Robust Planning with Imperfect Models

Russell Knight, Gregg Rabideau, Barbara Engelhardt, Steve Chien,
Jet Propulsion Lab, California Institute of Technology
4800 Oak Grove Drive, Pasadena, CA 91109-8099
{knight, rabideau, engelhar, chien}@aig.jpl.nasa.gov

Abstract

In general, most approaches to robust autonomy with respect to planning and execution are focused on either providing models that allow for flexibility or providing techniques for changing models to improve performance. We take these techniques into consideration, but focus the majority of our work on robust autonomous planning and execution with imperfect models. We deal with this through a number of means: 1) providing fast replanning in the event of unforeseen events, 2) modeling execution context, and 3) adjusting the planning and scheduling system heuristics given a model of the uncertainty of the environment. We have demonstrated these techniques using the Continuous Activity Scheduling, Planning, Execution, and Replanning (CASPER) system.

Introduction

Robust autonomy within the context of planning and execution is central to many NASA and JPL missions. Not only do communications constraints demand more autonomy but also budgetary constraints that demand more automation. In this context, systems must be able to autonomously plan, execute, and respond to the environment. Often, models are used for planning, and feedback systems provide the execution, using a number of layers to provide the required levels of abstraction and detail with respect to task dispatching and execution. A designer can provide robustness in one of three ways:

1) provide a flexible model which entails all possibilities of interest,
2) provide a system that modifies the model in the face of unexpected events, and
3) provide a system that attempts to be as robust as possible with a given model.

Model flexibility is probably the most common technique for providing robustness. Examples are the use of temporal constraint networks [9], resource profiling, and contingency planning [10]. The primary advantage of such systems is a-priori knowledge of its performance. But, some systems produce plans that in the end are overly pessimistic. For example, a task might take on average 10 minutes to complete, but can vary from 5 to 15 minutes according to a distribution. So, if we planned a large number of such tasks, human experts might use the average case time to project feasibility. Unfortunately, systems that provide model flexibility often would be forced to assume that each task would require 15 minutes. Even those that provide a margin of error (say, 1 standard deviation is acceptable) would choose a fixed approximated time for the events regardless of the number of events.

Changing models or developing new models automatically is quite an open area of research, and several systems provide some utility in this area. Hidden Markov Models, Neural Nets, and other adaptive systems fall into this category [11]. In practice, it has proven to be very difficult to learn a model that a symbolic system can then use for planning and scheduling, although this remains an active area of research. Still, no spacecraft has of yet flown with large-scale onboard adaptation of models and autonomy (although see [17, 18] for a description of a flight experiment of significant model-based autonomy).

Robust planning and execution compliments both flexible modeling systems and model modification systems in that the model is a given and the system provides robustness through its execution and planning. Therefore, we focus our efforts on providing robustness with a given model. We accomplish this through a number of means:

1) providing fast replanning in the event of unforeseen events,
2) modeling execution context, and
3) adjusting the planning and scheduling system heuristics given a model of the uncertainty of the environment.

Fast Re-planning

To facilitate fast replanning, we utilize an *iterative repair* algorithm [1,2]. We accept that in some circumstances, iterative repair does not perform as well as constructive algorithms, however our experience has been that in practice it has performed well for a number of real domains [8]. Iterative repair is a natural choice for a system that receives state updates because if an update causes a conflict iterative repair naturally accepts a current plan with a flaw and repairs it. The only algorithmic adaptation that is needed it to ensure that certain operations are not allowed in an execution context (e.g., changing activities in the past.)

During iterative repair, the conflicts in the schedule are analyzed and addressed one at a time until no conflicts exist, or a computation resource bound has been exceeded. A conflict is a violation of a constraint. We consider the following types of constraints: resources, states, temporal relationships, parameter dependencies, and decompositions. Conflicts can be repaired by means of several predefined methods. We consider the following repair methods: moving an activity, adding a new instance of an activity, deleting an activity, detailing an activity, abstracting an activity, making a reservation of an activity, canceling a reservation, connecting a temporal constraint, disconnecting a constraint, and changing a parameter value. The repair algorithm first selects a conflict to repair then selects a repair method. The type of conflict being resolved determines which methods can repair the conflict. Depending on the selected method, the algorithm may need to make addition decisions. For example, when moving an activity, the algorithm must select a new start time for the activity.

