GIT-CC-95-51

Some Experimental Results in Multistrategy Navigation Planning

Ashok K. Goel
Khaled S. Ali
Eleni Stoulia
Some Experimental Results
in Multistrategy Navigation Planning

Ashok K. Goel*
Khaled S. Ali
Eleni Strouilias
College of Computing
Georgia Institute of Technology

March 26, 1996

Abstract

Spatial navigation is a classical problem in AI. In this paper, we examine three specific hypotheses regarding multistrategy navigation planning in visually engineered physical spaces containing discrete pathways: (1) For hybrid robots capable of both deliberative planning and situated action, qualitative representations of topological knowledge are sufficient for enabling effective spatial navigation; (2) For deliberative planning, the case-based strategy of plan reuse generates plans more efficiently than the model-based strategy of search without any loss in the quality of plans or problem-solving coverage; and (3) For the strategy of model-based search, the “principle of locality” provides a productive basis for partitioning and organizing topological knowledge. We describe the design of a multistrategy navigation planner called Router that provides an experimental testbed for evaluating the three hypotheses. We also describe the embodiment of Router on a mobile robot called Stimpy for testing the first hypothesis. Experiments with Stimpy indicate that this hypothesis apparently is valid for hybrid robots in visually engineered navigation spaces containing discrete pathways such as office buildings. In addition, two different kinds of simulation experiments with Router indicate that the second and the third hypotheses are only partially correct. Finally, we relate the evaluation methods and experimental designs with the research hypotheses.

Keywords: Spatial navigation, navigation planning, hybrid architectures, and multistrategy reasoning.

*Contact: Ashok K. Goel, College of Computing, Georgia Institute of Technology, 801 Atlantic Drive NW, Atlanta, GA 30332-0280. Phone: (404)853-3971. FAX: (404)853-3978. Internet: goel@cc.gatech.edu.
1 Background, Motivations and Goals

Spatial navigation is a classical problem in AI and path planning is a core task in spatial navigation (e.g., [Davis 1986; Kuipers and Levitt 1988; Lawton and Levitt 1990; McDermott and Davis 1984]). The task of path planning takes a specification of an initial and a goal location in a physical world as input, and gives a specification of a path connecting the two locations as output. A theory of path planning is constrained by several factors, some of which are closely related:

1. *The nature of the navigation space.* Physical worlds can range across a multidimensional spectrum. In one dimension, the world may be continuous (e.g., a soccer field) or it may contain discrete pathways (e.g., interstate highways). In another dimension, the world may contain distinctive places (e.g., a T junction in an office corridor) or landmarks (e.g., the tallest building in downtown) or it may contain no known distinctive places or landmarks. In yet another dimension, the world may be static, gradually evolving or rapidly changing.

2. *The engineering of the navigation space.* Some worlds can be easily engineered to facilitate navigation while others cannot. For example, factory floors often can be structured both spatially, by making the pathways broad enough to allow the passage of a robot, and visually, by posting visual cues to guide the robot in navigating the pathways.

3. *The motor and perception capabilities of the robot.* Robots have limited motor and perceptual capabilities. For example, a robot’s may be limited to locomotion on flat surfaces; it may be incapable of navigating stairs. Similarly, a particular robot may be capable of reading only visual barcodes; it may be incapable of recognizing other objects.

4. *The cognitive capabilities of the robot.* A robot capable only of situated action may have no explicit representation of any knowledge of the world. A robot capable of deliberative planning may use different kinds of world knowledge and problem-solving strategies. A hybrid robot may combine deliberative planning and situated action.

5. *The knowledge constraints.* Different kinds of knowledge may be available for navigation planning in different worlds. For example, in one world the robot may have only qualitative topological knowledge
of the world, in another world it may have metrical knowledge in addition to qualitative knowledge, and, in yet another world, it may have access to a memory of past navigation plans in addition to a topological model of the world.

6. *The computational constraints.* The general task of path planning may be further constrained by requirements on the properties of the navigation plans (e.g., optimality, correctness), or on the properties of the process for producing the plans (e.g., efficiency, completeness), or both.

In this paper, we examine three specific hypotheses concerning spatial navigation in visually engineered worlds containing discrete pathways. Examples of physical worlds containing discrete pathways include urban areas, which contain expressways, roads and streets, and office floors, which contain hallways and corridors. Traversal in such worlds is limited to the discrete pathways.\(^1\) By visual engineering we mean artificial structuring of the world by posting visual cues for guiding the spatial navigation. In case of human navigation, examples again include urban areas, in which each street intersection contains a unique sign that distinguishes it from other street intersections for example, and office floors, where specific corridors and offices can be similarly detected by reading posted signs. In the case of robot navigation, the visual cues would need to match the perceptual capability of the robot. For example, the visual cues may be in the form of visual barcodes if the robot has the capability of reading such barcodes.

Note that this characterization of the navigation space includes worlds in which obstacles are static and also worlds with moving obstacles. In particular, it admits worlds in which obstacles move at a rate slower than both the processing-cycle and the motor-actuation times of the robot. A person walking in an office corridor is an example of this kind of moving obstacle, provided that the person is walking at a pace slower than the time taken by the robot to process the perceptual input and activate motor actions.

We now describe the three hypotheses in our experimental study including the background and motivation for each. The hypotheses are related in that they revolve around the common theme of multistrategy reasoning. The first hypothesis pertains to the knowledge boundary between deliberative planning and situated action in the context of a hybrid robot. The second hypothesis concerns the computational trade-offs between the strategies of model-based search and case-based reuse in the context of deliberative navigation.

\(^1\)Communication networks provide another, though abstract, example of navigation spaces containing discrete pathways.
planning. The third hypothesis concerns the principle for partitioning and organizing navigation spaces in the context of model-based search.

*Hypothesis 1:* For hybrid robots capable of both deliberative planning and situated action, qualitative topological representations are sufficient for effective spatial navigation in visually engineered worlds containing discrete pathways. Most deliberative path planning algorithms use metrical knowledge in addition to qualitative topological knowledge (e.g., [Latombe 1991]). This is true even for hybrid robot architectures capable of both deliberative planning and situated control (e.g., [Arkin 1989a]). However, metrical knowledge of the kind required by path planning algorithms often may not be available for many navigation spaces. Further, the (implicit) assumption that effective spatial navigation requires the use of metrical knowledge by the path planner remains largely unexamined in case of hybrid robots. Given the capability of situated action in a hybrid robot, the issue thus becomes whether the deliberative path planner may use only qualitative topological knowledge and yet enable effective spatial navigation.

*Hypothesis 2:* For deliberative navigation planning, the case-based strategy of reusing past plans results in more efficient problem solving than the model-based strategy of searching a model of the navigation space, without any significant loss in the quality of plans or the problem-solving coverage. Most deliberative path planning algorithms work by searching the problem space characterized by the spatial model of the world (e.g., [Latombe 1991]). Case-based reuse of past plans (e.g., [Alterman 1988]; [Hammond 1989]) offers an alternative strategy. In this strategy, new problems are solved by retrieving and adapting similar plans. But the claim that this strategy produces new plans more efficiently than model-based search without any loss in the quality of solutions (see, e.g., [Kolodner 1993]) remains largely unexamined. The issue thus becomes whether the case-based strategy of plan reuse generates navigation plans more efficiently than the strategy of model-based search and whether the former strategy also leads to navigation plans of equal quality as the latter. A related issue is whether the case-based strategy can be bootstrapped with a small number of cases relative to the number of all possible problems in the navigation space.

*Hypothesis 3:* For the strategy of model-based search, the "principle of locality" provides a productive basis for partitioning and organizing the navigation space. Specific methods in the family of model-based search methods for path planning use different schemes for partitioning and organizing the navigation space.
(e.g., [Latombe 1991]). Most of these methods are (implicitly) based on the “principle of locality”: spatially
adjacent locations are aggregated into regions of space and organized in a space-subspace hierarchy. The
differences between the specific schemes apparently lie not in the principle but in the details of the
partitioning and the organization of the spatial regions. The issue thus becomes whether, for navigation
spaces containing discrete pathways, different realizations of the principle of locality make a significant
difference to either the efficiency of problem solving or the quality of plans.

Note that the engineering of an optimal path planner is not the goal of our work. In fact, since our
path planner, called Router, uses only qualitative spatial knowledge in the context of a hybrid robot
architecture capable of situated action, it does not provide any prior guarantee of optimality. Instead,
our goal is to examine and evaluate three specific hypotheses of some intrinsic merit in the context of
navigation planning. In sum, the results of our experiments confirm Hypothesis 1 (which we discuss later).
But the experimental results pertaining to Hypotheses 2 and 3 initially surprised us.

The rest of this paper is organized as follows. In the next section, we discuss some of the evaluation issues
raised by our hypotheses and describe the general design of our experiments for addressing them. Then,
we describe the design of a deliberative path planner called Router that provides a basis for conducting our
experiments. Router combines the model-based search and case-based plan reuse. In sections 4, 5, and 6,
we refine the three hypotheses, describe our experimental designs, and report the results. The evaluation
of Hypothesis 1 involves the embodiment of Router on a mobile robot called Stimpy, described in section
4. In section 7, we discuss the experimental results, relate them to similar work, and conclude the paper.

2 Evaluation Methods and Experimental Designs

The three hypotheses described above are of three different kinds in that they raise different evaluation
issues and thus require different experimental designs. In this section, we discuss some of the evaluation
issues and describe the general design of our experiments.

The examination of the first hypothesis requires the design and construction of a path planner which

\footnote{This principle is a derivative of the principle of locality that Marr [1979] originally described in his work on computer vision.}
uses only qualitative spatial knowledge and its integration with a mechanism for situated action. A difficult evaluation issue here concerns realism. In principle, the hypothesis can be evaluated by simulation. One problem with this method of evaluation is that the simulation environment is bound to make a number of assumptions not only about the navigation task and domain but also about the robot’s motor and perceptual capabilities. The results of the evaluation then would be strictly limited by these assumptions. Another problem with evaluation by simulation is that it cannot accommodate “unknown factors” such as noise.

