

# Finding plans and defining plan constraints simultaneously

Jim Blythe  
Information Sciences Institute  
University of Southern California  
Marina del Rey, CA 90292, USA  
blythe@isi.edu

## Abstract

When people create plans, the constraints and criteria on the plans often become explicit only while potential plans are being examined. Therefore a tool that acquires plan constraints might benefit from including a tool to allow users to navigate the space of candidate plans. Preliminary work indicates that both this and the converse are true: being able to modify plan constraints will also help users to find good plans. I integrate a plan viewing tool based on Design Galleries with Constable, a system for acquiring procedural plan constraints. Constable includes several different knowledge acquisition tools as components, and the way that these tools share information with each other about the overall acquisition task provides considerable help to the user. At the same time, the plan viewing tool is more useful when users can modify constraints as they explore the plans.

## Introduction

Most plans created in the real world involve a compromise between the planner's desired goals and what can feasibly be achieved. Tools that help users to create plans should therefore support the dual process of describing constraints and preferences over the set of possible plans and finding plans that best match these constraints and preferences. However, most planning tools that currently exist fall into one of two categories. First, automatic planning systems [Weld, 1994; Veloso *et al.*, 1995] take as input a logical specification of a goal state and attempt to find a plan which will achieve the goal specification. These systems do not allow the user to modify the goals while planning is in progress, and in general permit very little user interaction during planning. The second category of planning tools comprises plan editors that allow a user to describe a plan, usually based on a fixed grammar [Frank & Szekely, 1998]. Using a plan editor, the user selects the goals, and chooses how to decompose them into tasks and subgoals. Although the user has complete control over the process, the tool provides little or no support in searching for plans or checking whether plans satisfy their constraints. An exception is the work on TRAINS and TRIPS by James Allen and his colleagues [Ferguson, Allen, & Miller, 1996], but although these tools allow users to modify their goals while planning, they do not provide help in

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

K-CAP'01, October 22-23, 2001, Victoria, British Columbia, Canada.

Copyright 2001 ACM 1-58113-380-4/01/0010...\$5.00

maintaining the constraints on plans or in viewing several alternative plans.

In this paper I present a system that helps users to search for good plans and describe or modify plan constraints simultaneously. This is a natural way to approach human planning tasks, and improves performance at both pieces of the task. The system is an extension to the already developed Constable tool for acquiring plan constraints [Blythe, 2001; Blythe *et al.*, 2001] with a novel interactive tool for viewing and exploring the space of possible plans based on the Design Galleries approach [Marks *et al.*, 1997]. In this approach, which I describe in more detail below, a subset of the possible plans is chosen and displayed graphically to the user in summary form, allowing the user to navigate easily between them. The subset is chosen so that the user can explore a broad coverage of the space of solutions, and arranged so that "similar" solutions are near each other on the screen.

To see why it is necessary to combine searching for good plans with describing what makes a good plan, consider the following holiday-planning problem. I wish to take a sailing holiday around Catalina Island, which can be reached by chartered boat from either Los Angeles, Orange County or San Diego airports, although the sailing time is different from each. I am free to leave any time in the next week, although this depends on the flights available, and I would prefer to spend at least five days on the island. On the day I arrive, I would like to sail and dock in time to view the sunset from the island. Depending on the kind of boat I charter, I may need to arrive within half an hour of a high tide.

A plan for this domain can be completely described by a flight, a boat charter and a time at which to start sailing. This is not a complex planning problem in terms of composing a sequence of actions to achieve my goal whose preconditions may interfere with one another. However there are at least two reasons that an interactive tool is needed for choosing the plan. First, the choices to be made have interactions, since the charter boats available and the time of high tide depend on the day of the flight, for example. Therefore the best plan cannot be found simply by choosing a preferred option at each choice point. Second, the user may not know explicitly how to compare each pair of plans ahead of time. Therefore a good plan cannot be found by an algorithm that doesn't take any input from the user.

At a minimum, an interactive tool to help in finding a good plan should help a user keep track of simple constraints that rule out infeasible plans, for example by ensuring that there is

enough time to get from the plane to the boat before it sails, and predicting the sailing time required. A tool could also assist the user by finding useful information such as available flights and the times of sunset and high tide from web sources. In order to do these things flexibly, our implementation allows the user to define these constraints explicitly and have them be applied to each candidate plan.

