Multistrategy Adaptive Path Planning

Ashok K. Goel, Khaled S. Ali, Michael W. Donnellan, Andres Gomez de Silva Garza, and Todd J. Callantine
Georgia Institute of Technology

WE USED A SPATIAL MODEL IN ADAPTIVE PATH PLANNING TO INTEGRATE MODEL- AND CASE-BASED METHODS. THIS RAISED QUESTIONS ABOUT STRATEGIC METACONTROL THAT WE SOUGHT TO ANSWER.

IN TRYING TO DESIGN ROBOTS THAT can navigate through space, path planning is a critical task. Path planning takes two locations in a physical world as input and gives a path connecting them as output. The traditional AI approach is to use a goal-directed heuristic search of problem spaces defined by spatial models of the navigation world. We call this family of search methods *model-based search.*

Case-based plan reuse offers an alternative approach, where planners solve new problems by retrieving and adapting previously formed plans for achieving similar goals.[1] The use of such methods for path planning raises complex questions: How do we index cases? How do we organize case memory? How do we retrieve relevant cases from the case memory? How do we store a new plan in the case memory for future use?

In 1990, we started the Router project to address these questions. Router is a multistrategy-strategy adaptive navigation path planner. It assumes that a mission planner such as Plexus[2] has generated a specific mission plan and identified specific path-planning tasks. Given a specific path-planning task, it uses a combination of model-based and case-based methods to solve it.

We began the project by developing a case-based adaptive path planner for navigation in simulated worlds. A spatial model of the navigation world provided the indexing scheme for organizing the case memory. This use of the spatial model in adaptive path planning suggested an integration of the model- and case-based methods, which raised questions about strategic metacontrol: How might the combined planner represent the two methods? How might it flexibly and dynamically select a specific method for solving a given path-planning task? How might it use one method for solving a subtask set up by the other?

New versions of the Router system view strategic metacontrol as a kind of design task that takes as input a specification of a problem-solving task and gives as output the specification of a virtual architecture for addressing it. One version of the system operates in simulated navigation worlds and provides a simple natural-language interface. Another version is embodied in Stimpy, an autonomous mobile robot.

Stimpy addresses issues in spatial navigation beyond path planning, such as plan execution and monitoring. Our goal here is to describe our general framework of multistrategy adaptive path planning, and the specific design of the Router system. To focus this discussion, we will not describe Router's embodiment in Stimpy, but instead report on a series of experiments with Router in simulated navigation worlds.

Spatial model

The spatial model of the navigation world plays multiple roles in Router. In addition to defining problem spaces for the model-based method, it provides the indexing

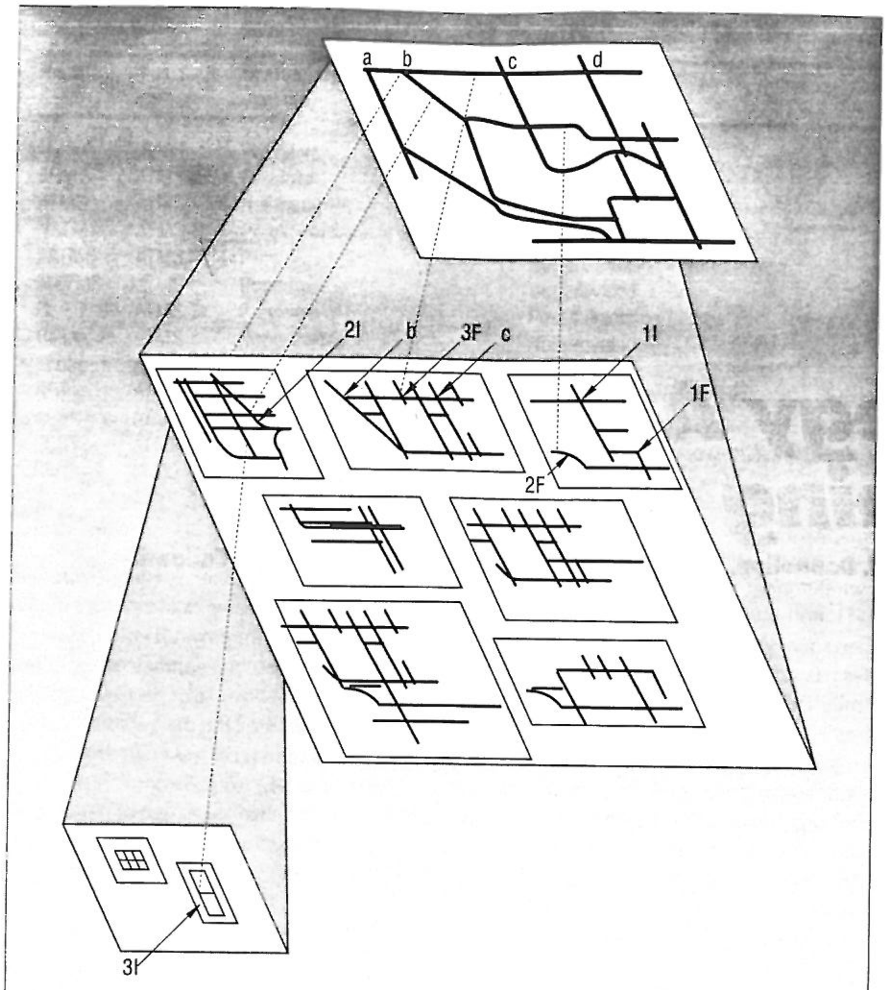

Figure 1. Hierarchical spatial model.

scheme for organizing the case memory, and enables model-based case validation and plan verification in the context of case-based path planning.