For this reason, we evaluated the first hypothesis on a real robot. But this raises another difficult evaluation issue: our experiments are constrained by the motor and perceptual capabilities of the available robot. For example, the robot readily available to us, a Denning MRV-2, is capable of locomotion only on flat surfaces such as an office floor. The robot’s sensors include shaft encoders, a ring of 24 ultrasonic sensors, and a LaserNav rotating barcode reader. The shaft encoders provide information about the orientation of the robot and the distance traversed from the initial location. The ultrasound sensors enable the detection of static and moving obstacles. Only the LaserNav barcode reader allows the detection of distinctive places or landmarks in the navigation space. This meant that we had to visually engineer the office floor by posting barcodes that could be recognized by the robot. First, we designed and constructed a deliberative path planner called Router that uses only qualitative topological knowledge of the navigation space [Goel and Callantine 1992; Goel et al 1993]. Then, we integrated Router in a hybrid robot architecture called AuRA that is due to Arkin [1989a]. We call the resulting system Raura. Next, we embodied Raura in a Denning MRV-2 robot that we call Stimpy. Finally, we gave the robot various navigation tasks and monitored its actions as it executed the tasks in the visually structured office floor.

The examination of the second hypothesis requires the design and construction of a path planner capable of using both model-based search and case-based reuse. The two methods then can be compared and evaluated through ablation experiments (see [Colen and Howe 1988]). Router is a multistrategy system capable of dynamically spawning tasks, and flexibly selecting and integrating multiple methods. The selection and integration of the two methods in Router is task directed. This enabled us to evaluate not only the methods of model-based search and case-based reuse but also their combination.
But the second hypothesis too raises a difficult evaluation issue: the results of our experiments are dependent on the design of the Router system. For example, model-based search, case-based plan reuse, and task-directed integration of these methods, all can be instantiated in a number of ways. In fact, neither the model-based method nor the case-based method nor their integration in Router is optimal. In designing Router, we generally opted for simplicity rather than optimality. Thus, it is quite possible to make Router's model-based search, case-based reuse, and task-directed integration more optimal in terms of problem-solving coverage, processing efficiency, and quality of solutions they produce. In designing Router, we tried to keep the design decisions explicit. This enabled us to vary some of its design parameters and to explore their effect on system's performance. Thus, the experimental results pertain to the class of path planners instantiated in Router, rather than a specific implementation, where the class is characterized by the allowed variations in the system's design parameters.

The examination of the third hypothesis requires the design and construction of different knowledge organizations for model-based search. The difficult evaluation issue here is how to craft different organizations of the topological model based on the same principle of locality, without letting our (the experimenter's) tacit knowledge influence the crafting. To address this issue, we handcrafted Router's initial topological model which partitions a navigation space into spatial neighborhoods and organizes the neighborhoods in a space-subspace hierarchy, based on the principle of locality. But then we developed a system called Meta-Router that works with Router and is capable of reflecting on Router's problem-solving experiences and reorganizing its topological knowledge, e.g., tweaking the sizes of the neighborhoods in its space-subspace hierarchy. Meta-Router's reorganization of Router's knowledge also abides by the principle of locality.

In addition, Meta-Router is capable of taking a “flat” map of the navigation space and a specific set of problem-solving examples as training input, partitioning the space into neighborhoods and organizing them in a hierarchy based on the principle of locality, and giving a space-subspace model as output. Meta-Router's learning method itself is not important for the present discussion because the learning of the space-subspace hierarchy is not the research issue here (see [Stroulia and Goel 1995] for details of Meta-Router). It is important to note only that, depending on the specific set of examples in the training input, the method is capable of organizing and reorganizing topological knowledge without involving handcrafting. This learning capability enables us to conduct experiments for evaluating the third hypothesis.
3 Router: Integrating Model-Based Search and Case-Based Plan Reuse

The Router system instantiates a multi-strategy theory of navigation planning in physical worlds which contain discrete pathways and in which traversal is confined to the discrete pathways. Router's current knowledge enables its operation in two kinds of domains within this class of worlds: a representation of the Georgia Tech (GT) campus in Atlanta, and representations of specific floors in the College of Computing (CoC) and the Manufacturing Research Center (MaRC) buildings on the Georgia Tech campus. In the domain of the Georgia Tech campus, the roads and streets are the discrete pathways, and, in the domains of CoC and MaRC office buildings, the corridors and hallways are the pathways. The pathways (streets, hallways) may be uni- or bi-directional, and more than two pathways may intersect at a given point.

In both navigation domains, the input to Router is a pair of spatial locations representing the initial and goal locations of the robot. The initial and goal locations are among the intersections between the pathways in the world. The output of the planner is an ordered set of path segments (segments of streets, hallways) connecting the initial and the goal locations.

Router is a multistrategy navigation planner: it dynamically integrates the model-based strategy of search and the case-based strategy of plan reuse. It uses a task-directed mechanism for this integration of multiple methods. However, it uses only qualitative spatial knowledge: it contains no metrical knowledge such as the length of a pathway or the distance between two intersections. Also, its organization of its spatial model is based on the principle of locality.

3.1 Topological Model

The qualitative topological model of the navigation space plays multiple roles in Router: it defines problem spaces for the model-based method, it provides the indexing scheme for organizing the case memory, and it enables model-based case validation and plan verification in the context of case-based path planning.

Representation and Organization of Topological Model Since Router operates in navigation spaces containing discrete pathways, its topological model of a navigation space explicitly represents known path-
ways. In addition, it groups the pathways into neighborhoods and organizes them in a space-subspace hierarchy. The representation of a pathway contains three types of information:

1. **Name of the pathway.** Curved pathways are broken into linear segments. The segments are named by appending a number to the base name of the pathway, and act like any other pathway. Discontinuous pathways are similarly broken into distinct segments.

2. **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.

3. **Allowed directions of traversal for the pathway.** Eight octants of the compass are used to provide a qualitative directional measure. Bidirectional pathways have a complementary direction-pair (e.g., (N S) or (NW SE)). If a pathway in principle allows travel in some direction but cannot be presently traversed due to some change in the navigation world, then it is annotated as ‘blocked’ in the given direction. Information about this kind of change in the world is acquired from feedback on the execution of a plan.

The navigation space is partitioned into neighborhoods and the neighborhoods are organized in a space-subspace hierarchy in accordance with the principle of locality. The space-subspace hierarchy provides a decomposition of the navigation space: children neighborhoods decompose the space covered by their parent neighborhood into subspaces. Figure 1 illustrates the space-subspace hierarchy for the GT campus.

In Router, the principle of locality is operationalized in the following two heuristics:

1. **Proximity of pathways.** Spatially adjacent pathways (and thus spatially adjacent intersections) are grouped together into neighborhoods. Adjacent neighborhoods at the same level in the space-subspace hierarchy may partially overlap so that an intersection situated close to their border can belong to both. This helps to insure that the search for connecting two points in close spatial proximity is localized to a single neighborhood; if they are in different neighborhoods, search in a higher-level neighborhood may result in a circuitous path between the two points. A neighborhood may contain both major and minor pathways as described below.

2. **Significance of pathways.** A more significant pathway connects more distant neighborhoods, and is
This replaces Figure 1. This figure is the same as Figure 1 in


To obtain a copy of this report with the figures included, please contact the authors.

Figure 1: Hierarchical Topological Model

represented at a higher level in the hierarchy than less significant ones. In Figure 1, thick lines represent major pathways (e.g., Tenth Street), and thin lines represent relatively minor pathways whose importance is limited to their immediate neighborhood (e.g., Curran Street). Higher-level neighborhoods in the space-subspace hierarchy cover larger spaces but contain knowledge of a only few major pathways; lower-level neighborhoods cover smaller spaces but contain knowledge of major and minor pathways that fall in them. Thus, children neighborhoods of a parent neighborhood may contain more detailed information about a given pathway. For example, a pathway may be represented in a parent neighborhood as having four intersections (labeled a, b, c, and d in Figure 1), but, as part of a child 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 due to its smaller spatial coverage.

The representation of a neighborhood contains three types of information:

1. *Names of the sub-neighborhoods of the neighborhood.*

2. *Relative directions of sub-neighborhoods.* Like the pathways, this is based on eight octants of the compass.

3. *Pathways in the neighborhood:* Pathways are represented separately for each neighborhood in which they occur.

In order to explore some interactions between Router's model-based and case-based methods in the presence of dynamic changes in the pathways in the navigation space, 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.³

### 3.2 Model-Based Method

Router's model-based method combines means-ends analysis and limited graph-based breadth-first search. Given a navigation 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 play the role of "operators". With in a neighborhood, the pathways form a graph and are searched by a limited breadth-first search method.

**Task Structure and Control Strategy for Model-Based Method** At the highest level, Router sets up two subtasks of the path-planning task as illustrated in Figure 2:

³A 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 this 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.* Then Router parses this feedback and updates 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 plan step, and the plan steps in Router's domain have no side-effects.
This replaces Figure 2. This figure is the same as Figure 2 in


To obtain a copy of this report with the figures included, please contact the authors.

Figure 2: Router’s Task Structure

1. Neighborhood-finding. This 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 initial and goal locations, starting with the highest-level neighborhood, then its subneighborhoods, and so on.

2. Path-finding. This 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:

2a. Initial and goal locations in the same neighborhood. II and 1F in Figure 1 illustrate this situation. This
is the simplest case as it requires only a straightforward search on the pathways in the given neighborhood.