In addition to these features, the tool provides an overview of the space of plans by laying out a subset of the possible plans graphically, allowing the user to see whether the plans violate any constraints and which alternative plans are nearby. Because the constraints are represented in an executable form, the user can modify, remove or add constraints to explore how this changes the landscape of possible plans. The ability to change plan constraints while searching for plans has two advantages. First, since the best plan may not satisfy all the initial constraints, it is helpful for users to sometimes soften or change constraints to see which plans form partial solutions. Second, users are often better able to describe constraints on plans when several plans can be explored in a real scenario.

In the next section I review work on visualizing alternative plans, in particular Design Galleries, and describe an initial implementation of this approach. The approach is implemented on top of Constable, a constraint editing tool that has been used to acquire plan constraints. The following section describes how users can modify plan constraints in Constable and how this is integrated with exploring possible plans. The final section includes a discussion of the tool and conclusions.

### Viewing the space of possible plans

To help users to choose between alternative plans, Constable displays as many of the alternatives as possible in a summary form. There are many ways to do this, depending on how many plans are displayed, how each plan summary is displayed and how the user can navigate through the plans. The approach used in Constable is based on Design Galleries (DG) [Marks *et al.*, 1997], a general approach to parameter tuning developed in the context of parameter tuning in computer graphics.

A DG display of candidates is created from several general elements. An *input vector* uniquely describes the candidate, while an *output vector* describes the relevant properties of the candidate from the user's point of view (including, for example, whether the boat arrives before sunset and the cost of the flight). A *distance metric* provides a measure of the similarity between two candidates. A *sampling* method aims to find a set of candidates that provide a broad, uniform cover of the space of candidates<sup>1</sup>. This method will be used to find a subset of all the candidates that will be displayed to the user. The candidates will be positioned in a screen using an *arrangement* method that uses the distance metric.

In the sailing trip domain, each plan can be described with an input vector consisting of the flight and the boat charter for the plan. The output vector includes annotations of prop-

erties of interest to the user, for example whether the boat arrives before sunset and the cost of the flight. In the simplest case, Hamming distance between output vectors can be used for the distance metric. Before discussing dispersion and arrangement methods for this example, we consider the size and complexity of the candidate plan space.

If a problem were small enough that all the candidates can easily be viewed together, there would be no need to consider sampling methods to reduce their number. Similarly, if there are only two choices to make in order to specify a plan, the arrangement method could be as simple as assigning an axis on the screen to each choice and laying out the candidates according to the choices they embody. However, sampling and arrangement techniques seem useful even in the sailing trip domain, which has two choices, and even when there are only a few hundred candidate plans, as we demonstrate here. Although the number of choices is small, the choice of which flight determines several pertinent features of a plan: which day the trip will start and which airport and airline are used. These are features which the user may want to consider independently and, therefore, to be able to view easily on the screen.

Figure 1 shows a constraint-based view of one plan. This is essentially the output vector for the plan. The choices of flight and charter boat appear with drop boxes, allowing the user to select different values for the choices individually. The view uses color to give the user feedback on whether constraints or preferences are being applied to the plan, and whether they are satisfied. For example, the background of the line labelled "Airport" is green in the original window, indicating that there is a constraint on this field that has been satisfied (the user wishes to fly to LAX).

The constraint view does not provide global information about the space of possible plans, however. Figure 2 shows part of a basic layout that includes every candidate plan. To generate candidate plans, three airlines are modelled as having two flights each arriving in the afternoon at each of three airports. Over six days this yields 108 flights, combined with two possible charter boats for 216 possible plans. Each flight is modelled as having independent probability 0.5 of being full, so the actual number of plans follows an approximately normal distribution with mean 108.

Although this is a relatively small set, the screen is too cluttered for the user to easily understand the possible plans. In order to reduce the number shown, unavailable flights have been left out. This leads to the plans that satisfy the current constraint, shown in green, having an apparently random distribution even though they are laid out systematically according to the flight and the boat.

Figure 3 shows a layout of the same set of plans, projected onto the dimensions of the date and the airport used. Each rectangle on the display now represents a set of candidate plans that share the same date and arrival time. A preferred member has been chosen for each set as the one with the earliest arrival time. Since some flights are not available, the number of candidate plans in each set, shown in the bottom of each rectangle, varies from day to day.