To achieve a higher level of responsiveness in a *dynamic planning* situation, we utilize a *continuous planning* approach and have implemented a system called CASPER (for Continuous Activity Scheduling Planning Execution and Replanning) [3, 8]. Rather than considering planning a batch process in which a planner is presented with goals and an initial state, the planner has a current goal set, a plan, a current state, and a model of the expected future state. At any time an incremental update to the goals, current state, or planning horizon (at much smaller time increments than batch planning) may update the current state of the plan and thereby invoke the planner process. This update may be an unexpected event or simply time progressing forward. The planner is then responsible for maintaining a consistent, satisficing plan with the most current information. This current plan and projection is the planner's estimation as to what it expects to happen in the world if things go as expected. However, since things rarely go exactly as expected, the planner stands ready to continually modify the plan. From the point of view of the planner, in each cycle the following occurs:

- changes to the goals and the initial state first posted to the plan,
- effects of these changes are propagated through the current plan projections (includes conflict identification)
- iterative repair is invoked to remove conflicts and make the plan appropriate for the current state and goals.

This approach is shown in below in Figure 1. At each step, the plan is created by using iterative repair with:

- the portion of the old plan for the current planning horizon;
- the updated goals and state; and
- the new (extended) planning horizon.

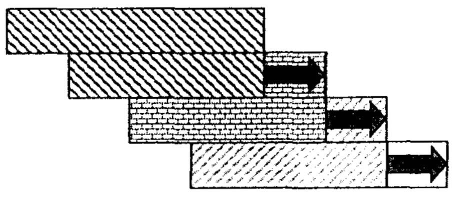
Fig. 1 Continuous Planning Incremental Extension

Modeling Execution Context

Modeling execution context provides a way to give the planner information about the execution. Ideally, this would not disrupt the design of the planner to the point that we need to research an entirely different problem. We achieve this through these means:

1. Abstraction Hierarchies
2. Commitment Strategies

Abstraction hierarchies provide information about the criticality of detail required given the current time with respect to an executing plan (see Figure 2). For example, consider the minutiae involved with executing the plan "get in the car, drive to McDonalds, buy some fries." When actually getting in the car, we take input from our environment before we actually decide on what way we will actually expand this activity. We might enter on the passenger side if the driver's side is blocked by our teenage daughter's car. But, when making the plan, we do allot a certain amount of time and other scarce resources. Basically, as activities become more urgent, our view of them changes, and we need to plan more details. If we plan all details in advance, we may waste considerable computational resources on decisions that become invalidated during execution.

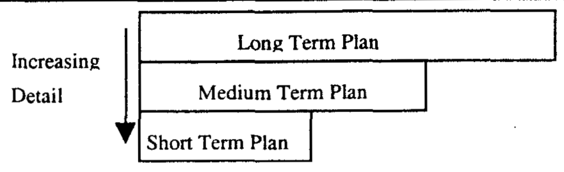

Fig. 2 Hierarchical Planning Horizons

Of course, this requires a certain amount of modeling, and one could argue that we are simply including execution context in the modeling problem and are therefore more similar to flexible modeling systems than we claim. This is true, if you consider modeling execution as an end. But, we model abstraction hierarchies to improve the responsiveness of the planner by simplifying the problems that we ask it to solve, not improve the actual veracity of a model.

Commitment strategies are another way to make the execution of a plan more robust. In many planning systems, some execution robustness is achieved using least-commitment techniques throughout planning to retain plan flexibility. In practice, however, many problems are over-constrained and the planner must interface to other software components that cannot handle flexible values. This can significantly reduce the flexibility of the resulting plans. Instead, we commit to decisions during planning (for the given abstraction hierarchies) but only commit to an upcoming piece of the plan for execution. Basically, commitment strategies are decisions about when to freeze decisions about an activity. For example, we could never change any information about an activity in the past (except by receiving information from the environment), so past activities must be frozen or *committed*. By committed, we mean that the planner is not allowed to change any information about the activity, and it is ready to dispatch to the execution system. But, how early must we commit activities? Should we wait until the last second, choose a different strategy for each type of activity, or choose a different strategy for each activity instance?

We choose to model the commitment strategy for each activity type. This affords us a significant amount of flexibility in designing an appropriate commitment strategy for different execution contexts.

Adjusting the Planner

Another way we provide robustness with imperfect models is by automatically adjusting the planner's search heuristics to the domain in question. This is facilitated using a technique called Adaptive Problem Solving (APS) [6]. The basic idea is that by inferring a probabilistic model of the domain, we can automatically adjust the planner to provide robust replanning for the simulation.