2b. Initial and goal locations in different neighborhoods on the same level. 2I and 2F in Figure 1 illustrate this situation. Using the direction information in the input, first the model-based method 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, in order to connect the initial and goal locations to the top-level path determined previously.

2c. Initial and goal locations on different levels. If the two locations are separated by only one level, the path-finding process is merely a simplification of 2b 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. 3I and 3F in Figure 1 illustrate this situation. 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 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. The procedure can backtrack and begin a new line of search if, after one or two levels of search, all of the next intersections to be queried appear on the used list. This search procedure operates uniformly regardless of the particular neighborhood in which the search is
conducted.

3.3 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”.

Representation of Cases and Organization of Case Memory A case in Router contains three kinds of information: (i) the initial and goal locations in a past planning episode, (ii) the spatial neighborhoods to which the two locations belong, and (iii) the path connecting the two locations. Each case is indexed in two ways: (a) the initial and goal locations of the stored plan, where the two locations act as the primary index, and (b) the spatial neighborhoods to which the initial and goal locations belong, where the neighborhoods of the two locations act as the secondary index to the case.

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

Task Structure and Control Strategy for Case-Based Method Again, at the highest level, Router sets up neighborhood-finding and path-finding as subtasks of the path-planning task. In addition, it uses the same procedure for the neighborhood-finding task as described in the previous section. 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 as illustrated in Figure 2.
In the case-retrieval task, 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: (i) a case that exactly matches the specification of the current problem is available in memory; (ii) a case that only partially matches the current problem is available; and (iii) 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 in the task structure of the case-based method is case validation. But the case-based method by itself cannot validate the retrieved plan. This is because while Router updates its spatial model based on the feedback on failed plans, it does not update the case memory. This in turn is because a specific pathway may potentially occur in a very large number of cases, and identifying and updating each case in which a specific pathway occurs is likely to be computationally expensive. Since Router does not update the spatial model, it can validate the retrieved plan by spatially “simulating” the plan. Thus, the case-validation subtask is skipped when Router runs purely in the case-based mode.

Next, if the case-retrieval task 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. Router 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 illustrated 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 endpoint(s) 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 additively 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.

This replaces Figure 3. This figure is the same as Figure 3 in


To obtain a copy of this report with the figures included, please contact the authors.

Figure 3: Case Adaptation

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

Case Compilation  The last subtask in the task structure of the case-based method 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, e.g., from $a$ to $c$, $a$ to $d$, $b$ to $d$ and so on, may also occur at some future time, the system also stores these “partial” paths as reusable cases. Thus, Router automatically acquires additional cases as it solves new problems.

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

3.4 Task-Directed Integration of the Model-Based and Case-Based Methods

Router’s model-based and case-based methods have method-specific control strategies. The case-based method, for example, sets up specific subtasks of a given navigation task, and specifies the order of their execution under different knowledge conditions as indicated in Figure 2 and described in the previous section. The integration of the two methods, however, raises the additional issue of strategic meta-control: what method to select and invoke, given a navigation task and a specific knowledge condition? Router’s strategic meta-control not only enables flexible and dynamic method selection, but also facilitates cooperation between the model-based and case-based methods in solving problems and in acquiring the knowledge needed for the problem solving.

**Method Selection** Router views strategic meta-control as a design task with the goal of designing a virtual problem-solving architecture for addressing a given navigation problem. It uses an “introspective” meta-reasoner for addressing this task. The meta-reasoner 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 illustrated in Figure 4. Given a specific task $T1$, 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 $M_1 \ldots M_N$; (2) the method sponsors inspect the memory and assess the applicability of the methods to $T_1$, (3) the method selector selects a specific method for $T_1$, and (4) the method selector invokes the selected method.

This replaces Figure 4. This figure is the same as Figure 4 in


To obtain a copy of this report with the figures included, please contact the authors.

Figure 4: Method Selection Architecture

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 the task of path-finding, the model-based method is useful for the tasks of 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 which at least partially matches the specification of the given task as described
in the previous section.

Given a specific task, the method sponsor associated with each method checks whether the method is applicable to the task. If the method is applicable, the method sponsor inspects the store of knowledge, and verifies if the knowledge needed by the method is available. If so, 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 its information about the tasks to which the method is applicable to decide that the case-based method is applicable to the path-finding task. Then the sponsor 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.

Method selection in Router is based on two criteria:

1. *The methods applicable to a given problem.* Given a navigation problem, the applicable methods are sponsored by the method sponsors.

2. *The computational properties of the applicable methods.* 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 [Goel et al 1993]. Thus, method selection in Router is biased towards 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 is invoked only if a case relevant to the given task is not available in memory.

**Cooperative Problem Solving** The method invoked by the meta reasoner may potentially set up several subtasks $T_{11}, T_{12}, \ldots, T_{1M}$ of the task $T_1$. Router's meta reasoner recursively applies the above mechanism for strategic meta-control to these subtasks. This results 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 illustrates, 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
since a past experience. 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, i.e., 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 a given task instance \( T1 \). If the retrieved case does not exactly match \( T1 \), 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, i.e., as finding a path that connects the ends of the path retrieved by by the case-based method to the initial and goal locations specified in \( T1 \), and is presented to the method selector. It 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, then 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-based and case-based methods thus cooperatively solve path-planning problems. In fact, the two method 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, for example.

4 Evaluation of Hypothesis 3

We now turn to the evaluation of three hypotheses we described in the introduction, beginning with the third hypothesis: For the strategy of model-based search, the principle of locality provides a productive basis for partitioning and organizing the navigation space. Most schemes for partitioning the navigation space and organizing the topological knowledge are based on the principle of locality, according to which, spatially adjacent locations are aggregated into regions of space and organized in a space-subspace hierarchy. The issue here is whether, for navigation spaces containing discrete pathways, different partitions of the
navigation space and different organizations of the space-subspace hierarchy make a significant difference to either the efficiency of problem solving or the quality of plans.

We conducted three main experiments to evaluate this hypothesis. The goal of the first experiment was to confirm the computational advantages of using Router's space-subspace hierarchy compared to a flat map. The goal of the second experiment was to determine the computational effects of small variations on Router's space-subspace hierarchy, in particular variations on the partitioning of the neighborhoods. And the goal of the third experiment was to compare the effects of different partitions of the navigation space and organizations of the space-subspace hierarchy.

All three experiments were conducted on a randomly generated sample of 195 problems in the GT domain. The second and the third experiments used Meta-Router working in conjunction with Router, as we mentioned earlier and describe below. In conducting the three experiments, we had to decide how to measure the efficiency of problem solving and the quality of navigation plans. For the domain of navigation planning, the logical answer to the second question is that a shorter navigation plan is better than a longer one. However, since Router contains no metrical knowledge, the question 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. Similarly, we used a qualitative measure to gauge the cost of problem solving, namely, the number of different street intersections the model-based method expanded to generate a path for connecting the initial and goal locations.

4.1 Experiment 3.1

To investigate the computational effects of Router's space-subspace neighborhood hierarchy, we designed a simple control condition in the form of an experimental system in which plans are generated by breadth first search on a flat map of the navigation space. We then compared Router's model-based method with the control condition, where Router's method too uses breadth-first search though the search is limited by a horizon of 3 and occurs only locally, within specific neighborhoods. The performance was compared on the same set of 195 randomly generated problems.
<table>
<thead>
<tr>
<th></th>
<th>Average Path Length</th>
<th>Average Processing Cost</th>
<th>Range of Path Length</th>
<th>Range of Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Router's Model Flat Map</td>
<td>12.34</td>
<td>6.86</td>
<td>2-24</td>
<td>1-17</td>
</tr>
<tr>
<td>Flat Map</td>
<td>9.15</td>
<td>13.72</td>
<td>2-22</td>
<td>1-82</td>
</tr>
</tbody>
</table>

Figure 5: Average number of path segments in the generated plans, average number of intersections visited during plan generation, range of path segments in the generated plans, and range of intersections expanded during plan generation in Experiment 3.1

Figure 5 summarizes the comparison between Router’s model-based method and the control condition. As we expected, Router’s problem solving is considerably more efficient than in the control condition. On average, Router expanded 6.86 intersections while the control system expanded 13.72 intersections for the same set of problems. Also, the range of variation in problem-solving cost in Router is much smaller than in the control condition. This means that for problems in which the initial and goal locations are distant, the use of the flat map is much more expensive.

In addition, we found that plans produced in the control condition were slightly more parsimonious plans than those generated by Router’s model-based method. As indicated in Figure 5, the average number of path segments in the plans produced in the control condition was 9.15 while for Router this number was 12.34.

4.2 Meta-Router: reorganizing topological models

To investigate the computational effects of variations in Router’s space-subspace hierarchy, we conducted a second experiment of a very different kind. In this experiment, we were especially interested in the issue of the partitioning of the navigation space into neighborhoods. To test the effects of the partitioning of neighborhoods, we needed the capability of reorganizing Router’s spatial model. Therefore, we designed and developed a system called Meta-Router that is capable of reflection on Router’s problem solving and reorganizing its space-subspace hierarchy. As Router solves a given navigation problem, Meta-Router monitors and records the problem solving. After Router completes its problem solving, the (human) experimenter inspects the generated plan. The experimenter may now provide a preferred plan as feedback, where the preferred plan is more parsimonious than the generated plan (but may or may not be optimal).
Meta-Router now autonomously reflects on the trace of Router's problem solving, assigns blame for the changes between the generated and the preferred paths, and may heuristically modify the neighborhood partitioning by expanding specific neighborhoods to include more intersections, or by shrinking them to exclude some intersections.\footnote{The learning method used by Meta-Router is not important for the present discussion because the learning of topological models is not the research issue here. Instead, the research issue pertains to the computational effects of variations in Router's topological models. Therefore, we limit the following description to only what Meta-Router does, not how it does it. The reader may consult [Stroulia and Goel 1995] for more information on the what and how of Meta-Router, including its methods for assigning blame and reorganizing knowledge.}

If Router classifies the initial and goal locations as belonging to different neighborhoods, and the path it generates is longer by three intersections or more than the preferred path, then Meta-Router decides that the initial and goal locations should have been classified as belonging to the same neighborhood. It then identifies the neighborhood in which the path should have been generated by identifying the neighborhood which contains the majority of the intersections in the preferred plan. Finally, it reorganizes Router's spatial model in order to make the problem solvable in that neighborhood, by expanding the neighborhood to include all the intersections of the preferred path (including the initial and goal locations).