<sup>1</sup>This is called a dispersion method in Marks *et al.*

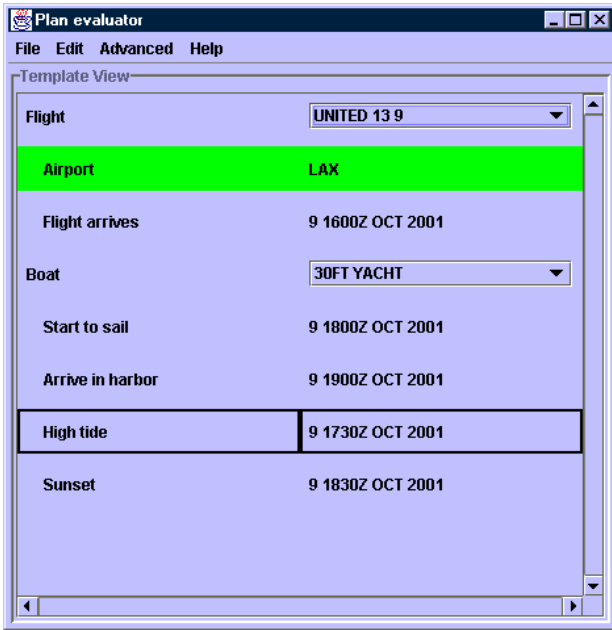


Figure 1: Constable's constraint view of a single plan shows the choices made in the plan as well as relevant derived information. Fields for which the user can make a choice have drop boxes to all a value to be chosen for the field independently of others.

This projection provides a way to implement the sampling method that makes up part of the general Design Gallery approach. The number of candidates shown is far smaller, 18 in this case, but the user can gain a greater understanding of the space since the candidates are arranged by pertinent features. Of course, this abstraction has hidden some information that might be important, depending on the user's goals. The airlines that can be used at a location and a time are hard to see, and one boat has consistently been chosen although two are available. This is a demonstration of a general point about design galleries, that to be effective they need to make use of any information that is available about which features are important to the user. In some cases this information is available in terms of a set of constraints or preferences expressed over the candidate set, even if the constraints may be incomplete or imprecise.

In the next section I describe how Constable allows users to specify constraints on plans, and how this can be used dynamically within the design gallery approach to improve the sampling and arrangement of candidates.

### Modifying constraints in Constable

Constable is designed to acquire constraints on plans that are represented as executable procedures. For example, in a travelling domain Constable can be used to express "prefer a direct flight unless it is double the price of the cheapest connecting flight", or "if the flight arrives late in the evening, the hotel should be near the airport, otherwise it should be near the meeting". Constraints expressed in Constable can be used to test candidate plans, usually resulting in a boolean satisfaction. These constraints are used in the plan displays shown in Figures 2 and 3, where a plan is outlined in red if it violates a

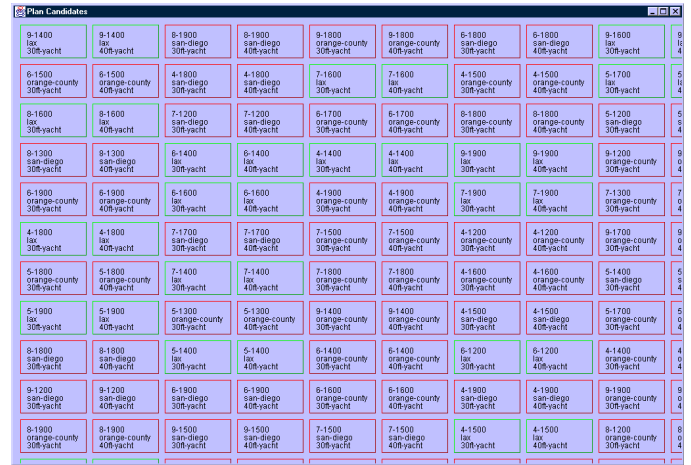


Figure 2: A display with every possible plan choice shown.

