RealPlan: Decoupling Causal and Resource Reasoning in Planning

Biplav Srivastava
Email: biplav@asu.edu
Department of Computer Science and Engineering
Arizona State University, Tempe, AZ 85287-5406.

Abstract

Recent work has demonstrated that treating resource reasoning separately from causal reasoning can lead to improved planning performance and rational resource management where increase in resources does not degrade planning performance. However, the resources were scheduled procedurally and limited to cases that could be solved backtrack-free. Terming the decoupled framework as RealPlan, in this work, I extend it with a general approach to convert the resource allocation problem as a declaratively specified dynamic constraint satisfaction problem (DCSP), compile it into CSP and solve it with a CSP solver. By doing so, the resource scheduling problem can be handled in its full complexity and can provide a computational characterization of the different scheduling classes. The CSP formulation also facilitates planner-scheduler interaction by helping the scheduler interpret the resource allocation policies proposed by the planner in terms of constraints on values of scheduling variables. Moreover, if the extraction of causal plan is also formulated as a CSP problem, the two CSPs can enable dependency directed backtracking between them. I have implemented declarative scheduling on top of Graphplan and GP-CSP planners (which poses the backward search of Graphplan as a CSP problem), and the resulting planners reiterate the benefits of decoupling planning and scheduling while providing elegant CSP models (RealPlan-MS, RealPlan-PP) for investigating planner-scheduler communication.

Introduction

AI Planning 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*. Humans come up with the Work Breakdown Structure (WBS)(Moder & Phillips 1964) 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 and meet deadlines. Project management tools like Microsoft Project(Microsoft 1998) help in sequencing the task network using Critical Path Method (CPM) or Program Evaluation and Review Technique (PERT) analysis.

In contrast, most AI planners do not distinguish between causal and resource reasoning and handle them within the same planning algorithm. Experimental results (Srivastava & Kambhampati 1999) show that this strategy severely curtails the scale-up potential of existing planners, including such recent ones as Graphplan(Blum & Furst 1995) and Blackbox (Kautz & Selman 1998). In particular, these planners exhibit the seemingly irrational behavior of worsening in performance with increased resources. The key observation is that 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. Most planners suffer performance drop due to the expanded flaw resolution.

In our recent work (Srivastava & Kambhampati 1999; Srivastava 2000), we demonstrated that treating resources separately from causal reasoning can lead to improved AI planning performance and rational resource management (for example, planning performance does not worsen with increased resources). However, the resources were scheduled procedurally and only those cases were handled that could be solved backtrack-free. Terming the decoupled framework as RealPlan, in this work, I extend it with a general approach to convert the resource allocation problem into a declaratively specified dynamic constraint satisfaction problem (DCSP(Mittal & Falkenhainer 1990)), compile it into CSP and solve it with a standard CSP solver. By doing so, the full resource scheduling problem can be handled with all its complexity. I provide a computational characterization of the scheduling classes presented in (Srivastava & Kambhampati 1999) in terms of specification of this CSP. The CSP formulation also helps the scheduler to interpret the resource allocation policies proposed by the planner in terms of constraints on values of scheduling variables.

Figures 1 and 2 provide a general overview of the RealPlan approach. The unified framework accepts a domain description along with optional annotations for resources[1], finds a plan modulo the choice of resource abstraction, and then allocates resources to produce a sound final plan (if the plan requires resources). After planning is

[1] The primary focus here is on reusable discrete resources which may be sharable or non-sharable.