If Router classifies the initial and goal locations as belonging to the same neighborhoods $N_i$, and the path it produces is longer than the preferred path, Meta-Router inspects its neighborhoods to find an alternative neighborhood $N_j$ which contains all the intersections of the preferred path. If there is such a neighborhood in Router's topological model, then Meta-Router reorganizes the model to make the problem unsolvable in $N_i$, by eliminating either the initial or the goal intersection from $N_i$. Thus, in the revised spatial model, $N_i$ would not contain both locations, and thus the more appropriate neighborhood $N_j$ would be chosen.

Note that Meta-Router's reorganization of Router's topological model abides by the principle of locality, in particular with the two design heuristics that operationalize this principle for navigation spaces containing discrete pathways as described in Section 3.

### 4.3 Experiment 3.2

Starting with the Router's topological model described in section 3, we trained Router and Meta-Router on a sequence of fifty randomly generated problems. After Router had solved the randomly generated sequence
of fifty training problems and assimilated the feedback on them, we found that Router’s topological model indeed was slightly different from the original model in that the boundaries among its neighborhoods were different. Adjacent neighborhoods that used to be completely non-overlapping began to overlap, and neighborhoods that used to overlap with each other over a certain area now overlapped over different areas.

<table>
<thead>
<tr>
<th></th>
<th>Average Path Length</th>
<th>Average Processing Cost</th>
<th>Range of Path Length</th>
<th>Range of Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Router</td>
<td>12.34</td>
<td>6.86</td>
<td>2-24</td>
<td>1-17</td>
</tr>
<tr>
<td>Modified Router</td>
<td>11.91</td>
<td>6.96</td>
<td>2-24</td>
<td>1-18</td>
</tr>
</tbody>
</table>

Figure 6: Average number of path segments in the generated plan, average number of intersections visited during plan generation, the range of number of path segments in the generated plans, and the range of number of intersections expanded during plan generation in Experiment 3.2

Then we ran the modified Router over the same set of 195 randomly generated problems as in experiment 3.1. Figure 6 summarizes the comparison between the original Router and the modified Router. Clearly, there is no significant difference in the performance of the two systems. Neither the average number of path segments nor the range of this number over 195 problems changed significantly with variations in Router’s space-subspace hierarchy. Similarly, neither the average number of intersections expanded during plan generation nor the range of this number varied significantly.

4.4 Meta-Router: learning topological models

Experiment 3.2 indicates that Router’s partitioning of neighborhoods and organization of the neighborhoods in a hierarchy is quite robust. But this still leaves open the issue whether “any” partitioning and organization of the navigation space based on the principle of locality would be equally productive. To test this hypothesis, we conducted a third experiment. In this experiment, Router starts with a flat map of the navigation space (as in the control condition of Experiment 3.1). But Meta-Router enables Router to incrementally acquire a space-subspace hierarchy based on its problem-solving experiences, where the learned hierarchy is based on the principle of locality. To achieve this, we enhanced Meta-Router’s learning capability as follows:

If Router classifies the initial and goal locations as belonging to the same neighborhood $N_i$, and the path
it produces is longer than the preferred path by five intersections or more, Meta-Router partitions $N_i$ into two neighborhoods $N_j$ and $N_k$, removes some information from $N_i$ and makes it a parent neighborhood of $N_j$ and $N_k$.

To decide on the contents on $N_j$ and $N_k$, Meta-Router draws a straight line perpendicular to the middle segment of the preferred path. Then, it includes the pathways in $N_i$ that fall on one side of this line in $N_j$, and similarly includes the pathways on the other side of the line in $N_k$. Pathways crossing the line are included in both $N_j$ and $N_k$. However, in each neighborhood these pathways are represented as containing only the intersections with other pathways in the same neighborhood. Both $N_j$ and $N_k$ are added in the neighborhood hierarchy as children of $N_i$.

Pathways from $N_i$ are deleted as follows. First, the pathways are sorted in ascending order according to the number of their intersections. Then, up to 30% of the pathways with the least number of intersections are removed from $N_i$. After the deletion of each pathway, the resulting neighborhood is tested for internal connectivity and availability of cross-indices with $N_j$ and $N_k$. Internal connectivity is tested by a depth-first traversal of the intersections contained in a neighborhood. Availability of cross-indices with the children neighborhoods is decided by finding the set intersections. If the deletion of a specific pathway from $N_i$ results in loss of either internal connectivity or cross-indices with $N_j$ and $N_k$, then the deletion is retracted and the deletion process is terminated.

*If Router classifies the initial and goal locations as belonging to different neighborhoods*, and the path it produces is longer than the preferred path by five intersections or more, then:

*If the initial location is adjacent to an intersection in the goal neighborhood*, the goal neighborhood is expanded to include the initial location.

*Similarly, if the goal location is adjacent to an intersection in the initial neighborhood*, the initial neighborhood is expanded to include the goal location.

*If the distance between the initial and goal locations is less than five intersections*, then a new neighborhood, overlapping with the initial and goal neighborhoods, is created. The new neighborhood contains
the pathways which were mentioned in the preferred path and the intersections between them. The new neighborhood is added to the neighborhood hierarchy as a child of the lowest-level super-neighborhood of the initial and goal neighborhoods.

Again, note that Meta-Router’s reorganization of Router’s topological model abides by the operationalization of the principle of locality in the two heuristics described in Section 3.

### 4.5 Experiment 3.3

Starting Router with a flat map, we trained Router and Meta-Router on three sequences of thirty randomly generated problems each. This led to three topological models, each based on the principle of locality. The three space-subspace hierarchies consisted of seventeen (17), twenty-three (23), and nineteen (19) neighborhoods organized in four (4), five (5) and four (4) levels, respectively.

<table>
<thead>
<tr>
<th></th>
<th>Average Path Length</th>
<th>Average Process Cost</th>
<th># of Unsolved Problems</th>
<th>% of Unsolved Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat Map</td>
<td>7.22</td>
<td>41.52</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>Hierarchical Model 1</td>
<td>10.90</td>
<td>23.61</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>Hierarchical Model 2</td>
<td>12.15</td>
<td>23.90</td>
<td>18</td>
<td>0.09</td>
</tr>
<tr>
<td>Hierarchical Model 3</td>
<td>11.02</td>
<td>25.63</td>
<td>11</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Figure 7: Average number of path segments in generated plans, average number of intersections expanded during plan generation, number of unsolved problems, and percentage of unsolved problems in Experiment 3.3

Then we ran Router’s model-based method on each of the three topological models described above. Each of these three experiments was conducted on the same sequence of 195 randomly generated problems used in Experiment 3.1. Figure 7 summarizes the average number of path segments in the generated plans, and the average number of intersections expanded during plan generation for each of the three space-subspace hierarchies. The figure also shows the numbers and percentages of unsolved problems with each of the three hierarchies. As figure 7 indicates, the problem-solving cost with all three space-subspace hierarchies decreased significantly compared to search using a flat map. In addition, the quality of plans also decreased a little in all three cases. Note that in the case of two of these knowledge organizations, the model-based search failed to generate any solution for some of the 195 navigation problems.
5 Evaluation of Hypothesis 2

In this section we describe our evaluation of Hypothesis 2: For deliberative planning, the case-based strategy of reusing past navigation plans results in more efficient problem solving than the model-based strategy of searching a model of the navigation space, without any significant loss in the quality of plans or the problem-solving coverage. The issue here is whether the case-based strategy generates navigation plans more efficiently than the strategy of model-based search, and whether the former strategy also leads to navigation plans of equal quality as the latter. A related issue is whether the case-based strategy can be bootstrapped with a small number of cases relative to the number of all possible problems in the navigation space.

The design of Router enables us to explore another issue concerning the task-directed integration of the model-based and case-based strategies for navigation planning. In order to investigate all these issues, we conducted four main experiments. The goal of the first experiment was to test two related hypotheses regarding the computational efficiency of Router's model-based method, case-based method, and multistrategy reasoning: H[2i(a)] Router's case-based method is computationally more efficient than its model-based method, and, hence, H[2i(b)] 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 processing cost, in this and the other experiments in this set, was measured in real computing time. All experiments were conducted on a dedicated Sun workstation.

The goal of the second set experiment was to test two related hypotheses regarding the quality of solutions produced by Router's model-based, case-based, and multistrategy reasoning: H[2ii(a)] the case-based method produces solutions of quality equal to those produced by the model-based method, and, hence, H[2ii(b)] the integrated method produces solutions of quality at least equal to those produced by the model-based method alone. The quality of plans was measured in the same way as in the set of experiments pertaining to Hypothesis 3: the number of path segments in a navigation plan, where a path segment is defined as the path between two consecutive street changes in the overall path.

The goal of the third experiment was to test a hypothesis about the decomposition of cases into partial
cases: $H[2\text{iii}]$ 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 (see [Kolodner 1993]). Testing this hypothesis raises the issue of what is a reasonable partial case. In the domain of navigational path planning, the logical answer is that partial cases correspond to the path segments in a navigation plan (where, again, a path segment is defined as the path between two consecutive street changes).