Representation and organization. Router's spatial model captures two kinds of knowledge: the pathways and the neighborhoods. The representation of a pathway contains three types of information:

- *Name of the pathway.* Curved pathways are broken into linear segments, which are named by appending a number to the base name of the pathway. Discontinuous pathways are also broken into segments.
- *Intersections between the pathways and their relative locations.* Intersections are specified as an ordered list that names all the other pathways that cross a given pathway *within* a particular neighborhood.
- *Allowable directions of traversal for the pathway.* Eight quadrants of the compass are used to provide a qualitative directional measure. Bidirectional pathways have a complementary direction-pair, such as (N S) or (NW SE). If a pathway allows travel in some direction in principle, but cannot be presently traversed due to some change in the navigation world, it is annotated as "blocked" in the given direction. Information about this kind of dynamic change in the world is acquired from feedback during plan execution.

Pathways are grouped into neighborhoods, which are organized in a space-subspace hierarchy as shown in Figure 1. This organization of spatial knowledge is based on two criteria:

- *Relative significance of pathways.* Significant pathways (such as those that serve as major thoroughfares connecting distal neighborhoods) are represented at a higher level in the hierarchy than less significant ones. In Figure 1, thick lines represent major pathways at the highest level; thin lines represent pathways whose importance is limited to their immediate vicinity.
- *Relative vicinity of pathways.* Pathways traversing the same localities in space are grouped together. Both major and minor pathways important for navigation in a given vicinity appear in the associated neighborhood.

The representation of a neighborhood contains three types of information: Names of the subneighborhoods; relative directions of subneighborhoods (based on eight quadrants of the compass); and pathways in the neighborhood. Pathways are represented separately for each neighborhood in which they occur. For example, a pathway may be represented in a high-level neighborhood as having four intersections (a, b c, and d in Figure 1), but as part of a lower-level neighborhood the same pathway might have seven intersections, with only two of the four (b and c) higher-level ones appearing in the lower-level neighborhood.

Adjacent neighborhoods at the same level may partially overlap so that an intersection situated roughly along their border can belong to both. This is important for ensuring that a search between two points in close spatial proximity is localized to a single neighborhood; if they are in different neighborhoods, a search in a higher-level neighborhood may result in a circuitous path between the two points. Note that Router's spatial model is only qualitative — it contains no quantitative information such as distances between locations.

Model updating. To explore some interactions between Router's model-based and case-based methods in the presence of dynamic changes in the navigation world, we endowed Router with the capability of updating its spatial model to reflect one kind of change: the "blocking" of a pathway in a specific direction of traversal. This model updating is directly based on the information in the feedback on the execution of a navigation plan. A user may provide the system reports on the execution of navigation plans in quasi-English. For example, suppose that a plan for navigating the Georgia Tech campus contains a step that specifies "`Go East on Ferst Avenue up to Atlantic Drive`." Suppose further that the user supplies the following feedback on the execution of the above plan step: "`Plan Step Go East on Ferst Avenue up to`

Atlantic Drive" failed because Ferst Avenue is blocked after intersection with State Street." Router would then parse this feedback and update its spatial model by annotating its representation of Ferst Avenue as blocked between State Street and Atlantic Avenue. The updating of the model is straightforward because the feedback explicitly specifies the cause of the failure of the plan step, and plan steps in Router's domain have no side-effects.

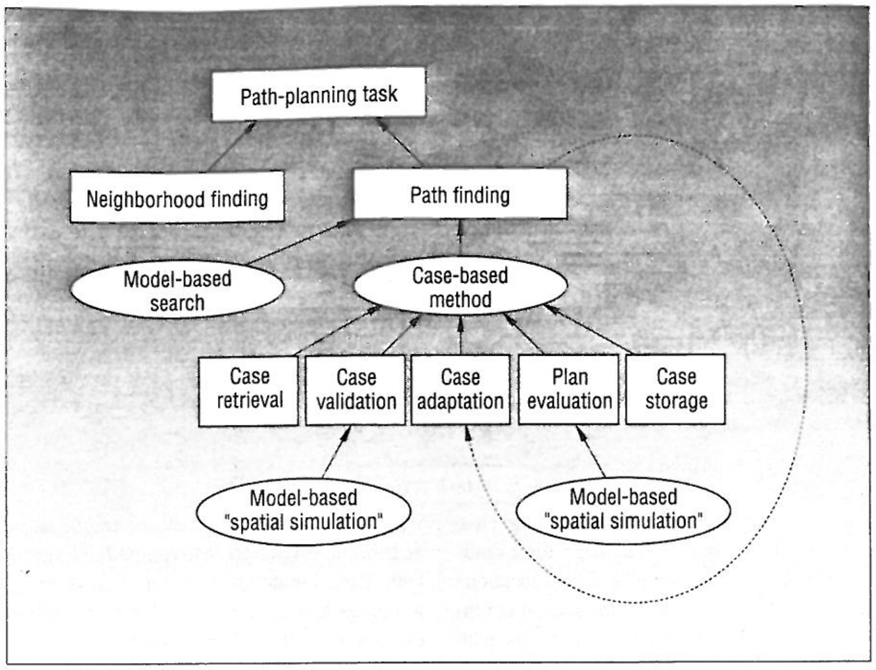

Figure 2. Router's task structure.

Model-based method

Router's model-based method forms navigation plans by performing a means-ends analysis on its qualitative spatial model of the navigation world. Given a specific path-planning task, the space-subspace hierarchy directly provides a decomposition of the goal. The neighborhoods in the hierarchy define the problem spaces associated with the subgoals, and the pathways in the neighborhood serve as operators in means-ends analysis.

Task structure and control strategy. As shown in Figure 2, at the highest level, Router sets up two subtasks of the path-planning task: the neighborhood-finding task, and the path-finding task.