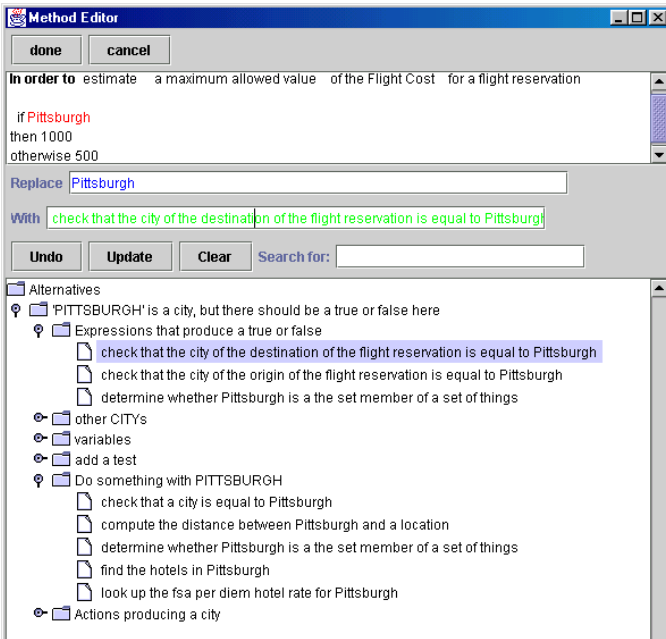


Figure 3: A display with plans projected onto date and airport. Each rectangle denotes a set of plans and the size of the set is shown as the number at the bottom of the rectangle.

constraint. In this section I discuss how the plan constraints are acquired and expressed, and the impact this has on exploring and choosing plans within this framework. More details about Constable are contained in [Blythe *et al.*, 2001; Blythe, 2001].

Constable is built on the Expect knowledge acquisition (KA) system [Gil & Swartout, 1995], and includes several inter-relating tools that help a user to acquire procedural knowledge. Although Constable focusses on plan constraints, most of these tools are general. First, an english-based editor allows the user to modify English paraphrases of the formal procedure representations used in Expect. Figure 4 shows Constable's editor with a procedure to compute the maximum allowed value of the cost of a flight reservation. The purpose and body of the procedure are automatically paraphrased in English [Blythe & Ramachandran, 1999]. When the user selects part of the procedure to change, in this case "Pittsburgh", the editor suggests replacements in the lower panel, also automatically paraphrased. The suggestions are generated by analyzing the current knowledge base for possible terms and grouping them. Examples of groups are procedures or relations that could be applied to the current term or that have the same type. This approach hides the formal syntax while avoiding the challenge of full natural language processing.

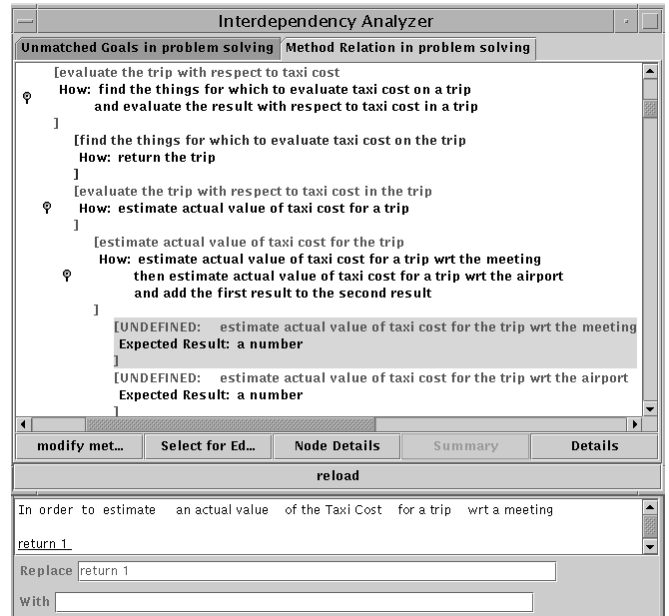
A constraint in Constable is typically defined in terms of a



**Figure 4:** Constable's editor being used to define how to compute the maximum allowed value of the cost of a flight reservation.

number of inter-relating procedures and supporting concepts, instances and relations. The interdependency manager helps to keep track of how the pieces work together by building an interdependency model [Kim & Gil, 1999]. The manager can then inform the user about KA subtasks that are still to be completed, and give an overview of the whole constraint definition as the subtasks are performed. The Interdependency Analyzer displays the problem solving steps as shown in Figure 5. At this point, the user has provided a few problem solving pieces, but the system highlights the substeps that it does not yet know how to solve. In this case, it indicates that it does not yet know how to estimate the taxi cost to and from the meeting. When the user clicks on that portion of the problem solving tree, the Interdependency Analyzer brings up the Method Editor, and creates an initial template for the method based on its place within the problem solving tree. In this case, the method needed is for estimating taxi cost trip from the hotel to the meeting, and it should return a number since the system knows that this result will be added to something else (in this case to the taxi cost from the airport to the hotel).

Constable also includes editors for concepts, instances and relations in the knowledge base. They can be invoked while using the other editors, such as the procedure editor, and make use of available contextual information about the KA task to automate as much as possible of the editing [Blythe *et al.*, 2001]. Overall, these editors and KA support tools allow end users to add or modify plan constraints that make use of procedural information even if they have little or no training in programming. I now discuss how modifying the constraints can be combined with the Design Gallery view to help in searching for a good plan.