The goal of the fourth experiment was to test two related hypotheses about the problem-solving coverage and knowledge requirements of Router’s case-based method: $H[2\text{iv}(a)]$ the case-based method can be bootstrapped with relatively few cases in memory, and $H[2\text{iv}(b)]$ the case-based method has the same problem-solving coverage as the model-based method even though its knowledge requirements are smaller.

We tested the hypotheses described above by conducting a series of ablation experiments [Cohen and Howe 1988]. In these experiments, first, a specific portion of the program is made inoperational and the performance of the remaining portions is observed on some set of problems. Then, the first portion of the program is made operational but a different portion of the program is made inoperational, and the performance the program is observed on the same set of problems. Finally, the two observed performances are compared and related to the different portions of the program.

In all experiments involving the exclusive use of the case-based method (in which the model-based methods and the task-directed integration mechanism were inoperational), we primed the case memory by adding (randomly generated) cases to the memory before conducting the experiments. In all experiments involving multistrategy reasoning (in which all portions of the program were operational), the case memory was empty at the beginning of the experiment, but it grew as subsequent problem-solving cases were stored in it.

The domains of the GT campus and the CoC building admit about 10000 and 1000 problems, respectively. We conducted our experiments with Router using 10 sets of 50 problems each from the GT
domain, the CoC domain, and the combined GT-CoC domain. In the combined domain, the initial or the goal locations could be either an intersection of streets on the GT campus or a specific office in the CoC building. 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.

5.1 Experiment 2.1

Figure 8 illustrates the results of the first experiment. The data confirms hypothesis H[2i(a)]: it shows that 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 system’s model-based method or integrated methods.

![Comparison of average problem solving times for different strategies.](image)

In addition, this data shows that hypothesis H[2i(b)] is false: when the number of cases in memory is small (not shown in the figure), Router’s model-based method performs faster than its combined method on most (approximately 92%) of the problems.
5.2 Experiment 2.2

The data from experiment 2 showed that both hypotheses H[2ii(a)] and H[2ii(b)] 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.

5.3 Experiment 2.3

Figure 9 illustrates the results of experiment 3. The data shows that hypothesis H[2iii] is false: the decomposition of cases into partial cases and the storage of partial cases increased the problem-solving time instead of reducing it. On average, the problem-solving time for the entire path-planning process, including the storage of partial paths, was 1.7 times more than the problem-solving time without storage of partial paths.  

Figure 9: Effects of partial cases.

5Because of this result, all other experiments were conducted without storing partial cases.
5.4 Experiment 2.4

Figure 10 illustrates the results of the fourth experiment. The data shows that while hypothesis H[2iv(a)] is true, hypothesis H[2iv(b)] 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 approximately linearly with the initial number of cases in memory. In particular, 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 (50%) of the problems given to it.

![Graph showing the number of problems solved](image)

Figure 10: Number of problems solved (out of 50) with different amounts of memory bootstrapping for the pure case-based method.

6 Evaluation of Hypothesis 1

Finally, in this section we describe our evaluation of Hypothesis 1: For hybrid robots capable of both deliberative planning and situated action, qualitative topological representations are sufficient for effective navigation of visually engineered worlds containing discrete pathways. As discussed in Section 2, the evaluation of this hypothesis requires an integration with Router with a mechanism for situated action and their joint embodiment in a physical robot. Therefore, we integrated Router in a hybrid robot architecture and embodied the architecture in a robot called Stimpy.
6.1 Overview of AuRA

The Autonomous Robot Architecture (AuRA), developed by [Arkin 1989a], is a general, hybrid robot architecture for integrating navigation planning and reactive control. AuRA produces plans at three levels of abstraction: mission, navigation, and pilot. The mission planner may take a set of goals as input (e.g., make a copy of this paper and collect my mail), and give a mission-plan for achieving the goals as output (e.g., go to the copier room, make a copy of the paper, go the mail room, collect my mail, etc.). At the navigation level, a path planner may take a step in the mission plan as input. This input may be in the form of an initial location and goal location in the navigation space, (e.g., go from my office to the copier room). The path planner may give a piecewise linear path-plan from the initial to the goal location as its output that avoids all known static obstacles (e.g., go from my office into the corridor, go right until the end of the corridor, go through the door, etc.). At the pilot level, a motion planner may take a path segment in the planned path as input (e.g., go from my office into the corridor), and may give a sequence of motor actions for accomplishing the goal as output (e.g., move left, move ahead, etc.) such that one is executed before the next is output. The motion planner directs the robot to move to the goal location of the specific path segment using reactive motor schemas. Figure 11 illustrates the AuRA architecture.

6.2 Situated Action in AuRA

The pilot level of AuRA uses schema-based reactive control to realize situated action [Arkin 1989b]. Figure 12 illustrates the reactive control mechanism. Each schema in the mechanism for reactive control is responsible for one basic “reflex” action, such as avoiding obstacles. Each schema outputs a vector, or a set of vectors, in response to a particular perceptual stimulus. The direction and magnitude of the vector are determined by the nature and gain of the schema. The set of vector outputs from all the schemas are summed and normalized. The normalized vector is given to the robot for execution. Thus the various schemas collectively determine the robot's reaction to the current environment.

In our integration of Router with AuRA, which we call Raura (for Router in AuRA), we used two schemas that were already developed for use in the reactive control mechanism of AuRA: move-ahead and avoid-obstacle. The move-ahead schema takes no input and always produces a vector in one particular
direction. The output vector of the move-ahead schema has a magnitude equal to the gain for the schema. The avoid-obstacle schema takes input from the ultrasonic sensors indicating the location of nearby objects in the environment. It produces a set of vectors, one for each obstacle. Each obstacle produces a force on the robot in a direction away from the obstacle and with a magnitude reflecting both the distance from the obstacle and the gain for the avoid-obstacle schema.

6.3 Raura: The integration of Router in AuRA

The mission planner for the AuRA architecture is yet to be standardized, although mission planners have been developed for particular tasks. In our experiment, the (human) operator plays the role of the mission planner. The current navigation planner in AuRA contains a global, flat, topological map with both qualitative and metrical knowledge. Given a navigation task, it uses the A* algorithm on the map for generating navigation plans.
Router fits nicely into AuRA as a substitute for the current navigation planner. Thus, for the purposes of our experimental study, we replaced AuRA’s old navigation planner by the Router system which uses only qualitative knowledge as described earlier.

However, the form of the output of Router is not the same as the form of the input to the pilot level in AuRA. Router outputs a path consisting of path segments from some initial location to a goal location. Each path segment contains a direction of travel, a street name, a start intersection for the street, and a goal intersection for the street. For instance, one segment of a Router plan might look something like, "Go E on 4th from Atlantic to Techwood." But the pilot level in AuRA needs input in the form of an instantiation of various motor schemas with appropriate parameters.

In Raura, Router produces a plan in its entirety before the execution of the plan begins. Then, for each segment of the plan, Router calls an interpreter. The job of the interpreter is to take the path segment from Router, instantiate the necessary schemas in the pilot in AuRA with appropriate parameters, and then return control to Router when a perceptual schema signals that the robot has completed executing the path segment. Router suspends its problem solving during the call to the interpreter.

The interpreter instantiates an avoid-obstacle schema, a move-ahead schema, and a special perceptual schema that is responsible for detecting when the robot has arrived at the goal of the current path segment. The direction of travel in the path segment is converted to radians and put into the direction
parameter of the move-ahead schema. The goal location of the current path segment is used as a probe into a data base of locations (intersections) that associates a unique numerical code with the location. This code is given to the perceptual schema described below.

The perceptual schema detects when the robot has arrived at the goal location of the current path segment. The navigation space is visually engineered so that each intersection between hallways on the office floor is marked with a unique barcode. The visual barcodes correspond to the codes in the data-base of intersections. The interpreter in Raura gives the perceptual schema the code for the goal location of the current path segment. The perceptual schema continuously monitors the world to see if it can find a barcode that corresponds to the code of the goal location and if the robot is directly in front of this barcode. When the robot arrives at the goal location of the current path segment denoted by the barcode, the perceptual schema sends a signal to the interpreter. In this way, the interpreter knows that the robot has completed the current path segment, and it transfers the control back to Router. Router continues to call the interpreter for each path segment until the path has been completed.

6.4 Stimpy: The embodiment of Raura on a mobile robot

We tested Raura on a Denning MRV-2 robot that we call Stimpy. As illustrated in Figure 13, Stimpy is a three-wheeled holonomic vehicle with shaft encoders, a LaserNav rotating barcode reader, and a ring of 24 ultrasonic sensors for detecting obstacles. As we mentioned earlier, the shaft encoders provide information about the orientation of the robot and the distance traversed from the initial location, the ultrasound sensors enable the detection of static and moving obstacles, and the LaserNav barcode reader allows the detection of distinctive places or landmarks in the navigation space marked with visual barcodes.

In our experiments, Raura ran on an off-board computer, with two-way communication between Stimpy and the computer through radio waves, as illustrated in Figure 14. This was because we did not have the hardware capability of running Raura on board Stimpy. But this limited the range of experiments we could conduct with Stimpy. Radio communication between Stimpy and the off-board computer running Raura had a strong tendency to break down as Stimpy wandered away from the computer, and especially as it took turns around offices, laboratories and hallways. Thus, the experiments had to be limited to situations
This replaces Figure 13.

To obtain a copy of this report with the figures included, please contact the authors.

Figure 13: Stimpy.

in which the initial and goal locations were relatively close and navigation between them did not require Stimpy to take lot of turns.

6.5 Experiments with Stimpy

We tested Raura on Stimpy in the Manufacturing Research Center (MaRC) building located on the Georgia Tech campus. This is an office building environment consisting mainly of laboratories, offices and corridors. The various walls, doors, pillars and other objects provide a variety of static obstacles in this environment. In addition, people walking to and fro during some of our experiments provided moving obstacles.

