. 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, `RealPlan-PP` supports interaction between the scheduler and planner to arrive at a schedulable plan.

9 Concluding Remarks and Future Work

The current work is motivated by the desire to exploit the loose-coupling between planning and scheduling in real world domains. A novel planning framework, `RealPlan`, was developed in which resource allocation is de-coupled 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. Resources were described and using the infinite resource assumption, it was showed that disregarding resources during planning and subsequently scheduling resources can lead to increased performance. Specifically, we considered `RealPlan-MS` where we investigated causal planning with Graphplan followed by different forms of resource scheduling, and showed that `RealPlan-MS` is efficient while being less sensitive to the quantity of resources. We also considered `RealPlan-PP` where GP-CSP is used along with declarative scheduling and showed that `RealPlan-PP` can gain in performance over `RealPlan-MS` if the planner-scheduler interaction is appropriately controlled. While we focused on Graphplan, our approach can be easily extended to other planning regimes.

The runtime of `RealPlan` is much less sensitive to the resource quantity than monolithic planning. It thus admits the paradigm of *plan once and schedule*

any time. If some allocated resource becomes unavailable during plan execution, the approach can handle the exception through resource re-allocation.

9.1 Limitations

In the current work, resources were modeled as single (non-sharable) or infinite (sharable) capacity resources in the current work. Most deployed planning systems allow modeling and reasoning with finite capacity resources. The operators and objects in the domain model can be enhanced to specify the finite capacity of resources and resource profiles can be used to reason with such resources. Optimistic and pessimistic resource profiles were used by O-Plan [8] to identify resource contention. We generate resource profiles while encoding the causal plan for resource allocation (see for example, right column in Figure 15) and can use them for similar resource reasoning.

Resource optimization has been addressed to a limited extent in the current work. The value ordering scheme of GAC-CBJ solver[55] tries values from one side of a variable's domain until the whole domain is exhausted. This ensures that an already freed resource is used before any other resource. For example, the solution of the *shuffle* problem with 10 robots uses 5 robots – the maximum robots needed to overcome all resource conflicts (refer to Figure 15). This limitation can be removed by posing the resource allocation as a standard optimizing CSP problem.

A related issue is that resource optimization is not rewarded in planning problems where there are only goals of achievement. The set of valid solutions for most problems can contain plans requiring significantly different number of resources. Since there are higher scheduling costs involved in generating the plans that require fewer resources, the user's need in this regard should be expressible in the specification of goals.

9.2 Future Extensions

For most multi-resource domains, resources can be identified by the domain writer. One can incorporate a domain modeling tool (e.g. TIM[17]) 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.

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

As identified in Section 1[52], decoupling planning and scheduling can benefit not only Graphplan-style state-space planning but also goal-directed planning as in UCPOP[45], and also form the framework for planning with metric and continuous resources. Only discrete sharable and non-sharable resources were handled in this work. One can incorporate continuous resources by modeling linear constraints and solving them with with linear programming techniques. Such interesting extensions would increase the scope of real world planning problems that can be solved.

Acknowledgments

We wish to thank Prof. van Beek for help with his CSP library and solvers. We also thank the other members of the Yochan group for many fruitful discussions on understanding and extending the ideas of `RealPlan`. Terry Zimmerman's help in proof-reading the earlier drafts is particularly appreciated.

This research is supported in part by an NSF young investigator award (NYI) IRI-9457634, ARPA/Rome Laboratory planning initiative grant F30602-95-C-0247, AFOSR grant F20602-98-1-0182, NSF grant IRI-9801676 and a NASA Cross-enterprise technology initiative grant.

References

[1] Allen, J., Hendler, J., and Tate, A. (ed.). *Readings in Planning.* Morgan Kaufmann Publ., San Mateo, CA. 1990.

[2] Bacchus, F. *AIPS-00 Planning Competition Results.* At http://www.cs.toronto.edu/aips2000/SelfContainedAIPS-2000.ps. 2000.

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

[4] Beck, J.C., and Fox, M. *A Generic Framework for Constraint-directed Search and Scheduling.* AI Magazine 19(4). 1998.

[5] Borning, A., Marriott, K., Stuckey, P., and Xiao, Y. *Solving linear arithmetic constraints for user interface applications.* In Proc. 1997 ACM Symposium on User Interface Software and Technology. October 1997.

