Completable Scheduling: An Integrated Approach to Planning and Scheduling

Melinda T. Gervasio and Gerald F. DeJong

Beckman Institute for Advanced Science and Technology
University of Illinois at Urbana–Champaign
405 N. Mathews Ave., Urbana, IL 61801
gervasio@cs.uiuc.edu dejong@cs.uiuc.edu

Abstract

The planning problem has traditionally been treated separately from the scheduling problem. However, as more realistic domains are tackled, it becomes evident that the problem of deciding on an ordered set of tasks to achieve a set of goals cannot be treated independently of the problem of actually allocating resources to the tasks. Doing so would result in losing the robustness and flexibility needed to deal with imperfectly modeled domains. Completable scheduling is an approach which integrates the two problems by allowing an a priori planning module to defer particular planning decisions, and consequently the associated scheduling decisions, until execution time. This allows a completable scheduling system to maximize plan flexibility by allowing runtime information to be taken into consideration when making planning and scheduling decisions. Furthermore, through the criterion of achievability placed on deferred decisions, a completable scheduling system is able to retain much of the goal–directedness and guarantees of achievement afforded by a priori planning. The completable scheduling approach is further enhanced by the use of contingent explanation–based learning, which enables a completable scheduling system to learn general completable plans from example and improve its performance through experience. Initial experimental results show that completable scheduling outperforms classical scheduling as well as pure reactive scheduling in a simple scheduling domain.

Introduction

The planning problem has traditionally been treated separately from the scheduling problem. Planning deals with the determination of an ordered set of actions for achieving a set of goals. In the context of scheduling domains, planning deals with determining an ordered set of tasks for a set of jobs. In contrast, scheduling deals with the actual assignment of tasks to machines and is generally concerned more with finding the best of several alternative task–machine assignments than with finding a particular task–machine assignment. As more realistic scheduling domains have been addressed, however, it has become apparent that planning and scheduling cannot be treated independently. The complexity of real–world domains makes perfect characterizations difficult to construct and often unwieldy. To this end, researchers in both planning and scheduling have investigated reactive approaches which allow for decision–making during execution [Agre87, Firby87, Kaelbling88, Muscettola90, Ow88, Prosser89].

However, the classical approach of first doing planning and then scheduling still remains a problem. Consider giving a classical system in a process planning domain the job of manufacturing a particular part. Its planner must decide a priori on an ordered set of actions or operations which will result in the conversion of the raw material into the desired product. Its scheduler is then given the responsibility of actually allocating resources and carrying out the operations on the machines. However, because the planner commits the system to a particular set of operations, the scheduler may not execute the best plan. For example, the planner may not be able to guarantee that the efficient new milling machine will be available and so choose the older, slower one. However, during execution, the more efficient machine may turn out to be available, but the scheduler does not have the option to alter the plan. Furthermore, an over–constrained classical plan may prevent quick fixes to unpredictable runtime situations. For example, a chosen drill bit may turn out to be unavailable, thus rendering invalid those subsequent actions involving a correspondingly sized bolt and wrench. A simple fix would be to use a different drill bit, and switch to the appropriately sized bolt and wrench, but a scheduler with a completely–determined plan does not have this capability.

A purely reactive approach, with no a priori planning, has its own problems. Most manufacturing domains are fairly well–behaved; there is much information available a priori and fairly accurate predictions can be made about the behavior of the world under particular circumstances. A purely reactive approach which performs no projection cannot take advantage of this information to constrain its actions and prevent thrashing. With the planning problem and the scheduling problem combined, the runtime decision–making problem also becomes a larger and more complex one.

This research began as an attempt to address the problems with classical, a priori planning and pure reactivity. In particular, completable planning was developed as an approach which combined the goal–directedness and provably correct plans of classical planning with the flexibility and ability to utilize runtime information afforded by reactive planning. This enabled a completable planner to more efficiently deal with the problems arising from imperfect a priori information while still retaining the benefits of planning beforehand in relatively well–behaved environments. More recently, we have been investigating scheduling, and we have found that many of the techniques originally developed as part of the completable approach to planning are also useful for solving some of the problems which arise from scheduling in realistic domains where perfect a priori information about the environment is unavailable. For example, in the scheduling scenario above, a completable scheduling system could defer the deci-

sion of which milling machine to use as well as the choice of bit, bolt, and wrench sizes. During execution, it can then use additional information regarding resource availability to address the deferred planning decisions and make the associated scheduling decisions as well.