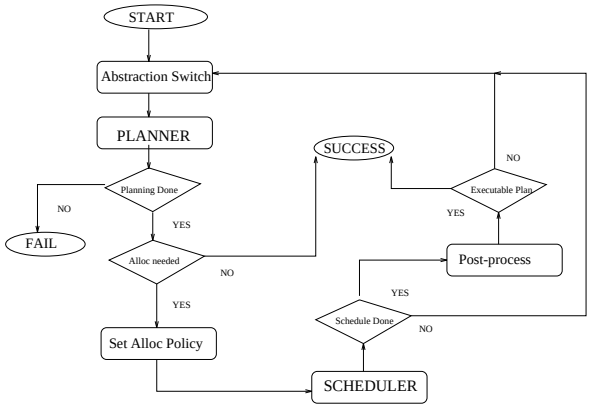

Figure 1: *RealPlan: Unified planning-scheduling framework*

1. Identify resources.
2. If no resource information is available or resources are so low (usually one) that postponing their reasoning is counter-productive, perform conventional planning (i.e. in this case, 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 scheduling criteria.
5. Return a valid final plan. As long as the algorithm ensures that all facts achieved during the planning phase are not undone by resource scheduling, the final plan is sound.

Figure 2: *Synopsis of RealPlan approach.*

complete, a scheduler can decide which resources to actually allocate based on resource allocation policies proposed by the planner. The different allocation policies include maintaining the concurrency of the plan, serializing the plan and inserting actions to free and reallocate the resources[2]. If freeing/ reallocating actions are allowed, the problem is in fact a dynamic constraint satisfaction problem (DCSP) because these new actions (variables) control the normal action variables.

I have implemented the declarative scheduling (hereafter, referred to as only scheduling) on top of the Graphplan(Blum & Furst 1995) and the GP-CSP(Do & Kambhampati 1999) planners. Planner-scheduler interaction is supported as *master-slave* relationship (called RealPlan-MS, see Figure 3) in Graphplan and GP-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 further and destroyed. Further, 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

[2]The policy of inserting freeing/reallocating actions assumes that actions are reversible in the domain. It could otherwise lead to incompleteness and should be disabled.

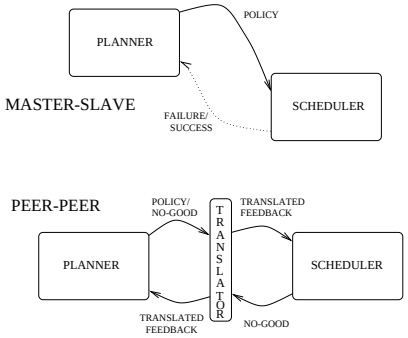

Figure 3: *Communication relationships between the planner and the scheduler. RealPlan-MS is the main focus here whereas RealPlan-PP is currently being developed.*

executability. If all the allocation policies lead to failure or inexecutable plans in a domain, this implies that planning and scheduling were infact, *not loosely coupled* in this instance. In such a case, the framework retains the ability to switch off resource abstraction and resort to traditional planning.

The framework can also support *peer-peer* relationship (called RealPlan-PP) between the planner and the scheduler which is being developed in the GP-CSP context (Srivastava, Kambhampati & Do 2000). In GP-CSP, the plan graph of Graphplan is converted into a CSP problem and solved with a standard solver. Using such a planner, the scheduler can inform the planner about the source of infeasibility in terms of the variables and constraints in the planner's CSP to handle even *tightly coupled* problems. This type of "multi-module dependency directed backtracking" approach is a variation on the hybrid planning methodology developed in (Kambhampati et al 1991), and is also akin to the approach used to link satisfiability and linear programming solvers in (Wolfman & Weld 1999).

Here is the outline of the paper. First, the specification of the constraints for the resource scheduling problem as a dynamic constraint satisfaction problem is described. Next, I discuss planner-scheduler interaction in this context. Through empirical results, we show the effectiveness of the decoupled approach vis-a-vis the nature of resources. The paper concludes with a discussion of related work and comments.