The *neighborhood-finding* task identifies the neighborhoods of the initial and goal locations, and the direction of the neighborhood of the goal location relative to the neighborhood of the initial location. This is done by searching the space-subspace hierarchy for the two locations, starting with the highest-level neighborhood, then its subneighborhoods, and so on.

The *path-finding* task takes the initial and goal locations, their neighborhoods, and their relative directions as input and gives a path connecting the two locations as output. Depending on the input, one of three situations may arise.

The simplest situation is when the initial and goal locations are in the same neighborhood (for example, 1I and 1F in Figure 1). In this case, a straightforward search of pathways in the given neighborhood is undertaken.

In the second situation, initial and goal locations are in different neighborhoods on the same level (for example, 2I and 2F in Figure 1). Using the direction information in the input, the model-based method first attempts to determine a path in the higher-level neighborhood that connects the lower-level neighborhoods. This involves determining locations common to both the top-level neighborhood and the lower-level neighborhoods containing the initial and goal locations. A simple search procedure (described below) then finds a path from the initial location's neighborhood to the goal location's neighborhood. Next, the model-based method synthesizes a complete path by recursively applying the same search procedure to the lower level neighborhoods, and thus connects the initial and goal locations to the top-level path determined previously.

In the third scenario, the initial and goal locations are on different levels (3I and 3F in Figure 1). If the two locations are separated by only one level, the path-finding process is merely a simplification of the previous situation, because the top-level path already reaches one of the specified locations. If they are separated by more than one level, Router performs the same subtasks recursively to yield a legal path; that is, a "top-level" path is found within the lower-level neighborhoods to effectively link pathways occurring at different levels in the space-subspace hierarchy. Router first plans a top-level path in the highest neighborhood, but it cannot link this path directly to location 3I. It therefore treats the point at which the topmost partial route enters the mid-level neighborhood as a secondary final location (designated 3F′) and recursively links the paths from 3I to 3F′ to the path from 3F′ to 3F.

The procedure for searching pathways within a neighborhood combines lookahead and backtracking with a short horizon of three. The procedure begins by examining intersections adjacent to the starting intersection. Adjacent intersections are further examined up to three levels, or until the desired destination intersection is found. Whenever a sequence of intersections is unsuccessfully queried, these intersections are placed on a "used-intersections" list. This enables the procedure to look ahead and locate intersections not present in the list and focus its search on those. After one or two levels of search, if all of the next intersections appear on the used list, the procedure backtracks and begin a new line of search. This search procedure operates uniformly regardless of the particular neighborhood that is searched.

Case-based method

Router's case-based method combines means-ends analysis and plan reuse. The case memory is organized around the space-subspace hierarchy with the neighborhoods acting as indices to the cases. Given a specific path-planning task, the space-subspace hierarchy directly provides a decomposition of the goal, the neighborhoods in the hierarchy define the problem spaces associated with the subgoals, and the plans stored in the cases indexed by the neighborhoods act as "situation-specific macro-operators."

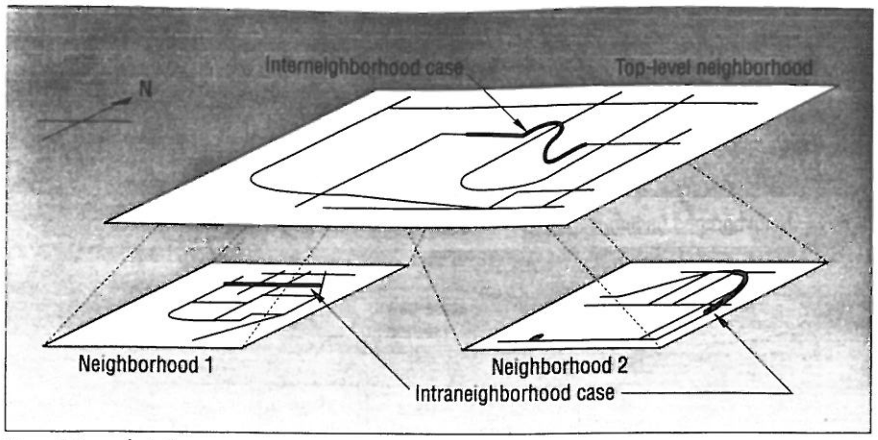

Figure 3. Case adaptation.

Representing cases and organizing case memory. A case in Router contains three kinds of information: The initial and goal locations in a past planning episode; the spatial neighborhoods the locations belong to; and the path connecting the two locations. Cases are indexed by the initial and goal locations of the stored plan (where the two locations act as the primary index), and by the spatial neighborhoods the locations belong to (where the neighborhoods act as the secondary index to the case).

The system's space-subspace hierarchy of neighborhoods shown in Figure 1 provides an indexing scheme for organizing case memory. A case whose initial and goal locations belong to neighborhoods X and Y, respectively, is indexed by those two neighborhoods. Thus, if X and Y are the same, then the case is stored as part of this neighborhood; if X and Y are different, it is stored as part of X, Y; and, if it contains the initial or goal locations, it is stored as part of the common super-neighborhood of X and Y.

Task structure and control strategy. At the highest level, Router sets up neighborhood-finding and path-finding as subtasks of the path-planning task. Router also uses the same procedure for the neighborhood-finding task described above. The system's case-based method, however, finds navigation paths differently. The case-based method sets up five subtasks of the path-finding task: case retrieval, case validation, case adaptation, plan evaluation, and case storage (see Figure 2).

In *case-retrieval*, the planner uses the output of the neighborhood-finding task as a probe into the case memory to search for cases that match the current problem as closely as possible. In particular, it searches the neighborhoods containing the two locations, first looking for cases exactly matching the specified task, then for partial matches. Exactly matching cases are those whose initial and goal locations are the same as those specified for a given path-planning task. Partial matches, in order of preference, are cases matching one of the two locations exactly, with the other location in the same neighborhood as the other location in the task specification and cases in which only the neighborhoods of the initial and goal locations match those of the task specification.