In this paper, we present an integrated approach to planning and scheduling called completable scheduling. We will first give an overview of the main ideas behind completable planning, and then discuss the extension to scheduling. We will then discuss how completable schedules are learned through an explanation–based learning strategy called contingent EBL. Finally, we will briefly discuss the implementation, including some preliminary results and ongoing experiments.

COMPLETABLE SCHEDULING

Overview of Completable Planning

In completable planning [Gervasio90a, Gervasio90b, Gervasio91], a classical planner is augmented with a reactive component which provides it with the ability to defer planning decisions until execution time. As an augmented classical planning approach, completable planning retains the advantages of classical planning while buying into the advantages provided by reactivity. From classical planning, completable planning borrows the ability to construct provably–correct plans for providing goal–directed behavior. From reactive planning, it borrows planning flexibility and the ability to utilize runtime information in making planning decisions. Completable planning achieves the integration through the achievability criterion, which requires every deferred goal to be proven achievable. Proving achievability requires proving that there exists a plan which will achieve the goal. Our research has shown that proving the existence of a plan does not necessarily entail determining the plan itself, and the intuition is that proving achievability requires much simpler and more readily available a priori knowledge than full–blown planning. A completable planner is thus able to construct provably–correct plans in spite of incomplete a priori information, and in doing so provide goal–directedness to its reactive component while allowing itself to defer decisions and utilize runtime information in addressing deferred goals.

Deferring Decisions

The deferment of scheduling decisions is a powerful tool in dealing with imperfect a priori information. The complexity of real–world domains makes it difficult to construct perfect models. Even when perfect models exist, their use often exceeds reasonable computational bounds. A realistic scheduling system is thus left to contend with imperfect knowledge. There are four types of incompleteness which can result from using classical planning techniques on imperfect information.

First, a schedule may be incomplete due to an unspecifiable parameter setting. With the lack of a fine–grained and tractable world model, a scheduling system may not be able to determine precise parameter settings a priori. For example, the parameters of an operation may be dependent on the properties of a particular object, which may not be known prior to execution. In attaching two parts using a bolt, all that a system may know is that it will be given a bolt of some size, but it may not know precisely what size. However, provided it has access to differently–sized bits and wrenches, it can plan a drilling operation followed by a bolting operation without specifying the precise bit and wrench to use. During execution, when it is given the bolt, it can then determine the appropriate values for bit size and wrench size. Completable scheduling allows the use of conjectured variables to act as placeholders for unspecified parameter settings provided achievability proofs can be constructed for their eventual instantiation. By allowing deferred parameter settings, completable scheduling enables a system to both plan ahead and yet remain flexible enough to deal with some uncertainty.

Second, a schedule may be incomplete due to an undeterminable number of iterations. An imperfect world model may include incomplete characterizations of operations and their effects. Consequently, for an action that requires repetition to achieve some goal, the precise number of iterations needed may not be determinable prior to execution. For example, the depth to which a milling operation cuts through a part is dependent on the face cutter used. Prior to execution, a system may not know which face cutter will be set up on the machine. However, it knows that regardless of which face cutter is used, the desired cut can be achieved by simply repeating the milling operation as many times as necessary. By not tying itself to a particular face cutter and consequently a particular number of iterations, during execution the system can choose to use the current set–up and save the cost of changing set–ups, or it can elect to change to a more efficient set–up. Completable scheduling permits the deferment of iteration decisions provided incremental progress which converges to the goal can be proven. Through this deferment, a completable system can use imperfect operation descriptions as well as make optimizations to a schedule based on runtime information.