Scheduling as a declarative CSP

For the purpose of illustration, the "*shuffle*" problem, which is the multiple robots version of the 6-block *blocks_facts_shuffle* problem in the Graphplan system, is selected. Here, a stack of blocks has to be systematically shuffled to achieve the goal arrangement. An example of the plan generated for the *shuffle* problem, by disregarding inter-resource conflicts during planning, is shown in Figure 4. The plan consists of 10 time steps (levels) with resource profile of the number of resources left allocated at each level shown in the right column (marked "#Robots"). In our experiments, the problem is generalized to k-block *shuffle* versions so that

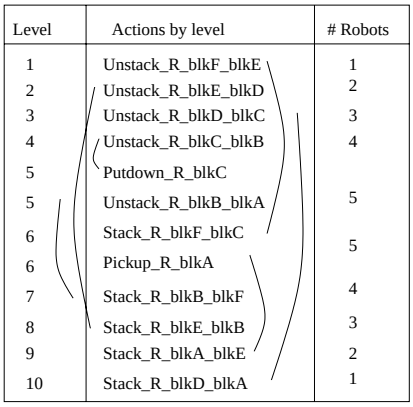

Figure 4: *A resource-abstracted solution for* shuffle *problem. Curved lines show resource usage spans (see below). The number of resources needed at each level (which equals the number of spans crossing that level) is also shown.*

the problems can also be scaled independent of the number of resources.

A Constraint Satisfaction Problem (CSP(Beck & Fox 1998)) 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(Mittal & Falkenhainer 1990)), 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.

Let us state the resource allocation problem for resource R. The abstract plan has a set of action pairs $\langle A_i, A_j \rangle \mid j \succ i$ where action A_i appears at time step i of the plan (actually written as A_i^m if it is the mth action at level i using resource R but the superscript is omitted for clarity) and they constitute resource spans ($S_{ij} : \langle A_i, A_j, C \rangle$ s) that we have to allocate resources to. The effect C of action A_i is produced at level i and consumed at a later level j for the precondition of action A_j.

Examples of spans in Figure 4 are $S_{1,6}$: $\langle A_1, A_6^1, holding\_R\_blockF \rangle$ and $S_{2,8}$: $\langle A_2, A_8, holding\_R\_blockE \rangle$. We note that the nature of problem is such that every resource allocation choice is a backtrackable point. Moreover, actions can move to lower or upper levels if causal dependencies allow them.

Each action A_i using resource R has two variables associated with it, RA_i for the resource allocated and PA_i for the position or level where the action will appear. Position

Action	Vars	Possible Values
A_i	$\langle RA_i, PA_i \rangle$	$\{1..N\}$,$\{i..L\text{-}1\}$
A_j	$\langle RA_j, PA_j \rangle$	$\{1..N\}$,$\{j..L\}$
F_{ij}	$\langle RF_{ij}, PF_{ij} \rangle$	$\{\perp,1,..N\}$,$\{\perp,i{+}1..L\text{-}2\}$
U_{ij}	$\langle RU_{ij}, PU_{ij} \rangle$	$\{\perp,1,..N\}$,$\{\perp,i{+}2..L\text{-}1\}$
N_i	$\langle PN_i \rangle$	$\{i..L\}$

Table 1: *Constraints on action variables and their values while scheduling for resource R. Number of resource of type R are N and the permitted length of the plan is L. $\perp \Rightarrow F_{ij}$, U_{ij} are not needed. N_i is R insensitive.*

Relationship among variables	Comments
$RA_i = RF_{ij} \vee (RF_{ij} = \perp \wedge RA_i = RA_j)$	If freeing action is needed, it uses the same resource as span starting the action
$RA_j = RU_{ij} \vee (RU_{ij} = \perp \wedge RA_i = RA_j)$	If realloc action is needed, it uses the same resource as span ending the action
$RF_{ij} \neq \perp \Leftrightarrow RU_{ij} \neq \perp$	If freeing action occurs, reallocating action also occurs and vice-versa
$PF_{ij} \prec PU_{ij}$ $\vee\ PF_{ij} = PU_{ij} = \perp$	Position of freeing action is before position of realloc action or both are NULL
$PA_i \prec PF_{ij} \vee PF_{ij} = \perp$	Position of freeing action is after start of span or is NULL
$PA_j \succ PU_{ij} \vee PU_{ij} = \perp$	Position of realloc action is before end of span or is NULL
$RF_{ij} = \perp \Leftrightarrow PF_{ij} = \perp$	If freeing action is not needed, its position is NULL and vice-versa
$RU_{ij} = \perp \Leftrightarrow PU_{ij} = \perp$	If realloc action is not needed, its position is NULL and vice-versa
$PA_i \prec PA_j$	Position of action starting a span is before the action ending it
$PN_i \prec PN_j$, $PN_i \prec PA_j$, $PA_i \prec PN_j$	Relative ordering of actions in the plan is maintained irrespective of resource usage
Non-sharable resource constraints (see Table 3)	If segments of two spans overlap, they cannot share resources over that segment