In general, three situations can result from this search: A case that exactly matches the specification of the current problem is available in memory; a case that only partially matches the current problem is available; or no case in memory even partially matches the specification of the current problem. In the first situation, the exactly matching case contains the solution to the current problem. The problem is solved simply by retrieving the previously planned route. In the second situation, the partially matching case is retrieved, and the previously planned route is adapted to arrive at a solution to the current problem. In the third situation, the case-based method alone cannot solve the current problem and terminates processing.

The next subtask is *case validation*. Although Router updates its spatial model based on the feedback on failed plans, it does not update the case memory, and thus the case-based method cannot by itself validate the retrieved plan. A specific pathway could occur in a large number of cases, and thus identifying and updating each case where the pathway occurs is likely to be computationally expensive. Since Router does update the spatial model, it can validate the retrieved plan by spatially "simulating" the plan. Thus, case validation is skipped when Router runs purely in the case-based mode.

Next, if case retrieval results in a case that only partially matches the specification of the given path-planning task, then the case-method attempts to adapt the retrieved plan to meet the task specification. The *case adaptation* subtask uses a recursive processing strategy for adapting the plan: It formulates path-planning subproblems, recursively spawns new path-finding subtasks, finds the solutions to the new path-finding subproblems, and combines their solutions with the initially retrieved route as shown in Figure 3.

Subproblem formulation involves determining the ways in which the path contained in the retrieved plan is incomplete for solving the current path-planning problem. It results in the formulation of one or possibly two new subproblems, for linking the endpoints of the retrieved path to the locations specified in the current problem. Recursive path planning is performed to solve the newly formulated subproblems. In the plan-synthesis phase, the solutions obtained in solving the subproblems are combined to the solution in the initially retrieved case. Thus, the case-based method forms new paths by combining partial solutions contained in multiple cases.

The next subtask is *plan verification*. Again, the case-based method by itself cannot evaluate the candidate navigation plan for the same reason it could not validate the retrieved case. This subtask is skipped when Router runs purely in the case-based mode.

The final subtask is *case storage*. If the case-based method is successful in combining previously planned paths to solve the given path-planning task, then it stores the newly found solution in its case memory. The new case is indexed by its initial and goal locations and the neighborhoods in which they lie.

In addition to complete plans, Router also stores partial plans in its memory. For example, if it has found a path to go from intersection *a* to intersection *z* — say *a*, *b*, *c*, *d*,..., *z* — then it stores the entire path as a case for potential reuse, since a future problem may require it to plan a path from *a* to *z* again. In addition, since the problem of going from any one intermediate location on the path to another — such as from *a* to *c*, *a* to *d*, *b* to *d* and so on — may also arise, the system also stores these "partial" paths as reusable cases.

Thus, Router automatically acquires additional cases as it solves new problems. It does so irrespective of whether the navigation plan is generated by the case-based method, the model-based method, or some combination of the two. As a result, as Router solves problems using the model-based method, it incrementally compiles general domain knowledge in the form of pathways into situation-specific cases.

Integration methods

Router's model- and case-based methods have method-specific control strategies. The case-based method, for example, sets up specific subtasks of a given task, and specifies the order of their execution under different knowledge conditions. The integration of the two methods, however, raises the additional question of strategic metacontrol: What method should Router select and invoke, given a specific task and a specific knowledge condition? Router's strategic metacontrol not only enables flexible and dynamic method selection, but also facilitates cooperation between the model- and case-based methods in solving problems and acquiring the knowledge needed for the problem solving.

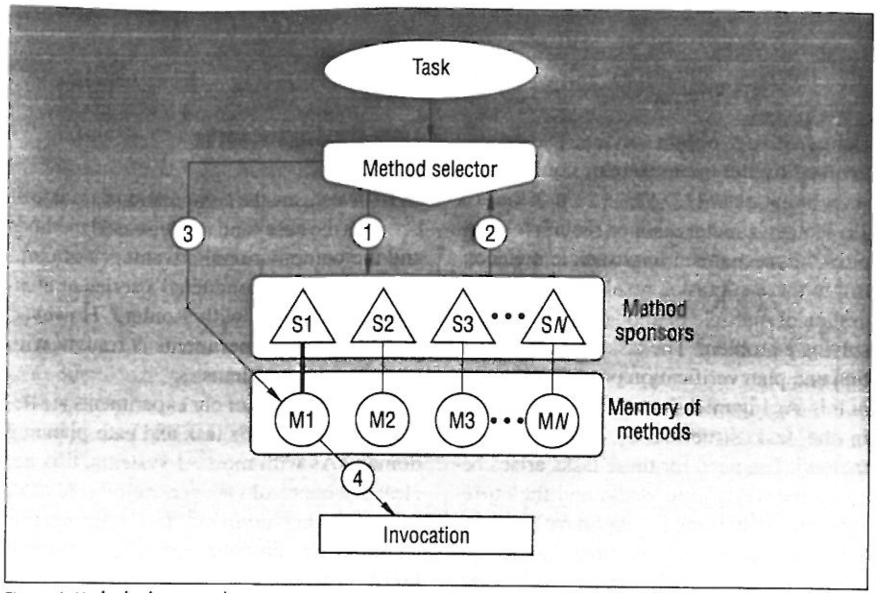

Figure 4. Method selection architecture.

Method selection. Router's mechanism for strategic metacontrol evolved from *task structures,*[3] where problem solving is described in the high-level vocabulary of tasks, the methods applicable to the tasks, and the content and form of knowledge used by methods as opposed to using low-level languages and architectures of implementation. Specifically, Router adopts and adapts the architecture for *task-integrated problem solving* (TIPS),[4] which addresses the issue of method selection based on the task-structures framework. Tips uses a *sponsor-selector* control structure[5] to select design plans in the context of routine design problems.