Third, a schedule may be incomplete due to an unidentifiable operation choice or task–machine assignment. This case arises when there are multiple ways of achieving the same goal from different states and the system lacks the necessary a priori information for identifying which particular state will be reached. This case also arises when there are multiple ways of achieving a goal, with different situations resulting in different preferences among the various alternatives, and the system does not know a priori which situation will be reached. For example, in planning to shape an object, a system might use some or all of various cutting operations, such as milling, planing, sawing, or grinding. Whether there are several possible states requiring different operations or multiple applicable operations with unknown preferences, a system can use additional runtime information to make a more–informed operation choice. Completable scheduling allows a system to defer operation choice provided it can prove that there exists a way to reach the next state regardless of which of the possible states is reached. This deferment is useful for two reasons. First, it enables a system to use the same schedule to achieve a goal from any of several different states. Second, it allows a system to apply preferences to a set of possible operations using more complete and accurate runtime information.

Fourth, a schedule may be incomplete due to an unorderable set of operations. Imperfect a priori information may result in insufficient constraints for completely ordering a set of operations. For example, in the construction of two parts, the only precedence constraints may be between the milling, drilling, and tapping operations for each part—i.e. the operations for the different parts can be ordered in any way. Depending upon a priori known factors such as the parts involved and the difficulty of changing set–ups as well as a priori unknown factors such as the initial set–up and machine availability, particular orderings will be more desirable than others. By deferring the decision until all the factors are known, a system can utilize runtime information to make decisions for more optimal orderings. Completable scheduling permits the deferment of ordering decisions provided the different orderings are all capable of achieving the goal. In doing so, a completable planner can utilize runtime information in making more–informed ordering decisions for an unconstrained set of actions.

Proving Achievability

While imperfect a priori information is the primary reason for deferring decisions, achievability is the primary criterion for deferment in completable scheduling. By requiring that a deferred goal be proven achievable, completable scheduling enables the construction of incomplete yet provably–correct plans. Previous work on achievability involved finding proofs for the existence of plans to achieve deferred goals. Achievability proofs for deferred parameter settings and number of iterations are discussed in [Gervasio90a, Gervasio90b], and for deferred operator choice in [Gervasio91]. In [Gervasio91], completable planning was also extended to probabilistic domains by relaxing the original criterion of absolute achievability to probable achievability.

Scheduling domains give rise to further new issues in achievability. In planning, the main focus is finding a plan, or sequence of actions, which achieves the goal from a given initial state. In scheduling, the existence of several possible schedules is taken as a given, and the focus is choosing one from among them using some set of preference criteria, maximizing particular performance measures. Examples of performance goals are meeting deadlines and minimizing idle time. Thus, simply defining a goal to be achievable if there exists a plan for it is insufficient for scheduling. Achievability must also be related to the idea of optimization and relative preferences between possible courses of action. For example, proving the achievability of the goal associated with an unordered set of actions is implicit in the construction of a nonlinear plan—i.e. actions are left unordered if there are no constraints requiring precedence relations between them. Thus there exists a plan for achieving the goal. However, there is the interesting issue of deciding on a complete ordering during execution. This involves seeking out additional information for evaluating the different options as well as carrying out the operations themselves. In tying the concept of achievability to optimization, we can also better investigate a primary motivation for combining classical and reactive techniques: the ability to utilize runtime information in planning. Goal–directed, robust behavior in the face of uncertainty is one reason for augmenting a classical planner with reactive abilities. However, another reason to integrate the two approaches, is to take advantage of the wealth of information which becomes available at runtime. This additional information facilitates planning by helping to focus the search for an appropriate action.

LEARNING COMPLETABLE SCHEDULES

Explanation–based learning [DeJong86, Mitchell86] has been demonstrated to be useful in improving the performance of various planning systems [Bennett90, Chien89, Fikes72, Hammond86, Minton85], and in [Gervasio90a, Gervasio91] we present an explanation–based learning strategy called contingent EBL for learning completable plans. Learning completable schedules basically involves learning to distinguish between a priori planning decisions and decisions which have to be made or are better made during execution. Learning when to defer decisions involves first identifying the deferred decision, then constructing an achievability proof for the associated deferred goal. Then a completor for making the deferred decision during execution must be incorporated into the learned general plan.