Table 2: *Relationship among values of action variables.*

of an action is also a variable because one way to allocate resources, given a resource limit, is by serializing the parallel plan. Actions that do not participate in manipulation of resources are noted as N_i and their corresponding position variable is PN_i. Given a span S_{ij}:$\langle A_i, A_j, C \rangle$, two additional actions are associated with it, F_{ij} for freeing the resource and U_{ij} to reallocate the resource. The constraints on variables and legal values are listed respectively in Table 1 and Table 2.

As an example, for span $S_{1,6}$, the domain of RA_1 = RA_6 = $\{1..7\}$ when 7 is the number of robots in the problem. The domain of position variables are PA_1 = $\{1..12\}$ and PA_6 = $\{6..12\}$ if the allocation policy permits movement of actions in the plan until level 12. If free/ reallocation actions are not allowed, $RF_{1,6}$ = $RU_{1,6}$ = $PF_{1,6}$ = $PU_{1,6}$ = $\perp$. Such an allocation policy in fact allows the scheduler to serialize the planner till level 12 (beyond the abstract plan length of 10 in Figure 4) without adding any new action.

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

Table 3: *INTERACT(a,b,c,d) = ($a \leq d \wedge c \leq b$). When two sections of resource spans interact, the interacting sections cannot share the same resource. The superscript refers to the spans S_1 or S_2 for which the actions (and variables) are applicable.*

The constraints on resource values enforce that the resource used by A_i is the same as A_j unless the freeing and reallocating actions are present. If they are present, A_i and F_{ij} have the same resource as 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 are summarized in Table 2.

Moreover, if a resource is non-sharable (meaning single capacity), additional constraints have to be specified as summarized in Table 3. The gist of the constraints is that if any segment of a span interacts with that of another, the two spans cannot share a resource. For example, spans $S_{1,6}$ and $S_{2,8}$ interact between levels 2 and 6. Therefore, they cannot share a robot (resource) in this interval unless their allocated robots are freed. Freeing (and reallocating) actions will result in sub-intervals over which a robot cannot be shared.

Finally, in addition to the constraints in Tables 2 and 3, we have the top-level constraints:

- The number of resource allocations at a level must not exceed the available resources.
- To optimize the plan, we can set the objective function as minimizing the total number of actions in the plan and/or amount of resources used. Minimizing resources is usually neglected in AI planning because the number of resources in a problem is part of the initial specification and there is no incentive for saving them.

The resource allocation problem can now be solved by any systematic method. We solve the CSP encodings with GAC-CBJ, a CSP solver in CPLAN(van Beek & Chen 1999) that performs generalized arc-consistency and conflict directed backjumping.