**Figure 5:** The Interdependency Analyzer helps the user to add problem solving knowledge by showing how the individual steps and substeps relate to each other.

### Using projections for intermediate control of design galleries

The Design Gallery approach is very general and was used in a number of different interfaces by its original designers, by varying the dispersion and arrangement algorithms [Marks *et al.*, 1997]. In this work we aim to provide a fixed implementation of a Design Gallery, that is still very flexible for showing candidate plans in widely different domains and with different constraints. The initial solution currently being investigated relies on the idea of *projections* that was introduced in the previous section. A projection is a set of plan properties that induce an equivalence relation over the candidate plans, where two plans are equivalent if they have the same value for all of the properties in the projection.

The Design Gallery implementation used with projections, which is fixed, uses a simple dispersion algorithm that takes as an input a number of candidates to display. If the number is greater than or equal to the number of equivalence classes in the projection, it displays them all. Otherwise the classes are sampled randomly, converging towards uniform coverage as the number of classes shown increases. Two different arrangement algorithms are used. If the size of the projection is two, the examples are arranged on a grid as shown in Figures 2 and 3. If there are more than two properties in the projection, a simplified version of multi-dimensional scaling is used to approximate the distance between two classes with distance on the two-dimensional screen [Borg & Groenen, 1997; Marks *et al.*, 1997].

The examples shown in the previous section can now be re-described in terms of projections. Initially, the user selects a projection containing the flight and airport (this selection can be made interactively from the plan view display). Since there are only two elements, this leads to the grid view of

candidates shown in Figure 2. However, this view is too cluttered, so the user creates a new property in Constable, defined as the day on which the flight arrives. This property can be created by copying the “flight arrives” field shown in Figure 1 and using the procedure editor to select the day of the flight arrival. Replacing the flight field in the projection with the new day field leads to the candidate display in Figure 3.

I hypothesize that the projection-based implementation of design galleries will be sufficient to allow most users to explore plans based on their current and evolving understanding of the constraints. The approach allows candidates to be grouped and arranged in many different ways while maintaining simplicity of control, and keeps attention focussed on the features that make a plan good, which typically occupy the user’s attention already. Although an initial implementation of this approach has been made, work is in preliminary stages and testing of this hypothesis will be made later. It is worth noting that the approach based on plan constraints suggests a natural way to display the candidate plan icons, another component of Design Galleries although not mentioned in the original paper. Candidates are shown with the values of the properties that define each equivalence class, and also with colouring that shows whether the constraints are satisfied. If at least one element in the class satisfies the constraints, the class is shown in green, otherwise it is shown in red. There are many variations that can be made to this basic approach to the sampling, arrangement and display of candidates based on projections, and I touch on some of them in the next section.

### Discussion

This paper describes the use of Design Galleries to search for good plans at the same time as acquiring knowledge about the constraints themselves. Both activities can be viewed as knowledge capture, although acquiring the constraints includes considerably more involved KA tasks. There are interesting issues to consider from the point of view of interactive systems for knowledge capture arising from both the use of Design Galleries and its integration with a KA system like Constable.

In AI planning research, most recent work has concentrated on fast algorithms for finding plans that solve a logically specified goal, but several researchers have considered criteria of plan quality. Perez and Carbonell present a system to automatically learn rules that will direct a planning system to produce plans of higher quality in [Pérez & Carbonell, 1994]. They also define a framework for plan quality criteria. Estlin and Mooney use a different learning approach in [Estlin & Mooney, 1996]. Neither of these tools are interactive, however. Tate et al present a way for users to interact with an AI planning system and make judgments about plan quality in [Tate, Dalton, & Levine, 1998], and the already mentioned TRIPS and TRAINS systems have pioneered the field of mixed-initiative planning [Ferguson, Allen, & Miller, 1996]. These interactive tools do not use a formal procedural constraint definition that non-programmers can manipulate while searching for plans, and also do not provide ways to view the entire space of candidate plans.

In the original Design Gallery approach, the candidate so-

lutions are computed off-line, typically overnight, to ensure a good response time for the user during selection. This presents an interesting tradeoff in terms of interactive tools for knowledge capture. On the one hand, response time is important for users to interact with a system. On the other hand, by pre-computing one view of the candidate space, the tool sacrifices the chance to give interactive support over all views of the space that the user might wish to see. In the current implementation in Constable, views are created on-line, which is feasible because the number of constraints and overall number of candidates is typically small, and each constraint is usually fast to run. However, Constable constraints can sometimes make use of information extracted from web-based sources, through Heracles [?], and this introduces a significant time lag. When this happens we pre-cache the web-based sources for all candidates to ensure good response time. The flexibility of the tool, however, means that there can be no guarantee of response time, for example if the user modifies a constraint to use web sources that were not pre-cached.