The advantages of this are clear—given a model, we can adjust a planner's parameters to perform well on a problem regardless of flaws in the planner, the planner's heuristics, or the planner's parameters. This means that certain types of programming errors, logical flaws, and lack of domain expert advice can be automatically corrected.

Adaptive problem solving uses stochastic optimization [16] to find the best parameters for a planner applied to a specific domain. The algorithm takes the set of parameters to optimize, called *strategies*, and selects the strategy that has the highest *expected utility* using a decision criterion. Using the selected strategy, the algorithm makes local steps using a chosen neighborhood algorithm (e.g., local mutations or genetics-inspired search) to generate the subsequent set of strategies. The algorithm stops when it reaches a maximum time or number of iterations, or quiescence.

Expected utility is the criteria by which the strategies are ranked. Expected utility is the average utility of the final plan, repaired using the chosen strategy, over a number of stochastic simulations of the domain with stochastic starting points (or alternatively execution traces in the environment). Higher expected utilities, where utility is defined by the domain preferences, can be built so as to be a criterion for robustness.

The decision criterion used in adaptive problem solving enables strategy selection at a low sampling cost. Statistical decision criteria, such as the Nadas criterion [15], Bernstein's inequality[14], Hoeffding's inequality[13], and Chernoff Bounds [12], generate estimates of their expected utility based on sampling the strategies in the stochastic domain. The criteria are used to determine when the number of samples is enough to guarantee the selection accuracy within a certain error. Error allocation techniques are used to enable selection with a minimal number of samples.

APS can be applied to any parameter of the planner, including (but not limited to) the commitment strategy, the set of heuristics used by the planner, weightings of these heuristics, or the amount of time dedicated to replanning.

It should be noted that this could also be applied on-line, but in practice the number of epochs to produce a significantly better planner may be very high. Even in ground-based simulations, APS would enable a planner to improve robustness by learning its parameters based on feedback from the environment.

Discussion and Future Work

Robustness can be thought of as an aspect of plan quality. We may have a preference for more robust plans, but this must be weighed other preferences. Often there is a preference for packing the plan with as many goals as possible, which can often decrease robustness. We plan to provide a representation for robustness preferences and use optimization techniques from [4] to help maintain robust plans.

However, preferences of this type can be particularly difficult to express. In addition, they often directly compete with many of the science objectives. For example, one dominant objective is to simply achieve as many goals as possible. This requires packing activities into the time frame of the plan. Tight temporal and resource packing can leave very little room for execution error, resulting in a

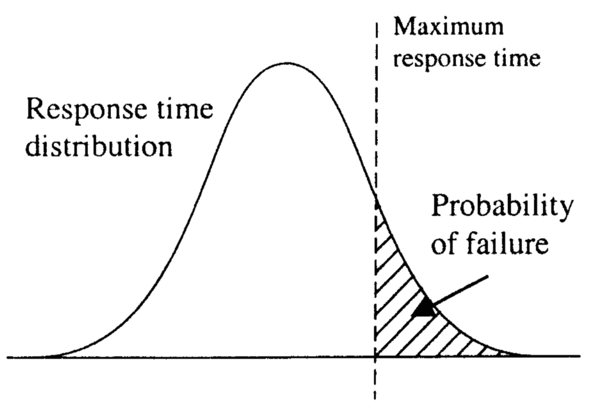

Fig. 3 Robustness as a function of response time characteristics

brittle plan. We would like the user to be able to express preferences for keeping the plan robust and making an informed trade-off with other preferences.

Plan robustness can vary over the temporal boundaries of the plan. Robustness at time *t* is a function of the maximum allowable response time at time *t* and the distribution of potential responses that may be required at time *t*. In other words, a plan is robust if responses are fast relative to the time limits for making the responses. We can determine approximate actual and maximum response times using static analysis and simulations. The necessary response at a given time depends on the constraints and dependencies associated with the activities in the plan at that time. Static analysis will reveal the constraint networks and response time will be approximately correlated to the complexity of these networks. Running simulations of plan executions in dynamic environments will also give us estimates of response time distributions.