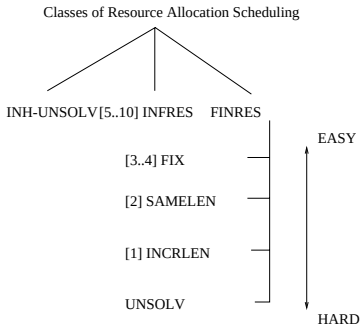

Figure 5: *A classification of resource allocation instances (with resource quantities that put* 6-shuffle *problem in each of the classes). INH-UNSOLV refers to causally infeasible plan for which no scheduling is needed, while UNSOLV refers to an unschedulable plan.*

Policies for Planner-Scheduler Interaction

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

In (Srivastava & Kambhampati 1999), the resource allocation problem was classified into a variety of classes (see Figure 5). The complexity of resource scheduling instance increases from left to right and from top to bottom. It was proposed that rather than using one general scheduling method for all classes, one could cycle through the scheduling methods tailored to each of the specific classes. However, the implementation avoided Class INCRLEN and backtracking cases in other classes, and reverted to normal planning.

The CSP formulation allows the different classes to be fully supported in the form of resource allocation policies. The different policies include maintaining the concurrency of the plan, serializing the plan and inserting actions to free and reallocate the resources. Table 4 summarizes the different policies and what they imply in terms of legal values of variables. Maintaining concurrency of the plan corresponds to all actions A_i in the plan being immovable while no freeing/ reallocating actions are permitted. The domain of RA_i is the range of available resources. Serializing the plan implies that the action of the plan can move subject to an upper plan length, L^{MAX}, provided by the planner. Again, no freeing/ reallocating actions are permitted to be inserted. An example of L^{MAX} is the number of actions in the plan, which allows the plan to be completely serialized.

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

In introducing resource freeing/reallocating actions, three sub-cases are identified . If actions are considered immovable, this corresponds to Class FIX. Here, the freeing action (F_{ij}) can be introduced immediately after A_i while the reallocating action (U_{ij}) can come immediately before A_j. The second sub-case is when the actions are allowed to move upto the length of the abstract plan, and this corresponds to Class SAMELEN. Finally, the actions are allowed to move till any upper limit L^{MAX} ($L^{MAX} \succ L$) in Class INCRLEN.

The advantage of multiple allocation policies is that it helps the planner in communicating the plan preference 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.

Experiments

The aim of the experiments is to show that the declarative scheduling method is not only general but also makes the overall planning efficient vis-a-vis the nature of resources. Implicitly, it also tests if the *master-slave* form of relationship (RealPlan-MS) is effective. We now compare the performance of the approach (as implemented in Graphplan and GP-CSP) to standard Graphplan, when one varies the amount of sharable/ non-sharable resources. I consider the blocks world (robots), the rocket domain (rocket) and the shuttle domain (cranes and shuttles) with varying numbers of resources noted in the parenthesis respectively.

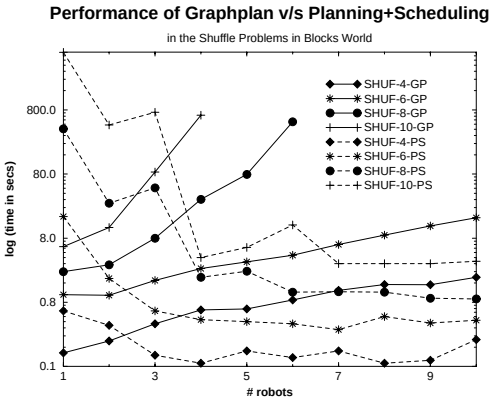

Figure 6: *Comparative performance of my approach of decoupling causal and resource reasoning v/s Graphplan in* shuffle *problem of 4, 6, 8 and 10 blocks. (Total: 80 problems)*

Figure 6 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--GP). The 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 RealPlan, the causal reasoning time is constant and the resource reasoning time is dependent on the specific allocation policy (in Table 4) that successfully allocated the resources. For fair comparison, since Graphplan only looks for shorter length of the plan while the serializing allocation policy prefers both shorter length as well as fewer number of actions in the plan, this policy is disabled. The allocation policies are iterated in the following order: class INFRES, class FIX, class SAMELEN and finally class INCRLEN. In the *6-shuffle* case, problems with 5 to 10 robots 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.