We conducted three sets of experiments with Raura. The first two sets consisted of four trials each and were conducted in the presence of only static obstacles. The third set of experiments consisted of twelve trials and allowed for moving obstacles. Each trial in each of the three sets of experiments required Raura
Figure 14: Raura running on an off-board computer and communicating with Stimpy through radio waves.

to go from an office in the MaRC building to either the hallway or another office on the same floor of the
the same building.

The four trials in the first set of experiments were all on the same navigation problem illustrated in
Figure 15: Stimpy had to go from a laboratory to a specific hallway intersection outside the laboratory.
In the first trial in this set, Raura’s (or Router’s) case memory was empty. But as Raura solved navigation
problems, generated plans (using the model-based method), and executed them, it compiled the generated
plans into a case and stored it in its case memory. In the subsequent three trials, Raura had access to the
case compiled in the first trial. This enabled us to test Router’s model-based method in the first trial and
its ability to retrieve cases in the other three. We found that Stimpy successfully solved the navigation
problem and reached the goal location on each of the four trials.

The four trials in the second set of experiments were also all on the same navigation problem illustrated
in Figure 16: again, Stimpy had to go from a laboratory to a specific hallway intersection outside the
laboratory. However, in this set of experiments, the goal intersection was in the same direction as that in
the first set of experiments but farther away from the initial location. Thus, the pathway generated in the
second set of experiments was a superset of the pathway in the first set. In addition, in the second set
of experiments, Raura had access to the case generated in the first. Thus, to solve the second navigation
problem, it reused the case generated in the first, and adapted it using the model based method (because
it did not have any other case in its memory). This enabled us to test case adaptation and task-directed integration of the model-based and case-based methods in Router. We found that Stimpy successfully solved the navigation problem and reached the goal location on three of the four trials but failed on one because of a loss of radio communication between the off-board computer and the robot.

The twelve trials in the third set of experiments were all on the same navigation problem but this time Stimpy had to go from one laboratory to another as illustrated in Figure 17. In this set of experiments,
we had quite a few problems with the radio communication breaking down between Stimpy and the off-board computer during some of the trials. The radio communication typically broke down because the third navigation task required Stimpy to travel further from the controlling off-board computer than in the first two sets of experiments. This resulted in abortions of the trials. Even so, Raura succeeded in guiding Stimpy on eight of the twelve trials. Note that these trials allowed for moving obstacles in the form of people moving to and fro. We found that as long as the obstacles moves slowly compared to the processing-cycle and the motor-actuation times of the robot, Stimpy could react to the moving obstacle, move away from it, and then resume its goal-directed, plan-guided navigation.

![Diagram of moving obstacle and barcodes](image.png)

Figure 17: Navigation problem of the third set of experiments.

7 Discussion

In this section we discuss the design, hypotheses and results of our experiments, relate them to other work, and conclude this paper.

7.1 Design of Router

The goal of this study was to evaluate specific hypotheses pertaining to multistrategy spatial navigation, not the design of an optimal path planner. Thus, in designing Router, we opted for simplicity rather
than optimality. This simplicity is apparent in the design of Router’s model-based method which uses only qualitative topological knowledge in the form of a space-subspace hierarchy, and generates navigation plans by combining means-ends analysis with limited, local, breadth-first search.

Kuipers’ [1978] TOUR program probably was the first AI program to use a qualitative topological representation of navigation spaces characterized by discrete pathways. TOUR simulated the exploration of a navigation space containing routes, assimilated the information into a “cognitive map”, and inferred incidence relations, i.e., intersections between pathways. While it had the notion of spatial regions or neighborhoods, it did not infer the partitioning of the navigation space into neighborhoods, nor did it organize the neighborhoods in a space-subspace hierarchy. Router assumes this kind of partitioning and organization and then uses it to generate navigation plans.

Of course, Router’s model-based method can be made more optimal. For example, if metrical knowledge in the form of distances of path segments is available, and if the goal is to generate more optimal paths, then it could use Dijsktra’s [1959] algorithm for searching the neighborhoods instead of breadth-first search. In fact, this is precisely what Liu et al.’s [1994] RFinder does: it uses a Router-like space-subspace hierarchy and then runs Dijsktra’s algorithm in the neighborhoods.

Two issues are critical in any instantiation of the case-based strategy [Kolodner 1993]: case retrieval and case adaptation. Addressing the issue of case retrieval requires a scheme for indexing and organizing the case memory so that case retrieval is efficient and effective. Router uses model-based case indexing and organization. Its use of topological primitives for indexing cases is similar to Alterman’s [1988] PLEXUS system, which uses similar primitives for indexing cases of mission planning. In addition, in Router the navigation-planning cases are indexed by the spatial neighborhoods and the case memory is organized around the space-subspace hierarchy.

The data from Experiment 2.3 described in Section 5 indicates that this model-based scheme for case indexing and organization works well. In this experiment, Router stored in its case memory not only the navigation-planning cases but also all partial cases. We found that the process of case retrieval worked efficiently, despite the large and rapid increase in the number of cases stored in memory. In particular,
we found that the cost of retrieving an appropriate case was a small fraction of the total cost even in the presence of partial cases in memory.

Router uses two methods for adapting past navigation plans. The first method adapts the current case by recursively spawning adaptation goals, retrieving relevant cases, and combining them with the current case. The Mediator system [Kolodner and Simpson 1989] probably was the first AI program to use this adaptation strategy, though in the different domain of everyday decision making. [Anderson, Kushmerick, and Lebiere 1993] suggest that humans often adapt past navigation plans for new problems simply by concatenating them. Router provides a computational model of how relevant navigation plans can be recursively retrieved from memory and combined to solve new navigation problems.

The second method for case adaptation in Router is model-based. This method uses the topological model for completing the retrieved plan. This method builds on our earlier work on integrating case-based and model-based reasoning in the context of engineering design. In that work, we found that structure-behavior-function models of physical devices enable the adaptation of design cases [Goel 1991; Goel and Chandrasekaran 1992].

Router’s mechanism for method-directed spawning of subtasks and task-directed method selection and integration directly evolves from Chandrasekaran’s [1989] conceptual framework of task structures. In this framework, methods directly specify the subtasks they set up. With in this framework, Punch [1989] developed an architecture for task-integrated problem solving (or TIPS) that addresses the issue of method selection and integration. TIPS uses the sponsor-selector control structure originally developed by Brown and Chandrasekaran [1989] for the task of selecting design plans in the context of routine design problems.

Router adopts and adapts the TIPS architecture for multistrategy path planning. In general, the selection of a specific method may depend on three criteria [Goel and Chandrasekaran 1992]: (1) Since different methods may yield solutions with different properties (e.g., optimality, correctness), what are the properties desired of the solution to the task T1; (2) Since different methods may have different computational requirements (e.g., processing time, memory size), what are the computational constraints on T1 and what are the computational resources available to the problem solver; and (3) Since different methods use dif-
different types of knowledge (e.g., heuristics, associations, plans, cases, models), what knowledge is available to the problem solver? 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 irrespective of other factors. In principle, the method sponsors are responsible for evaluation criterion (3), and the method selector is responsible for evaluation criteria (1) and (2). In the current implementation of Router, if both the model-based and case-based methods for a task are sponsored, then the method selection depends solely on (2), and is biased towards the case-based method because early experiments showed that this method is computationally more efficient.

7.2 Hypothesis 1

Our Hypothesis 1 pertains to the knowledge “boundary” between navigation planning and situation action in navigation spaces characterized by discrete pathways. It also concerns two different assumptions regarding navigation planning. Almost all AI work related to navigation planning uses qualitative knowledge of the world (e.g., [Alterman 1988; Fikes and Nilsson 1971; Fikes, Hart and Nilsson 1972; McDermott and Davis 1984]). This line of work is based on the assumption that qualitative knowledge is sufficient for planning in most navigation domains. Our work on Router and related theories shares this assumption. In contrast, almost all robotics work on navigation planning uses metrical knowledge in addition to qualitative knowledge (e.g., [Arkin 1989a; Latombe 1991]). This line of work argues that in general qualitative reasoning is not sufficient for navigation planning in the context of physical robots operating in the real world. This is because, the argument goes, the movement of a robot needs to be computed with an accuracy and precision that is beyond qualitative knowledge and reasoning.

Recent work on robotics (e.g., [Brooks]) shifts the focus and emphasis from navigation planning to situated action, i.e., to reactive mechanisms that directly map perceptions of the world to actions on it. The mechanisms for reactive control are not based on explicit representations of world knowledge, qualitative or quantitative. This line of work apparently is based on two main computational arguments. First, navigation planning in general is computationally too complex for modeling “real time” behavior. Second, the quality of navigation plans depends on the quality of the knowledge of the world, which often is incomplete and uncertain. Situated action, the argument goes, avoids both problems. In particular, it
avoids the second problem because it uses the information that the world affords, where the world is its own best representation. Note that theories of situated action move away not only from AI theories but also from traditional robotics theories of navigation planning.

This in turn has led to proposals for hybrid robot architectures that combine navigation planning with situation action (e.g., [Arkin 1989a, 1989b]). The utility of hybrid architectures is apparent in our experiments with Stimpy. Suppose that Stimpy was a purely situated actor, with no capability of planning. Then, in Experiments 1.1, 1.2 and 1.3, once it moved out of the initial laboratory into the hallway, it’s decision on the direction in which to move next would be arbitrary because the navigation space affords no clue about the direction of the goal relative to the current location. If Stimpy made the “wrong” choice, i.e., it turned away from the direction of the goal, it still might be able to eventually reach the goal location, but only after exploring a (potentially much) larger portion of the navigation space. On the other hand, suppose that Stimpy formed navigation plans but had no capability of situated action. Then, in Experiment 1.3, it would collide with moving obstacles because knowledge of moving obstacles, unlike that of static obstacles, is necessarily incomplete, e.g., it is hard to predict what obstacle may come in the way, where and when.