Router views strategic metacontrol as a design task with the goal of designing a virtual problem-solving architecture for addressing a given path-planning problem. It uses an "introspective" metareasoner for addressing this task. The metareasoner contains a method selector and a set of method sponsors, with one sponsor for each method. The general mechanism for method selection in the system is shown in Figure 4. Given a specific task *T*1, method selection occurs in four steps:

(1) The method selector uses the specification of *T*1 to probe the method sponsors associated with the methods *M*1 ... *MN.*
(2) The method sponsors inspect the memory and assess the applicability of the methods to *T*1.
(3) The method selector picks a specific method for *T*1.
(4) The method selector invokes the selected method.

A core issue in strategic metacontrol is deciding what criteria to use for selecting a method, given a specific task *T*1. In general, the selection of a specific method depends on three criteria:[6]

(1) *Desired properties of the solution.* Different methods can yield solutions with different properties, such as optimality or correctness.
(2) *Computational constraints of the task.* Different methods can have different computational requirements, such as processing time or memory size. These requirements must be considered with respect to the computational resources available to the problem solver.
(3) *Knowledge available to the problem solver.* Different methods use different types of knowledge, such as heuristics, associations, plans, cases, or models.

While all three criteria are important, the third is critical: If the knowledge used by a method is not available, then the problem solver cannot use the method regardless of other factors. In Router's metareasoner, the method sponsors are responsible for evaluation criterion 3 and the method selector is (or will be) responsible for criteria 1 and 2. (The evaluation of criterion 1 is not yet implemented in Router, and the evaluation of criterion (2) is implemented only partially and implicitly.)

The method sponsor associated with a method in Router has two kinds of information:

(1) *The tasks for which a method is applicable.* For example, in addition to path finding, the model-based method is useful for case validation and plan evaluation as described below.
(2) *The knowledge needs of the method.* For example, the sponsor for the case-based method has knowledge that the method needs a case that at least partially matches the specification of the given task.

Given a specific task, the method sponsor checks whether its method is applicable to the task. If so, the method sponsor inspects the store of knowledge, and verifies if the knowledge needed by the method is available. Given this, it offers the method to the method selector for possible invocation. For example, given a specific path-finding task, the sponsor for the case-based method first uses information about the task to determine if the case-based method is applicable. The sponsor then inspects the case memory to determine whether a case that at least partially matches the task specification is available in memory. If it finds a relevant case, the sponsor offers the case-based method to the method selector for possible invocation.

Early experiments with Router revealed that if a case similar to a given problem is available in memory, then in general, the system's case-based method is computationally more efficient than its model-based method.[7] This is because the similar case immediately provides a solution in the "neighborhood" of the desired plan. Thus, the method selector in Router is biased toward the case-based method: If a case relevant to a given task is available in memory, then it invokes the case-based method instead of the model-based method; the model-based method is invoked only if a case relevant to the given task is unavailable.

Cooperative problem solving. The method invoked by the metareasoner could set up several subtasks *T*11, *T*12, ..., *T*1*M* of the task *T*1. Router's metareasoner recursively applies the mechanism for strategic metacontrol to these subtasks, resulting in the integration of multiple methods in the course of solving a problem. The tasks of case validation and plan verification provide examples of this. As Figure 2 shows, these tasks occur in the task structure of the case-based method. The need for these tasks arises because the navigation world and the corresponding spatial model may have changed. The case-based method alone cannot accomplish these tasks. Instead, these tasks require knowledge of the updated spatial model. For example, Router validates a retrieved case by spatially "simulating" the plan stored in the case, that is, by using the spatial model to trace the plan steps. Similarly, it uses its spatial model for simulating, and thus evaluating, a plan generated by the case-based method.

The task of case adaptation illustrates a different but related point. Let us suppose that the case-based method is selected to solve task instance *T*1. If the retrieved case does not exactly match *T*1, then Router has to select a method for adapting the retrieved case. The adaptation task is phrased in the same form as the initial task specification — as finding a path that connects the ends of the path retrieved by the case-based method to the initial and goal locations specified in *T*1. This is presented to the method selector, which uses the method-selection mechanism recursively to select a method applicable to the adaptation task. If it can find a case similar to the new task instance, then the method selector again invokes the case-based method. But if no such case is available, it invokes the model-based method. When the model-based method achieves a solution to the adaptation task, the control of processing returns to the case-based method, and the complete plan is synthesized by linking the segment found by the latter method to the segment found by the former.

Router's model- and case-based methods cooperatively solve path-planning problems. In fact, the two methods also cooperate in acquiring the knowledge needed for the problem solving: A plan generated by the model-based method is stored as a case for potential reuse by the case-based method, and the spatial model is revised based on experience with specific plans.

The experiments

To investigate the computational trade-offs between the case- and model-based methods and the computational advantages of combining them, we conducted a series of ablation experiments with Router.[8] However, analysis of the experiments is fraught with some serious problems.

First, the results of our experiments are dependent on Router's task and path-planning domain. As with most AI systems, it is not clear that our results are generalizable to other tasks in other domains. Determining this would require similar experiments in the context of different tasks in different domains.

Second, our results are also dependent on the design of the Router system. For example, the theories of model-based search, case-based plan reuse, and task-directed integration of multiple methods can be instantiated in a number of ways. In fact, neither the model-based method nor the case-based method in the system is optimal. In designing the system, we generally opted for simplicity rather than optimality. Thus, it is quite possible to improve Router's model- and case-based methods in terms of their problem-solving coverage, their processing efficiency, and the quality of solutions they produce. But these are only some of the dimensions along which a given method can or should be analyzed. The cognitive plausibility of the method, for example, is another dimension of evaluation — one that our experiments with Router do not explore.