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). 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. In the case of single resource, resource postponement is useless in transferring planning complexity to scheduling and is infact counter-productive, because the planner is banking on concurrency in the plan while resource availability suggests a serial executable plan. This pathological case could have been easily detected and avoided upfront.

Utility of the scheduling classes: The idea of progressively

# 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 5: 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. (Total: 21 problems)

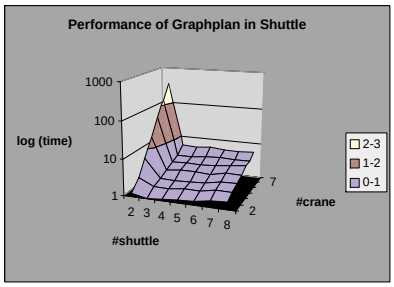

Figure 7: Comparative performance of Graphplan in *shuttle* problems of 2..8 cranes and 2..8 shuttles. (Total: 49 problems)

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

More results are available in (Srivastava 2000). Let us now investigate the relationship between the nature of resources (sharable v/s non-sharable) and declarative scheduling time. Consider the *rocket* domain where the sharable rocket can be used to transport items between location. Table 5 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. 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 the new approach is much higher. It turns out that the causal reasoning in the space of abstracted plans takes an average of 2.38 sec (note that causal reasoning is constant for the decoupled approach) while average scheduling time is mere 0.03 sec.

The third column in Table 5 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 is being solved by GP-CSP more efficiently than Graphplan.

The scenario is highlighted if there are non-sharable resources in addition to sharable resources. To study the

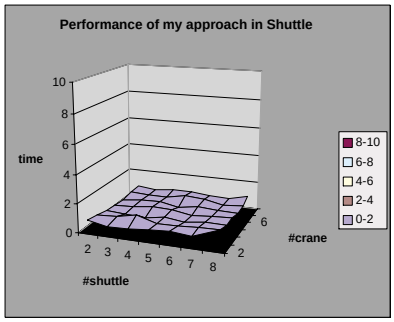

Figure 8: Comparative performance of my approach in *shuttle* problems of 2..8 cranes and 2..8 shuttles (Total: 49 problems).

inter-play between these types of resources, I created a domain called the *shuttle* domain. In this domain, there are sharable shuttles and non-sharable cranes to move boxes between inter-stellar bodies (e.g. Earth and Moon). Problems where the number of both of these resources are varied independently are considered. In Figure 7, we see the performance of Graphplan which degrades sharply with the number of non-sharable cranes and lesser so with the number of sharable shuttles. In Figure 8, the performance of the new approach is shown. We note that run-time is fairly constant and much lesser than Graphplan with varying number of non-sharable cranes and sharable shuttles.

Discussion and related work

Scheduling has been studied widely in Operations Research (OR)(Pinedo 1995) and Artificial Intelligence (AI). In AI, the resource allocation approaches are constraint-based as in systems like OPIS, ISIS and MICRO-BOSS (each summarized in (Zweben & Fox (ed.) 1994)) with very limited action selection choices, if any.

The work on O-Plan (Currie & Tate 1991, pp. 73), has identified the inefficiency of combining resource scheduling with planning (although, to my knowledge, no specific steps were taken to address that inefficiency in the O-Plan work). Among planners that have considered resources, in SIPE(Wilkins 1988), domain-specific operator ordering can be provided by defining what are resource objects in the domain. In IxTeT (Laborie & Ghallab 1995) and HSTS (Muscettola 1994), planning and resource constraints are converted to set of common data-structures and search applied to get a plan. In these systems, planning has been extended to include specification about physical resource usage and this increases expressivity but does not defer flaw resolution. We conjecture that performance degradation with increasing resources will also be seen in these systems.

Work more closer to RealPlan are *parc*Plan (El-Kholy & Richards 1996) and TRP(Cesta & Cristiano 1996) where temporal and resource reasoning is performed after a plan is obtained. In (Liatsos & Richards 1999), 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.