But note that hybrid robot architectures, such as AuRA, still rely on traditional robotics methods for navigation planning, i.e., they use both qualitative and quantitative knowledge. Our experiments with Stimpy indicate that quantitative knowledge may not be needed for all navigation domains. In particular, they suggest that, for navigation spaces characterized by discrete pathways and visual landmarks, qualitative knowledge may suffice. Router uses only qualitative knowledge and Raura combines Router with reactive control. Router’s qualitative plans guide the “macro behavior” of Stimpy and Raura’s reactive control governs its “micro behavior”. Thus, in Experiments 1.1, 1.2, and 1.3, Router’s qualitative plans guide the decision about what direction to move in after Stimpy exits the initial laboratory and enters the hallway. But in Experiment 1.3, Raura’s reactive control prevents Stimpy from colliding with obstacles not modeled in Router such as moving obstacles. Even in avoiding moving obstacles, the qualitative plan guides Stimpy’s actions through the move-ahead schema which continues to exert influence in the direction indicated by the plan.
Our experiments also point to two limitations of Router and Raura. Router generates qualitative plans for navigating spaces characterized by discrete pathways, and represents its topological knowledge as a graph consisting of pathways and intersections among them. This works well for the GT domain, in which navigation is limited to long and narrow roads and streets. Similarly, in the CoC and MaRC domains, the hallways and corridors are long and narrow and thus can be appropriately modeled as discrete pathways. However, navigation in the laboratories in CoC and MaRC buildings is not really limited to discrete pathways, and thus the laboratories are not represented well in Router's topological model. Since Router can only form plans from and to intersections on pathways, we had represent offices and laboratories by “virtual” pathways. But this restricted Stimpy to those locations in a laboratory that actually were specified in the “virtual” pathways. Thus, from an engineering perspective, Router's representation of continuous navigation spaces such as laboratories is inappropriate.

The navigation space in our experiments is engineered so that each intersection is characterized by a unique barcode readable by Stimpy. Raura monitors the execution of a path segment in a navigation plan by reading the barcodes along the current pathway. This mechanism for execution monitoring enables Raura to determine when it has succeeded in completing a path segment. But it has no way of recognizing plan failure due to some incompleteness of Router's topological model. For example, if the current pathway is blocked, then Raura will not realize this. In this case, Stimpy may enter a behavioral pattern in which the reactive move-ahead schema attempts to move Stimpy in the direction of the goal intersection but the avoid-obstacle schema attempts to move it away from the object that is blocking the pathway.

This problem has two causes: the inherent incompleteness of Router's topological model in the presence of dynamic changes in the navigation space, and the monitoring of plan execution solely at the reactive level in Raura. Note that if Router knows that given pathway is blocked, it would not generate a navigation plan containing the pathway. But if the navigation space is dynamically changing and Router's model is incomplete, then the plan it generates may fail upon execution. But Raura's reactive mechanism for monitoring plan execution does not recognize plan failures. The recognition of plan failures appears to require additional (deliberative and qualitative) reasoning. We are presently exploring this issue in the Autognostic project [Stroulia 1995].
7.3 Hypothesis 2

One of the arguments for theories of situated action is that deliberative planning is computationally too complex to model behavior close to “real time”. Most theories of planning in both AI and robotics use some kind of model-based search to generate plans from start. Recent theories of case-based plan reuse offer a more efficient planning strategy. This is because the case-based method adapts old plans rather than generating new ones from the start, and an old plan that closely matches a new problem rapidly provides a solution in the “neighborhood” of the desired plan. Thus, we hypothesized that, for navigation spaces characterized by discrete pathways, the case-based method will generate navigation plans more efficiently than the model-based method. Experiment 2.1 confirms this hypothesis.

The quality of plans produced is another important dimension for analyzing a planning strategy. We hypothesized that, for navigation spaces characterized by discrete pathways, the case-based method will generate navigation plans of a quality equal to that of the model-based method. But, in Experiment 2.2, we found that this is not always true: for the same set of randomly generated problems, Router’s model-based method always produced plans of quality as good as or better than those produced by the case-based method. In principle, the lower quality of plans produced by the case-based method can be due to two reasons. First, the scheme for indexing cases and the method of accessing them may not be effective, i.e., it may not result in the retrieval of closely matching cases. Second, the method for adapting a retrieved case may not be effective, i.e., it may not result in the generation of high-quality plans. Router’s model-based scheme for indexing cases and organizing the case memory, and its method for accessing cases from memory appear to work well, as became evident in Experiment 2.3. The main cause for the lower quality of plans generated by Router’s case-based method apparently lies in its method for adapting a case by combining multiple cases. This method results in concatenating old plans. As a result, the generated navigation plan may contain zig-zags and loops.

We already have given the reasons for Router’s use of this method for adapting navigation plans: case combination is a well known and powerful adaptation strategy [Kolodner and Simpson 1989], and psychological data [Anderson, Kushmerick, and Lebiere 1993] appears to indicate that humans often use concatenation of navigation plans as a primary plan adaptation strategy. Of course, it is possible to
modify Router's case-adaptation strategy so that it produces better quality plans. For example, once a candidate plan is generated, it can be processed further to remove (or reduce) loops and zig-zags in it. Alternatively, the plan retrieved from memory may be examined carefully and only a fragment of the plan may be combined with plan fragments from other cases. But this naturally will add to the computational cost of the case-based method. Thus, there is a fundamental tension between the efficiency of planning and the quality of plans produced by Router's case-based method. Perhaps this tension is obvious but our experimental results make it explicit.

It is often argued (e.g., [Kolodner 1993]) that decomposing a case into partial cases and storing the partial cases in memory may aid the reasoner in retrieving cases more relevant to a given problem. This in turn, the argument continues, would further enhance the efficiency of case-based reasoning. But the results of our Experiment 2.3 indicate this is not always true. On the contrary, we found that the use of partial cases significantly adds (about 70%) to the cost of Router's case-based method.

A different aspect of the results of Experiment 2.3 is more surprising. Router's case-based method in Experiment 2.3 not only retrieves and adapts cases but also decomposes cases into partial cases and stores them in memory. We expected that much of the added cost of using partial cases would be in the retrieval step. We thought that the use of partial cases would add to the cost of retrieving appropriate cases from memory since the memory would contain many more cases. Our analysis of the data, however, shows that the added cost of retrieval is only a small fraction of the added cost of Router's case-based method as a whole. This indicates that the model-based scheme for indexing cases and organizing the case memory works well even in the presence of thousands of navigation cases in memory. Instead, much of the added cost lies in decomposing the case into partial cases and storing the partial cases in memory. This suggests that if cases are decomposed into partial cases at all, the decomposition should be done during problem solving, as and when needed.

Perhaps the result that most surprised us concerns the number of cases needed to bootstrap the case-based method. Theories of case-based reasoning (e.g., [Kolodner 1993]) assume that the number of cases needed to seed the case memory is small relative to the total number of problems in a given world. Thus, we hypothesized that, for a given navigation space, the number of cases needed to bootstrap the case-
based method would be a small fraction of the total number of problems in the navigation space. We expected that the problem-solving coverage of Router’s case-based method would increase rapidly with the number of initial cases in memory. But this did not happen. Instead, in Experiment 2.4, when we seeded the case memory with randomly generated problems, we found that Router’s case-based method needed about 16% of all possible problems before it started working effectively in that it could solve about half of all randomly-generated problems given to it. Further, we found that the number of problems Router’s case-based method can be solve increases approximately linearly with the initial number of cases in memory.

Note that in most case-based systems (e.g., [Hammond 1989]) the initial set of cases is handcrafted for a specific class of problems rather than randomly generated. In Experiment 2.4, in contrast, Router’s initial set of cases was randomly generated. The relatively small problem-solving coverage of Router’s case-based method and the relatively slow rise of the coverage as a function of the initial number of cases in memory, could be a result of Router’s task of navigation planning and domain of discrete pathways. The confirmation of this result requires additional experiments with different tasks in different domains, which is beyond the capability of Router.

Finally, Router’s integration of the case-based and model-based methods enabled us to partially evaluate its task-directed mechanism for multistrategy reasoning. In particular, we expected the efficiency of the multistrategy method to be no worse than that of the model-based method. We thought that since the task-directed strategy selector would select the case-based method if an appropriate case was available in memory and the model-based method if such a case was not available, and since the case-based method is more efficient than the model-based method, the efficiency of the multistrategy method would be no worse than that of the model-based method, especially if the number of cases in memory increased. But, in Experiment 2.1, we found that the computational cost of the multistrategy method was significantly more than that of the model-based method. This is because of the high cost of the task-directed mechanism for strategy selection.

The main benefit of task-directed mechanism for multistrategy reasoning, our Experiment 2.4 indicates, is robustness of reasoning, not the lowering of the processing cost. In this experiment, we found that if a
case appropriate to a given problem was available in memory, then Router used the case-based method. But if the given problem could not be solved by the case-based method because an appropriate case was not available, then Router used the model-based to solve the problem. Similarly, with in the high-level context of the case-based method, if the case adaptation strategy of combining multiple cases was infeasible because the needed cases were not available in memory, then Router used the strategy of model-based case adaptation.

7.4 Hypothesis 3

Router's topological model plays a central role both in model-based search and in case-based plan reuse. It defines the problem spaces for the model-based method. It also provides the indexing scheme for organizing the case memory and thus enables case retrieval. In addition, it enables model-based case adaptation, one of the two case adaptation strategies in Router. Thus, Router's scheme for partitioning the navigation space into neighborhoods and organizing the neighborhoods in a space-subspace hierarchy is a critical design decision.