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

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

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

[9] Frost, D., and Dechter, R. 1994. Dead-end driven learning. *Proc. AAAI-94.*

[10] Do, B., and Kambhampati, S. *Solving Planning Graph by Compiling it into CSP.* Proc. AIPS-00. 2000.

[11] Do, B., Srivastava, B., and Kambhampati, S. *Multi-Module Dependency Directed Backtracking for Integrating Planning and Scheduling Systems.* Technical Report. Dept of CS. ASU. August 2000 (forthcoming).

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

[13] Erol, K. *Hierarchical Task Network Planning: Formalization, Analysis and Implementation.* Ph.D. Dissertation, Dept. of Computer Science, Univ. Maryland, USA. 1995.

[14] 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.

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

[16] Fox, M. ISIS*: A Retrospective.* In [61], pages 3-28. 1994.

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

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

[19] Hoffman, J., and Nebel, B. *The FF Planning System: Fast Plan Generation Through Heuristic Search.* Submitted to JAIR. Also website at http://www.informatik.uni-freiburg.de/ hoffmann/ff.html. 2000.

[20] Kambhampati, S. Planning Graph as (dynamic) CSP: Exploiting EBL, DDB and other CSP Techniques in Graphplan. *Journal of Artificial Intelligence research (JAIR).* 2000.

[21] Kambhampati, S. On the Relations between Intelligent Backtracking and Failure-Driven Explanation-based Learning in Constraint Satisfaction and Planning. *Artificial Intelligence.* 1999.

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

[23] Kambhampati, S., Cutkoksy, M.R., Tenenbaum, J.M. and Lee, S. *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.

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

[25] Kambhampati, S., Mali, A., and Srivastava, B. *Hybrid planning for partially hierarchical domains* In Proc. AAAI-98. July 1998.

[26] 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.

[27] 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.

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

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

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

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

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

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

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

[35] Liatsos, V. and Richards, B. *Scaleability in Planning.* Proc. ECP-99. 1999.

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

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

[38] 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.

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

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

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

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

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

[44] parcPlan Group *parcPlan Home Page.* http://www.icparc.ic.ac.uk/parcPlan/index.html. 1999.

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

[46] Pinedo, M. *Scheduling Theory, Algorithms and Systems.* Prentice Hall. 1995.

[47] Prosser, P. *Domain Filtering can Degrade Intelligent Backjumping Search.* Proc. IJCAI-93, 262-267. 1993.

[48] Sadeh, N. *Micro-opportunistic Scheduling: the Micro-boss Factory Scheduler.* Technical Report CMU-RI-TR-94-04, Carnegie Mellon Robotics Institute. Also in [61]. 1994.

[49] Sadeh, N., Hildum, D., Laliberty, T., McA'Nulty, J., Kjenstad, D., and Tseng, A. *A Blackboard Architecture for Integrating Process Planning and Production Scheduling.* Concurrent Engineering: Research and Applications, Vol. 6, No. 2. 1998.

[50] Smith, S. OPIS*: A Methodology and Architecture for Reactive Scheduling.* Intelligent Scheduling, M. Zweben and M. Fox, ed., Morgan Kaufmann Publ., San Mateo, CA. 1994.

[51] Smith, D. E., Frank, J. and Jonsson, A. K. *Bridging the Gap Between Planning and Scheduling.* Knowledge Engineering Review. 1999.

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

[53] Srivastava, B. Effective planning by Efficient Resource Reasoning. *Ph.D. Dissertation. Arizona State Univ., USA.* 2000.

[54] Srivastava, B. *RealPlan: Decoupling of Causal and Resource Reasoning in Planning.* Proc. AAAI-00. 2000.

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

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

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

[58] Yochan. AltAlt Web Page. *http://rakaposhi.eas.asu.edu/altweb/altalt.html.* 2000.

[59] Yokoo, M. and Hirayama, K. Algorithms for Distributed Constraint Satisfaction To appear in *A Review Autonomous Agents and Multi-Agent Systems.* 2000.

[60] Yokoo, M. and Hirayama, K. Distributed Constraint Satisfaction Algorithm for Complex Local Problems. *Proc. ICMAS-98.* 1998.

[61] Zweben, M., and Fox, M. (ed.). *Intelligent Scheduling.* Morgan Kaufmann Publ., San Mateo, CA. 1994.

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
```

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>)))))))
```