TO INVESTIGATE THE TRADE-OFFS BETWEEN THE CASE- AND MODEL-BASED METHODS, WE CONDUCTED A SERIES OF EXPERIMENTS WITH ROUTER. HOWEVER, ANALYSIS OF THE EXPERIMENTS IS FRAUGHT WITH SOME PROBLEMS.

In sum, these experiments evaluate the system, not the theories underlying it. Nevertheless, we believe that such experimental evaluations are important. They provide data points for comparison with other systems, raise additional issues about the theories implemented in the systems, and indicate the kinds of experiments needed to resolve them. In addition, ablation experiments can help guide the design of a system. For example, early ablation experiments with Router indicated that its case-based method is more efficient than its model-based method. This resulted in our biasing the method selector in Router toward the case-based method in subsequent designs.

Navigation task and domain. Router operated with representations of the Georgia Tech campus in Atlanta, including specific floors in the College of Computing building. The input to Router was a pair of spatial locations representing the initial and goal positions that the path should connect. The two locations were among the intersections between the pathways; the pathways were the streets in the campus domain, and the corridors and hallways in the College of Computing domain. The pathways could be uni- or bidirectional, and more than two pathways could intersect at a given point. The system output was an ordered set of path segments (streets, hallways) between the initial and the goal locations.

The campus yielded about 10,000 problems, and the computing building about 1000. We conducted most of the experiments with Router using 10 sets of 50 path-planning problems each, where the problems and their order within a problem set were generated randomly. Since 50 problems may be too small a number for testing some of the above hypotheses, we conducted some experiments on a larger set of 1000 problems.

In all experiments involving multistrategy reasoning, the case memory was empty at the beginning of the experiment, but grew as subsequent problem-solving cases were stored in it. In the experiments involving the exclusive use of the case-based method, the case memory had to be primed by adding (randomly formed) cases to the memory before conducting the experiments. All experiments were conducted on a dedicated Sun workstation.

Experiment 1. We designed the first set of experiments to test two related hypotheses regarding the computational efficiency of Router's model-based method, case-based method, and multistrategy reasoning:

(1) Router's case-based method is computationally more efficient than its model-based method, hence,

(2) Router's integration of case-based and model-based method results in processing at least as efficient as that of the model-based method.

The data from experiment 1 confirms Hypothesis 1: When appropriate cases can be found in memory to execute Router's (pure) case-based method, this method is indeed more efficient than either the model-based or integrated methods (see Figure 5). The reason is that the case-based method reuses old plans rather than forming new ones — an old case that closely matches a given problem rapidly provides a solution in the "neighborhood" of the desired plan. In addition, this data shows that Hypothesis 2 is false: When the number of cases in memory is small, Router's model-based method performs faster than its integrated method on approximately 92 percent of the problems.

Experiment 2. In the second set of experiments, we set out to test two related hypotheses regarding the quality of solutions produced by Router's model-based, case-based, and integrated methods:

(1) The case-based method produces solutions of quality equal to those produced by the model-based method; hence,
(2) The integrated method produces solutions of quality at least equal to those produced by the model-based method alone.

Testing these hypotheses raises the issue of how to measure the quality of a solution. In the domain of navigational path planning, the logical answer is that a shorter navigation plan is better than a longer one. However, since Router contains no quantitative information (such as distances between locations), the issue becomes how to measure the shortness of a navigation plan. We used the number of path segments in a navigation plan for this purpose, where a path segment is defined as the path between two consecutive street changes in the overall path:

The data from experiment 2 shows that both hypotheses are false: In general, Router's case-based method produced plans that were not as parsimonious as the plans produced by its model-based method, and, as a result, the integrated method also produced less parsimonious plans. The model-based method *always* produced paths with a smaller or equal number of path segments than the case-based method.

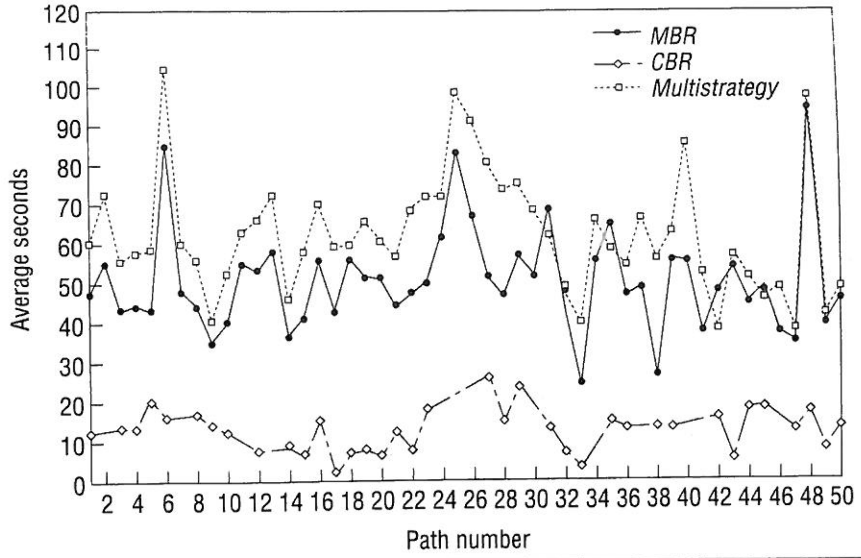

Figure 5. Comparison of average problem solving times for different strategies.