The organization of Router's model is based on the principle of locality: spatially adjacent locations are aggregated into regions of space and the spatial regions are organized in an abstraction hierarchy. Marr [1976] probably first clearly articulated this principle in his work on computer vision. He described an image understanding process [Marr 1982] in which the "pixels" in a raw image get locally grouped and understood in terms of spatial aggregations called edges, and the edges in turn get locally grouped and understood in terms of higher-level spatial abstractions called surfaces, and so on.

This principle implicitly forms the basis of much of AI work on the organization of spatial knowledge and reasoning (e.g., [Davis 1986] [Lawton and Levitt 1990]). For navigation spaces characterized by discrete pathways, we operationalized the principle into two heuristics pertaining to the proximity and the significance of the pathways. The application of these heuristics led us to the design of Router's topological model in the form of space-subspace hierarchy of neighborhoods described in Section 3. Experiment 3.1 shows that this hierarchical organization of the topological model results in a significant improvement in the computational efficiency of Router's model-based search compared to the use of a flat map as illus-
trated in Figure 5. The cost of Router's problem solving is much smaller for such problems because its space-subspace hierarchy helps to significantly prune the search space.

This experiment also shows that the plans produced using the flat map are slightly more parsimonious plans than those generated by Router's model-based method. Actually, for 132 of 195 problems (67.7%), Router produced more parsimonious plans than the control system. Much of the increase in the average number of path segments in Router's plans is due to 3 of 195 (1.5%) problems for which it produced very long plans. Detailed examination of the data suggests that this is occurring because of the horizon effect of Router's breadth-first search method.

A more interesting issue is whether our design heuristics result in a robust partitioning of the neighborhoods such that minor variations in the neighborhood partitions make little or no difference to either the efficiency of model-based search or the quality of plans generated by the search method. In Experiment 3.2, Meta-Router makes modifications to Router's neighborhood partitions based on its analysis of Router's problem-solving traces and experimenter's feedback in the form of preferred paths for a sequence of 50 randomly generated problems. The modifications to the neighborhoods abide by our two heuristics for designing the topological model. As Figure 6 illustrates, the performance of Router's search method with the modified model on 195 randomly generated problems is nearly identical to its performance with the original model both in the parsimony of generated plans and the cost of generating them. This shows that Router's partitioning of the neighborhoods is robust. This also indicates that our design heuristics, which operationalize the principle of locality for navigation spaces characterized by discrete pathways, lead to robust partitioning of the navigation space.

An even more interesting issue is whether any partitioning and organization of topological knowledge based on the two design heuristics, not just tweaks on Router's space-subspace hierarchy, would be equally productive. In Experiment 3.3, Router starts with a flat map. Then, Meta-Router learns three different space-subspace hierarchies based on its analysis of Router's problem-solving traces and experimenter's feedback in the form of preferred paths for three sequences of 30 randomly generated problems each. As Figure 7 indicates, the cost of Router's model-based method with all three space-subspace hierarchies decreased significantly compared to search using a flat map. In addition, comparing Figures 5 and 7,
we found that the quality of plans produced using the three learned space-subspace hierarchies is almost identical to those produced using Router's original hierarchy.

In comparing Figures 5 and 7, however, we also noted that the cost of Router's model-based method with the three learned space-subspace hierarchies, while significantly lower compared to search using a flat map, is not as low as that of the original Router. In addition, in case of two of the three learned hierarchies, the model-based search failed to solve some of the 195 navigation problems as indicated in Figure 7. A detailed examination of the models, traces and generated plans suggests that both of these behaviors occurred because Meta-Router's process of pathway aggregation and neighborhood abstraction resulted in a loss of some connections between parent and children neighborhoods.

This data points to an important issue. We used the two design heuristics to handcraft Router's original space-subspace hierarchy. Apparently, in handcrafting the original hierarchy, our tacit knowledge played an important role in insuring that all pairs of parent-child neighborhoods in the hierarchy were well connected, i.e., there were a number of common intersections among them. But Meta-Router did not have access to this tacit knowledge. Thus, while it abided by the two design heuristics, Meta-Router did not insure that each parent-child neighborhood pair in the three learned hierarchies was well connected. Thus, Experiment 3.3 indicates that our Hypothesis 2 is not true. More precisely, the combined results of Experiments 3.1, 3.2, and 3.3 indicate that the two design heuristics pertaining to proximity and significance of pathways need to be complemented with a third one, namely, each parent-child pair of neighborhoods in the space-subspace hierarchy of neighborhoods needs to be amply connected.

7.5 Evaluation Methods

One of the reasons for reporting on an experimental study of three different hypotheses in this paper is that the hypotheses are all related to the same issue of multistrategy path planning in navigation spaces characterized by discrete pathways and engineered visual landmarks. Another important reason is that the study of the different hypotheses illustrates different evaluation issues and methods.

In AI experiments, the dependent variable in general is the behavior of a computer program. The
program instantiates an AI theory for the design of intelligent agents capable of addressing a class of problems, i.e., a class of tasks in a class of domains. The behavior of the program is what the AI experimenter observes. This behavior has three important dimensions: (i) the problem-solving coverage, i.e., the class of problems that the program solves; (ii) the quality of solutions produced by the program, where the operational measure of solution quality often is specific to the class of problems solved; and (ii) the computational cost of problem solving. Our experiments pertaining to Hypothesis 2, for example, measure Router’s behavior in all three dimensions.

The behavior of an AI program in general may depend on both on the structure of the program’s external environment and its internal design. The program’s external environment and its internal design thus are the independent variables in AI experiments. Again, both independent variables may have more than one dimension. The design of Router, for example, is characterized by several variables such as the type of knowledge used (models, cases and partial cases), the organization of this knowledge (space-subspace hierarchy of neighborhoods), the methods (model-based search and case-based plan reuse for the task of path planning, recursive case combination and model-based plan modification for the task of case adaptation), and the control of processing (method-directed spawning of subtasks, task-directed method selection and integration).

Different hypotheses in AI experiments pertain to different relationships between the independent and the dependent variables. In some hypotheses, the structure of the program’s external environment is a central issue. For example, our Hypothesis 1 pertains to the relationship between the structure of the program’s environment, the knowledge boundary between path planning and reactive control, and the behavior of the Raura program. But real environments inherently are both noisy and open. They may contain features unknown to the program. They may even contain features unknown to the experimenter. This implies that a simulation is insufficient evaluation of an AI hypothesis that specifically pertains to the structure of the environment of a program. This is because any simulation can at best only cover the factors that are known to the experimenter. Therefore, the evaluation of this kind of hypothesis requires the operation of the program in the environment itself, not in a simulation of the environment. This is why we embodied Raura in the Stimpy robot and observed the behavior of Stimpy on an office floor in the real MaRC office building.
The key issue in other AI hypotheses is not so much the structure of the program's external environment as much as the structure of the program's internal design. These hypotheses typically relate the functional design of the program and the output behavior of the program, under some assumptions about the external environment. For example, our Hypothesis 2 pertains to the relationship between two functional designs of path planners and their output behaviors, where the functional designs can be model-based or case-based. For hypotheses of this kind, ablation experiments provide a useful evaluation method. The experimenter may design a computer program that contains design elements corresponding to the different functional designs in question. Then the experimenter may lesion different design elements in different stages of the experiment, and observe the program's behaviors. At the end, the behaviors of the program can be related to the different functional designs. This is what we did in Experiments 2.1, 2.2 and 2.3.

But even AI hypotheses pertaining primarily to the internal design of the program can run into the problem of "unknown factors". This problem typically is manifested in the knowledge contained in the program. Typically the experimenter handcrafts the (initial) knowledge in the program, including the schemes for representing and organizing the knowledge. But the product of this handcrafting of knowledge often is based in part on the tacit knowledge that the experimenter brings to the program. For example, we handcrafted Router's space-subspace hierarchy of neighborhoods. We thought that our design of the space-subspace hierarchy is based only on the two design heuristics concerning the proximity and significance of pathways. But our tacit knowledge of the class of problems that Router is intended to solve might have played a role in the construction of the hierarchy. The evaluation issue thus becomes how to account for this unknown factor.

One part of the answer to this question lies in endowing the program with the capability of autonomous acquisition of knowledge where the knowledge acquisition is governed by the same design heuristics. For example, Experiment 2.4 examines the dependencies between the number of cases in memory and the problem-solving coverage of Router's case-based method. But Router acquires the cases autonomously, though compilation of solutions to generated by the model-based method. This removes the unknown role that our tacit knowledge may have played in handcrafting the cases. Similarly, our Hypothesis 3 pertains to the relationship between a scheme for partitioning and organizing topological knowledge and the behavior of Router. Initially, we handcrafted Router's topological model using two design heuristics.
This introduced an unknown factor in the design of Router. By creating a Meta-Router program that can autonomously reorganize Router’s topological knowledge based on the same heuristics, we eliminated the unknown role our tacit knowledge may have played in Router’s behavior. By comparing the performance of Router with the handcrafted and the learned models, we uncovered the role played by our tacit knowledge.

In sum, our experiments with the Router family of systems illustrate that different AI hypotheses raise different evaluation issues, and thus require different evaluation methods and experimental designs. One of the hard tasks in AI experiments is to first classify the dependency between the behavior of a program and structures of its external environment and internal design under study, and to then employ an evaluation method that can illuminate the specific dependency while accounting for different design decisions and “unknown factors”.

Acknowledgements

This work has benefited from many discussions with members of the AI and Robotics groups at Georgia Institute of Technology. We are especially grateful to Professor Ronald Arkin at Georgia Tech for allowing us to us to play with AuRA and Stimpy. We are also thankful to Todd Callantine, Michael Donnellan, Andres Gomez, Paul Rowland, Murali Shankar, and Juan Santamaria for their contributions to the Router, Meta-Router and Raura projects. This research has been partially 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