Identifying Deferred Decisions

A main difference between classical plans and completable plans is the existence of deferred decisions in completable plans. In constructing an explanation for how a given training example achieves a target goal, an EBL system must explain how each action is chosen for execution. In planning, this usually means verifying that previous actions achieve the preconditions necessary for the execution of an action. However, with the addition of reactive abilities and the option to utilize runtime information, a system needs to distinguish between a priori satisfied preconditions and runtime–verified preconditions. Our solution is to allow the system to distinguish between a priori information and runtime–gathered information and to prefer a classical proof of correctness to an explanation of achievability. Thus, in explaining how an action is chosen for execution, a system first attempts to explain its preconditions with a priori available information. If this is unsuccessful, then the action being explained is tagged as a potential deferred decision, and the system attempts to construct an achievability explanation for the precondition. Only if it is successful is the learning process allowed to continue. The final explanation will thus contain the identified deferred decisions as well as their supporting achievability explanations.

Tying the concept of achievability to optimization adds further concerns. An explanation of executability is no longer enough. Explanations for preferences may also need to be constructed, and as with other deferred decisions, the associated runtime verified conditions need to be distinguished from a priori satisfied conditions. As with proofs of correctness and explanations of achievability, explanations of preferences may also be constructed in standard EBL fashion.

Constructing Achievability Proofs

To construct provably–correct plans, a completable planner must construct achievability proofs for the deferred goals of

its incomplete plans. While the mechanics of constructing proofs of correctness vs. proofs of achievability are essentially the same—both use standard EBL on a given domain theory—there are some requirements needed for a domain theory to be used in proving achievability.

There are four types of deferred decisions and each requires particular kinds of information for proving achievability. First, deferred parameter settings must be represented, and this is done using conjectured variables. These variables may only be introduced in the context of the rules used to construct their corresponding achievability explanations, thus guaranteeing that every conjectured variable in an explanation has a supporting achievability proof. Second, a system must be able to reason about the incremental progress achieved by a repeated action. This requires action characterizations to include statements regarding the changes made with respect to some measurable quantity. This can then be used to reason about progress towards the goal. Third, the incompletely known situation requiring a deferred operator choice must be represented in such a way that the system can reason about the space of possibilities. Achievability can then be measured in terms of the coverage provided by the alternative actions over this space. Finally, proving achievability with respect to an unordered set of operations is implicit in the absence of precedence constraints between the operations, which means that any of the possible total orderings will achieve the goal.

The second aspect of achievability, optimality, also imposes certain requirements on the domain theory used to construct explanations. The heuristics to be used in making dispatching or scheduling decisions must be built in to the domain theory. These heuristics can then be used both for constructing a priori explanations and making runtime decisions. In explaining particular decisions made in a training example, a system can then construct explanations incorporating the heuristics and learn general completable plans which will employ the heuristics in future applications.

Incorporating Completors

The final step in learning how to construct a completable schedule is to incorporate completion steps into the learned general plan. There are four types of completors corresponding to the four different types of deferred decision. The first, a monitor, finds a value to replace a conjectured variable—i.e. it determines a specific parameter setting. The second, a repeat loop, repeatedly executes an action until a particular exit condition, the deferred goal, is reached. The third, a conditional, evaluates the current state and determines an appropriate action based on which conditions are satisfied. Finally, the fourth, a dispatcher, determines a complete ordering for an unordered set of operations, based on a given set of heuristics. The achievability proofs constructed for the deferred decisions addressed by these completors are incorporated into the explanations supporting the learned plan. Thus the achievability conditions guaranteeing the existence of a completion are also in the learned plan. Provided these conditions, along with other preconditions, are satisfied in future instances, a completion is guaranteed to be found for the incomplete plan yielded by the learned general plan.

IMPLEMENTATION

A simple scheduling domain theory has been constructed to compare the performance of a completable scheduling system with that of a purely classical scheduling system as well as a purely reactive scheduling system. The domain involves a single machine which can be set up in various ways, each set-up of which is capable of performing some set of tasks. The same task may take different processing times on different set-ups. Furthermore, there is a set-up cost involved in changing set-ups. A job consists of a partially ordered set of tasks, and a scheduling problem involves a set of independent jobs. Initially, the only ordering constraints between tasks are based on deadlines. However, additional precedence constraints may be imposed between the tasks of a job if the a priori planning module of the system determines that one task is needed to establish the preconditions for another task. Uncertainty enters into the picture through an unknown initial state—i.e. the system does not know a priori which set-up will be on the machine when it starts executing its plan. Finally, the goodness of a schedule is measured by the length of time taken by the system to finish a set of jobs.