Viewed as a tool for acquiring plan constraints, an important feature in Constable is the way that all the component tools share contextual information about the KA task that is of benefit to all the tools. We view the user’s interaction with the display of candidate plans as an interesting new source of contextual information that may improve the process of acquiring constraints. For example, if the user selects a candidate and then begins to modify a constraint, it might be the case that the candidate is one that satisfies this constraint, and this information in turn might provide powerful hints about the way the user wants to modify the constraint. We plan to explore these issues in future work.

### REFERENCES

- Blythe & Ramachandran, 1999. Blythe, J., and Ramachandran, S. 1999. Knowledge acquisition using an english-based method editor. In *Proc. Twelfth Knowledge Acquisition for Knowledge-Based Systems Workshop*.
- Blythe *et al.*, 2001. Blythe, J.; Kim, J.; Ramachandran, S.; and Gil, Y. 2001. An integrated environment for knowledge acquisition. In *Proc. Conference on Intelligent User Interfaces*.
- Blythe, 2001. Blythe, J. 2001. Integrating expectations from different sources to help end users acquire procedural knowledge. In *Proc. 17th International Joint Conference on Artificial Intelligence*. Seattle, WA.: Morgan Kaufmann.
- Borg & Groenen, 1997. Borg, I., and Groenen, P. 1997. *Modern Multidimensional Scaling: Theory and Applications*. Springer.
- Estlin & Mooney, 1996. Estlin, T., and Mooney, R. 1996. Multi-strategy learning of search control for partial-order planning. In *Proc. Thirteenth National Conference on Artificial Intelligence*. AAAI Press.
- Ferguson, Allen, & Miller, 1996. Ferguson, G.; Allen, J.; and Miller, B. 1996. Trains-95: Towards a mixed-initiative planning assistant. In Drabble, B., ed., *Proc. Third International Conference on Artificial Intelligence Planning Systems*. University of Edinburgh: AAAI Press.

- Frank & Szekely, 1998. Frank, M., and Szekely, P. 1998. Adaptive forms: an interaction technique for entering structured data. *Knowledge-Based Systems Journal* 11(1):37–45.
- Gil & Swartout, 1995. Gil, Y., and Swartout, B. 1995. Expect: Explicit representations for flexible acquisition. In *Proc. Ninth Knowledge Acquisition for Knowledge-Based Systems Workshop*.
- Kim & Gil, 1999. Kim, J., and Gil, Y. 1999. Deriving expectations to guide knowledge-base creation. In *Proc. Sixteenth National Conference on Artificial Intelligence*, 235–241. AAAI Press.
- Marks *et al.*, 1997. Marks, J.; Andalman, B.; Beardsley, P.; Freeman, W.; Gibson, S.; Hodgins, J.; and T.Kang. 1997. Design galleries: A general approach to setting parameters for computer graphics and animation. In *Proc. of SIGGRAPH*, 389–400.
- Pérez & Carbonell, 1994. Pérez, M. A., and Carbonell, J. 1994. Control knowledge to improve plan quality. In Hammond, K., ed., *Proc. Second International Conference on Artificial Intelligence Planning Systems*, 323–328. University of Chicago, Illinois: AAAI Press.
- Tate, Dalton, & Levine, 1998. Tate, A.; Dalton, J.; and Levine, J. 1998. Generation of multiple qualitatively different plan options. In Reid Simmons, M. V., and Smith, S., eds., *Proc. Fourth International Conference on Artificial Intelligence Planning Systems*. Carnegie Mellon University, Pittsburgh, PA: AAAI Press.
- Veloso *et al.*, 1995. Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The prodigy architecture. *Journal of Experimental and Theoretical AI* 7:81–120.
- Veloso, Mulvehill, & Cox, 1997. Veloso, M.; Mulvehill, A.; and Cox, M. 1997. Rationale-supported mixed-initiative case-based planning. In *Proc. Conference on Innovative Applications of Artificial Intelligence*, 668–673. AAAI Press.
- Weld, 1994. Weld, D. 1994. A gentle introduction to least-commitment planning. *AI Magazine*.