Restricted table blocks world from parcPlan[3]: Experiments done in some problems from the restricted table blocks world domain of (Liatsos & Richards 1999) ("arm" is a resource) showed that causal and resource reasoning interact closely here. For smaller problems, e.g. with 4 blocks, 4 table position and 2 arms (b4x4x2), RealPlan-MS performs comparable to Graphplan (in a fraction of seconds). For medium problems, e.g. b6x6x3 and b8x8x4, the performance could suffer if the initial plan is quite parallel because the scheduling cost increases with fewer resources.

These problems are being experimented with RealPlan-PP being studied in (Srivastava, Kambhampati & Do 2000) where the scheduler attempts only cheaper allocation policies and on failure, passes the "failure" explanation back to the planner for re-planning. Initial results show that this approach can solve the medium problems (b6x6x3 and b8x8x4) in drastically less time.

Conclusion

Decoupling of causal and resource reasoning can lead to a big performance edge in planning. To this end, the RealPlan framework allows advances to be made in the two components as well as in planner-scheduler interaction. In this work, I presented a general approach to convert the resource reasoning problem as a declaratively specified DCSP and solved it with a standard CSP solver. The approach is not only more general than previous procedural scheduling methods but also supports intelligent planner-scheduler interaction. In Graphplan and GP-CSP, the *master-slave* relationship (RealPlan-MS) is implemented while in GP-CSP, a truly dependency-directed *peer-peer* relationship (RealPlan-PP) is envisaged. The runtime of RealPlan is much less sensitive to the resource quantity available. Infact, RealPlan-MS admits the paradigm of *plan once and schedule anytime*.

Acknowledgements

I thank Subbarao Kambhampati for his guidance and comments, and BinhMinh Do for discussions on GP-CSP and inter-CSP interactions. I also thank Prof. van Beek for putting the CPLAN planning system and its CSP solvers in public domain, and answering some of my questions. Support for this work comes in part by NSF young investigator award (NYI) IRI-9457634, ARPA/Rome Laboratory planning initiative grant F30602-95-C-0247, Army AASERT grant DAAH04-96-1-0247, AFOSR grant F20602-98-1-0182 and NSF grant IRI-9801676.

References

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

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. *Proc IJCAI-95* 1636–1642.

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

Currie, K. and Tate, A. 1991. O-Plan: the open planning architecture. *AI, Vol 52, 49-86*.

Do, B., and Kambhampati, S. 1999. Solving planning graph by compiling it into CSP *To Appear in AIPS 2000*.

El-Kholy, A. and Richards, B. 1996. Temporal and Resource Reasoning in Planning: the *parc*Plan approach. *Proc. ECAI-96*.

S. Kambhampati, M.R. Cutkoksy, J.M. Tenenbaum and S. Lee. Integrating General Purpose Planners and Specialized Reasoners: Case Study of a Hybrid Planning Architecture. *IEEE Trans. on Sys., Man and Cyber., Vol. 23, No. 6, Nov/Dec, 1993*.

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

Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. *Proc. IJCAI-95*.

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

Microsoft. 1998. Microsoft Project Version 4.0 User Guide. *Microsoft Press*.

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

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

Muscettola, N. 1994. Toward real-world science mission planning. *Proc. AAAI Fall Symposium*.

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

Srivastava, B. March 2000. Efficient Planning by Effective Resource Reasoning. *Ph.D. Dissertation. Arizona State Univ., USA*.

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

Srivastava, B, Kambhampati, S. and Do, B. 2000. Planning the Project Management Way: Efficient Planning by Effective Integration of Causal and Resource Reasoning. *Technical Report. Arizona State Univ., USA.*

van Beek, P., and Chen, X. 1999 CPlan: A constraint programming approach to planning *Proc. AAAI-99*.

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

Wilkins, D. E. 1988. Practical planning: Extending the classical AI planning paradigm. *Morgan Kaufmann Pub., San Mateo, CA.*

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

[3] At http://www.icparc.ic.ac.uk/parcPlan/ecp99/index.html.