Preliminary results show that a completable system's ability to adapt to varying initial states enables it to construct more efficient plans/schedules than a classical scheduling, which commits itself to specific set-ups and complete task orderings prior to execution. Furthermore, the completable system needs less time both to learn a general completable schedule as well as construct a specific completable schedule, although it does incur the additional cost of runtime plan completion. The completable system is also able to construct more efficient plans than a reactive system because it is more focused in its search for an applicable action, having determined as many precedence constraints between tasks as it can prior to execution. Although both use the same heuristics for choosing between multiple applicable actions, the reactive system has the additional burden of sorting out precedence relations between tasks during execution. Furthermore, although the completable system initially needs to construct a completable schedule, the use of learning helps reduce the a priori planning cost it incurs over the reactive scheduler. We are currently running experiments to gather more data about the performance of the three approaches given different distributions and different machine/set-up/task-processing profiles. The results are expected to help identify particular domain and problem characteristics which favor the different approaches.

SUMMARY AND CONCLUSIONS

This work integrates planning—the determination of an ordered set of tasks—and scheduling—the assignment of those tasks to resource—through completable plans. Because completable plans are incomplete, additional planning is necessary during execution, when scheduling has begun to dispatch the tasks. Thus, this work differs from reactive approaches, such as those discussed in [Ow88, Prosser89, Smith90, Zweben90], where planning is separated from scheduling, and the main approach to uncertainty in the environment is to replan when the constraints of the original plan are violated. While replanning is a valuable tool which any real system will even-

tually need, our work first focuses on constructing plans which are as flexible as possible to minimize the need for failure recovery. In this sense, it is similar to ideas presented in [Drummond90, Martin90]. Drummond and Bresina present an algorithm for maximizing the probability of goal satisfaction in the case of actions with different possible outcomes, which is one of the problems the conditionals in completable scheduling address. Martin and Allen also prove the achievability of goals deferred to the reactive planner, but they do so using empirical methods, in contrast to the explanation–based methods we use. Completable scheduling may also be viewed as a shallow hierarchical planner, where runtime decisions are at the lowest level. However, unlike other hierarchical planners and schedulers, such as ABSTRIPS [Sacerdoti74], MOLGEN [Stefik81], and ISIS [Fox84], a completable scheduling system uses the achievability constraint to guarantee completability at lower levels. The ordered monotonic hierarchies of ALPINE [Knoblock90] are a similar idea. The difference is that ALPINE performs abstraction based on the deletion of literals, while in proving achievability completable scheduling uses explicitly more general or abstract knowledge regarding the deferred goals and their properties.

The idea of deferred decisions is not a novel one—the least commitment principle is a basic foundation of nonlinear planning, for example. What completable scheduling does is extend the least commitment principle to execution time and in doing so, achieving a well–founded integration of planning and scheduling. Unlike other reactive approaches, in which all decisions are subject to deferment, in completable scheduling only achievable decisions may be deferred. This has two main benefits. The first is that the cost of dynamic decision–making is minimized, since only some goals must be planned for and scheduled during execution. The second is that the robustness and flexibility afforded by reactivity is gained without losing the goal–directedness and guarantees of success afforded by a priori planning. Additionally, the use of contingent EBL enables a completable scheduling system to improve its performance through experience. By learning general completable schedules from example, the system can amortize the cost of constructing a completable schedule over the number of times the learned general schedule is applied in future instances as well as reduce the planning cost incurred by the system's a priori planning module.

Acknowledgments. This research was supported by the Office of Naval Research under grants N–00014–86–K–0309 and N–00014–91–J–1563. We would also like to thank Scott Bennett, Steve Chien, Jon Gratch, and Dan Oblinger for many interesting discussions, as well as Michael Shaw, for introducing us to the domain of process planning and scheduling.