Once we have response time (i.e., re-planning time) information, we can evaluate and improve a robustness preference. Robustness can be viewed as a function of the probability of failure, where failure occurs when response time is greater than the maximum allowable. This is the area under the response time distribution curve and greater than the maximum response time (see Figure 3). Robustness will increase if we: 1) decrease the response time mean or standard deviation, or 2) increase the maximum allowable response time. Thus, either can be done to increase quality. Understanding and decreasing the response time mean and standard deviation is a difficult problem. This can be thought of as analyzing the edit distance between a plan and its neighboring valid plans [5]. Robust plans will have smaller edit distances, and the challenge is keeping the average distance small. The alternative, increasing maximum response time, is only a slightly less difficult problem. Here, we can analyze and relax the dependencies between activities in the plan.

Conclusion

We have classified the three approaches to robustness with respect to models in a planning and execution context. We argue that robust execution and planning can occur in the face of imperfect models, and have described three techniques for achieving this robustness. They are: 1) providing fast replanning in the event of unforeseen events, 2) modeling execution context, and 3) adjusting the planning and scheduling system heuristics given a model of the uncertainty of the environment.

Acknowledgements

This work was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

[1] M. Zweben, B. Daun, E. Davis, and M. Deale, "Scheduling and Rescheduling with Iterative Repair," in *Intelligent Scheduling*, Morgan Kaufman, San Francisco, 1994.

[2] G. Rabideau, R. Knight, S. Chien, A. Fukunaga, A. Govindjee, "Iterative Repair Planning for Spacecraft Operations in the ASPEN System," *International Symposium on Artificial Intelligence Robotics and Automation in Space*, Noordwijk, The Netherlands, June 1999.

[3] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau, "Using Iterative Repair to Improve Responsiveness of Planning and Scheduling," In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, Breckenridge, CO, April 2000.

[4] G. Rabideau, B. Engelhardt, S. Chien, Using Generic Preferences to Incrementally Improve Plan Quality, In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, 2000, 236-245.

[5] M. Ginsberg, A. Parkes, A. Roy, Supermodels and Robustness, In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 1998, 334-339.

[6] B. Engelhardt, S. Chien. Empirical Evaluation of Local Search Methods for Adapting Planning Policies in a Stochastic Environment. *Local Search in Planning and Scheduling*, Springer-Verlag, 2001.

[7] S. Chien, R. Knight, G. Rabideau, "An Empirical Evaluation of the effectiveness of local search forreplanning," Local Search in Planning and Scheduling, Springer-Verlag, 2001.

[8] S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins, D. Tran , "ASPEN - Automating Space Mission Operations using Automated Planning and Scheduling," SpaceOps 2000, Toulouse, France.

[9] A. Jonsson, P. Morris, N. Muscettola, K. Rajan, and B. Smith, "Planning in Interplanetary Space: Theory and Practice," *in Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, Breckenridge, CO. April, 2000.

[10] J. Bresina, K. Golden, D. Smith, and R. Washington , "Increased Flexibility and Robustness of Mars Rovers" *International Symposium on Artificial Intelligence Robotics and Automation in Space*, Noordwijk, The Netherlands, June 1999.

[11] Advances in Neural Information Processing Systems MIT Press.

[12] T. Hagerup and C. Rub, (1990). A Guided Tour of Chernov Bounds, Information Processing Letters 33: 305-308.

[13] W. Hoeffding, (1963). Probability inequalities for sums of bounded random variables, Journal of the American Statistical Association 58(301):13-30

[14] S. N. Bernstein, (1946). *The Theory of Probabilities*, Gastehizdat Publishing House.

[15] A. Nádas (1969), "An extension of a theorem of Chow and Robbins on sequential confidence intervals for the mean," The Annals of Mathematical Statistics 40:2, pp. 667-671.

[16] Boyan, J. A. and W. L. Buntine, eds. "Statistical Machine Learning for Large-Scale Optimization." *Neural Computing Surveys* 3, 2000.

[17] D. Bernard , G. Dorais , E. Gamble, B. Kanefsky, J. Kurien, G. Man, W. Millar, N. Muscettola, P. Nayak, K. Rajan, N. Rouquette, B. Smith, W. Taylor, Yu-Wen Tung, "Spacecraft Autonomy Flight Experience: The DS1 Remote Agent Experiment," *Proceedings of the American Institute of Aeronautics and Astronautics Conference*, 1999 Albuquerque NM.

[18] N. Muscettola, P. Nayak, B. Pell, and B. Williams, "Remote Agent: To Boldly Go Where No AI System Has Gone Before," *Artificial Intelligence* 103(1-2): 5-48, August 1998.