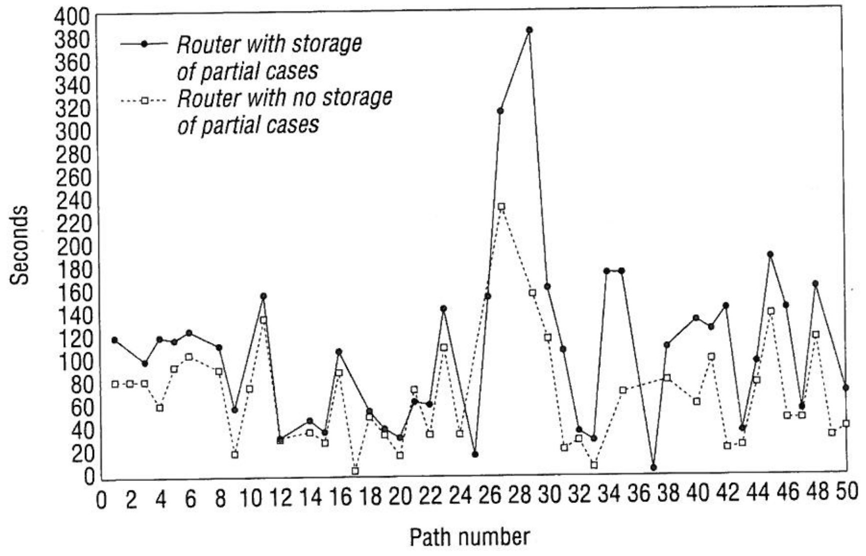

Figure 6. Effects of partial cases.

Experiment 3. We designed the third set of experiments to test a hypothesis about the decomposition of cases into partial cases: The decomposition of cases into partial cases (at storage time) results in more efficient problem solving (on future problems).

The *prima facie* justification for this hypothesis is that, in general, storing partial cases would enable the case-based method to retrieve a case more appropriate to a given problem, and this would reduce the computational cost of adapting it to meet the specifications of the problem. Testing this hypothesis raises the question of what a reasonable partial case is. In the domain of navigational path planning, the logical answer is that partial cases correspond to a subset of the original plan, that is, a path from two intersections specified in the original plan.

The data from experiment 3 shows that the hypothesis is false. Problem-solving time was actually increased by the decomposition of cases into partial cases and the storage of these partial cases. On average, the problem-solving time for the entire path-planning process, including the storage of partial paths, was 1.7 times longer than the problem-solving time without storage of partial paths (see Figure 6). (Because of this result, all other experiments were conducted without storing partial cases.) We expected that the use of partial cases would add to the cost

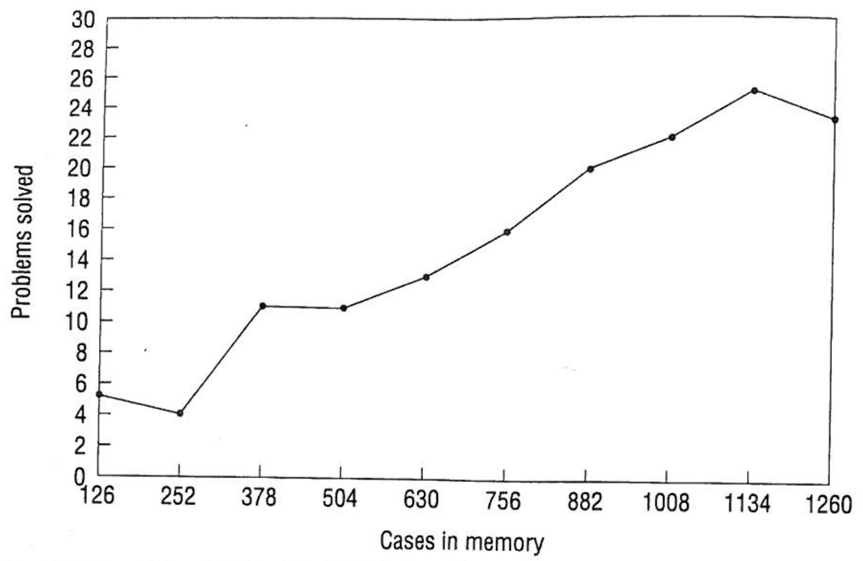

Figure 7. Number of problems solved (out of 50) with different amounts of memory bootstrapping for the pure case-based method.

of retrieving appropriate cases from memory, because the memory would contain more cases. Our analysis of the data, however, indicates that the added cost of retrieval is small compared to the added cost of decomposing the case into partial cases and storing the partial cases in memory.

Experiment 4. We designed this set of experiments to test two related hypotheses about the problem-solving coverage and knowledge requirements of Router's case-based method:

(1) The case-based method can be bootstrapped with relatively few cases in memory.
(2) The case-based method has the same problem-solving coverage as the model-based method, even though its knowledge requirements are smaller.

The data from experiment 4 shows that while hypothesis 1 is true, hypothesis 2 is false. While Router's case-based method can indeed solve some problems with relatively few initial cases in the case memory, it covers fewer problems than its model-based method. This data also shows that the number of problems that can be solved by the case-based method increases linearly with the initial number of cases in memory (see Figure 7). Specifically, we needed to seed Router's case-based method with solutions to approximately 16% of all the possible problems in our domain before it could solve approximately half of the problems given to it.

OUR EXPERIMENTS WITH ROUTER suggest several general lessons for designing multistrategy adaptive navigational path planners.

First, the use of case-based methods for navigational path planning requires some measure for estimating the spatial closeness between a given problem and a case stored in memory. The notion of spatial neighborhoods provides one such measure — a new path-planning problem and a stored case are spatially close if they belong to the same neighborhood. This suggests a scheme for indexing the stored cases by the neighborhoods to which they belong.

Second, the spatial neighborhoods can be organized in a space-subspace hierarchy based on the notion of relative significance and vicinity of pathways in the navigation world. The computational advantage of this organization is that the space-subspace hierarchy directly provides the goal decomposition of a path-planning problem, and the neighborhoods in the hierarchy define the problem spaces corresponding to the subgoals.