References

[Agre87] P. Agre and D. Chapman, "Pengi: An Implementation of a Theory of Activity," *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987, pp. 268–272.

[Bennett90] S. Bennett, "Reducing Real–world Failures of Approximate Explanation–based Rules," *Proceedings of the Seventh International Conference on Machine Learning*, Austin, TX, 1990, pp. 226–234.

[Chien89] S. A. Chien, "Using and Refining Simplifications: Explanation–based Learning of Plans in Intractable Domains," *Proceedings of The Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 590–595.

[DeJong86] G. F. DeJong and R. J. Mooney, "Explanation–Based Learning: An Alternative View," *Machine Learning 1*, 2 (April 1986), pp. 145–176. (Also appears as Technical Report UILU–ENG–86–2208, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana–Champaign.)

[Drummond90] M. Drummond and J. Bresina, "Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction," *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, August 1990, pp. 138–144.

[Fikes72] R. E. Fikes, P. E. Hart and N. J. Nilsson, "Learning and Executing Generalized Robot Plans," *Artificial Intelligence 3*, 4 (1972), pp. 251–288.

[Firby87] R. J. Firby, "An Investigation into Reactive Planning in Complex Domains," *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987, pp. 202–206.

[Fox84] M. S. Fox and S. F. Smith, "ISIS—a knowledge–based system for factory scheduling," *Expert Systems 1*, 1 (July 1984),.

[Gervasio90a] M. T. Gervasio, "Learning General Completable Reactive Plans," *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, August 1990, pp. 1016–1021.

[Gervasio90b] M. T. Gervasio, "Learning Completable Reactive Plans Through Achievability Proofs," Technical Report UIUCDCS–R–90–1605, Department of Computer Science, University of Illinois, Urbana, IL, May 1990.

[Gervasio91] M. T. Gervasio and G. F. DeJong, "Learning Probably Completable Plans," Technical Report UIUCDCS–R–91–1686, Department of Computer Science, University of Illinois, Urbana, IL, April 1991.

[Hammond86] K. Hammond, "CHEF: A Model of Case–Based Planning," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 267–271.

[Kaelbling88] L. P. Kaelbling, "Goals as Parallel Program Specifications," *Proceedings of The Seventh National Conference on Artificial Intelligence*, Saint Paul, MN, August 1988, pp. 60–65.

[Knoblock90] C. Knoblock, "Learning Abstraction Hierarchies for Problem Solving," *Proceedings of the Eight National Conference on Artificial Intelligence*, Boston, MA, 1990, pp. 923–928.

[Martin90] N. G. Martin and J. F. Allen, "Combining Reactive and Strategic Planning through Decomposition Abstraction," *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, San Diego, CA, November 1990, pp. 137–143.

[Minton85] S. Minton, "Selectively Generalizing Plans for Problem–Solving," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, August 1985, pp. 596–599.

[Mitchell86] T. M. Mitchell, R. Keller and S. Kedar–Cabelli, "Explanation–Based Generalization: A Unifying View," *Machine Learning 1*, 1 (January 1986), pp. 47–80.

[Muscettola90] N. Muscettola and S. F. Smith, "Integrating Planning and Scheduling To Solve Space Mission Scheduling Problems," *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, San Diego, CA, November 1990, pp. 220–230.

[Ow88] P. S. Ow, S. Smith and A. Thiriez, "Reactive Plan Revision," *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, MN, August 1988, pp. 77–82.

[Prosser89] P. Prosser, "A Reactive Scheduling Agent," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 1004–1009.

[Sacerdoti74] E. Sacerdoti, "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence 5*, (1974), pp. 115–135.

[Smith90] S. F. Smith, P. S. Ow, N. Muscettola, J. Potvin and D. C. Matthys, "OPIS: An Integrated Framework for Generating and Revising Factory Schedules," *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, San Diego, CA, November 1990, pp. 497–507.

[Stefik81] M. Stefik, "Planning and Metaplanning (MOLGEN: Part 2)," *Artificial Intelligence 16*, 2 (1981), pp. 141–170.

[Zweben90] M. Zweben, M. Deal and R. Gargan, "Anytime Rescheduling," *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, San Diego, CA, November 1990, pp. 251–259.