Third, if the navigation world is dynamic, then the adaptive approach to navigational path planning is faced with the issue of updating the case memory. This might be computationally costly if the case memory is large, and it might be relatively easier and cheaper to update the spatial model. But if only the spatial model is updated, then the case-based method by itself may not be able to validate the cases retrieved from memory or verify the candidate plans. Instead, the tasks of case validation and plan verification may require the capability of model-based spatial simulation.

Fourth, the integration of multiple methods for solving a given task leads to more robust reasoning. Given a specific instance of the task and a specific knowledge condition, if a particular method is not useful because the knowledge it uses is not available, then another method can be brought into service if the needed knowledge is available in the given situation. However, the mechanism for integrating the methods itself incurs computational costs because of the additional processing needed for method selection.

Finally, spatial models of the navigation world play multiple roles in navigational path planning. For example, in Router, spatial models define the problem spaces to be searched by the model-based search method, enable spatial simulation for validating cases and verifying new plans, and provide an indexing scheme for organizing the case memory.

Acknowledgments

This work has benefited from many discussions with members of the AI groups at Ohio State University and Georgia Institute of Technology. We are grateful to B. Chandrasekaran at Ohio State, who suggested the problem of navigational path planning and commented on earlier drafts of this article, and to Eleni Stroulia at Georgia Tech for her help with the analysis and modeling of Router. This research has been supported by a CER infrastructure grant from the National Science Foundation (CCR-86-19886), and research grants from the National Science Foundation (IRI-92-10925) and the Office of Naval Research (N00014-92-J-1234).

References

1. J. Kolodner, *Case-Based Reasoning,* Morgan Kaufmann, San Francisco, 1993.
2. R. Alterman, "Adaptive Planning," *Cognitive Science,* Vol. 12, No. 3, 1988, pp. 393-421.
3. B. Chandrasekaran, "Task Structures, Knowledge Acquisition, and Learning," *Machine Learning,* Vol. 4, No. 3/4, Dec. 1989, pp. 339-345.
4. W. Punch, *A Diagnostic System Using Task-Integrated Problem Solving Architecture, Including Causal Reasoning,* doctoral dissertation, Dept. of Computer and Information Science, Ohio State Univ., 1989.
5. D. Brown and B. Chandrasekaran, *Design Problem Solving: Knowledge Structures and Control Strategies,* Pitman, London. 1989.
6. A. Goel and B. Chandrasekaran, "Case-Based Design: A Task Analysis," in *Artificial Intelligence Approaches to Engineering Design, Volume II: Innovative Design,* C. Tong and D. Sriram, eds., Academic Press, San Diego, 1992, pp. 165-184.

7. A. Goel et al., "An Integrated Experience-Based Approach to Navigational Path Planning for Autonomous Mobile Robotics," *Proc. IEEE Int'l Conf. Robotics and Automation*, IEEE Computer Society Press, Los Alamitos, Calif., 1993, pp. 818-825.

8. P. Cohen and A. Howe, "How Evaluation Guides Research," *AI Magazine*, Vol. 9, No. 4, Winter 1988, pp. 35-43.

Ashok K. Goel is an assistant professor in the College of Computing at Georgia Institute of Technology. His research interests include multistrategy problem solving, qualitative models and model-based reasoning, case-based reasoning and learning, and computer-aided design. He received his MS in physics and his PhD in computer and information science from Ohio State University. He has served on several conference program committees, including AAAI-92 and AI in Design-94, and was a Lilly Teaching Fellow in 1992-93. Ashok K. Goel can be reached at the College of Computing, Georgia Institute of Technology, 801 Atlantic Drive NW, Atlanta, GA 30332-0280; Internet: goel@cc.gatech.ed

Khaled S. Ali is working on his PhD at Georgia Institute of Technology. His research interests include multiagent robotics, robot path planning, and teleoperation of robot societies through teleautonomy and modification of abstract personality traits. He received his BS in computer science from Vanderbilt University in 1992. Khaled Ali can be reached at the College of Computing, Georgia Institute of Technology, 801 Atlantic Drive NW, Atlanta, GA 30332-0280; Internet: kali@cc.gatech.edu.

Michael W. Donnellan is a graduate student in computer science at Georgia Institute of Technology, and works as a software engineer at AT&T Global Information Solutions Human Interface Technology Center in Atlanta. His research interests include dynamically adaptive human-computer interface design; application of computing technology to training and education; and the study of best practices in software engineering. He received his BS in information and computer science from Georgia Tech in 1991. Michael Donnellan can be reached at AT&T Global Information Solutions, Human Interface Technology Center, 500 Tech Parkway NW, Atlanta, GA, 30313; Internet: mwd@cc.gatech.edu.

Andres Gomez de Silva Garza is a PhD candidate at the University of Sydney, Australia, where he is doing research on AI in computer-aided design. He earned his BS cum laude in computer engineering from the National Autonomous University in Mexico, and worked at IBM of Mexico before entering Georgia Tech where he earned an MS in computer science. Andres Garza can be reached at the Key Centre of Design Computing, University of Sydney, NSW 2008, Australia; Internet: andres@arch.su.edu.au.

Todd J. Callantine is working on his PhD in human-machine systems at Georgia Institute of Technology. His research interests include automation in complex dynamic systems, and applications of model- and case-based reasoning systems for operator training and support. He received his BS in chemical engineering and his MS in operations research from Stanford University. Todd Callantine can be reached at the Center for Human Machine Systems Research, School of Industrial and Systems Research, Georgia Institute of Technology, 801 Atlantic Drive NW, Atlanta, GA 30332; Internet: tc@chmsr.gatech.edu.