

## Pragmatic navigation: reactivity, heuristics, and search

Susan L. Epstein \*

*Department of Computer Science, Hunter College and The Graduate School of  
The City University of New York, 695 Park Avenue, New York, NY 10021, USA*

Received 17 April 1997; revised 15 October 1997

---

### Abstract

FORR (FOr the Right Reasons) is an architecture for learning and problem solving that integrates a possibly incomplete and overlapping set of solution methods to address complex problems. Each method, although it represents some facet of domain expertise, may vary in reliability and speed. The principal contribution of this paper is the extension of FORR to include situation-based behavior (the serial testing of known, triggered techniques for problem solving in a domain) with reactivity and heuristic reasoning. FORR categorizes methods as reactive, heuristic, or situation-based, and addresses problem solving with one category of methods at a time. A hierarchical reasoner first has the opportunity to react correctly. If no ready reaction is computed, the reasoner activates a set of reactive triggers for time-limited search procedures tailored to specific situations. If they, too, fail to produce a response, the reasoner resorts to collaboration among heuristic rationales. All three components reference knowledge learned from experience. In a series of experiments, this architecture is shown to be effective and efficient. Ablation experiments demonstrate how each component plays an important role in problem solving. Additional contributions of this paper include a FORR-based, pragmatic, cognitively plausible approach to navigation with learned heuristic approximations that describe two-dimensional territory and travel experience through it, and a careful study of how situation-based behavior, reactivity, and heuristics interact there. Empirical evidence demonstrates that the resultant system is both effective and efficient, and guidelines for generalization to other domains are provided. © 1998 Elsevier Science B.V.

*Keywords:* AI architectures; Heuristic search; Machine learning; Multistrategy learning; Navigation; Problem solving; Satisficing; Situation-based search; Spatial representation

---

---

\* Email: epstein@roz.hunter.cuny.edu.

*Naive geographic reasoning is probably the most common and basic form  
of human intelligence—Egenhofer & Mark, 1995*

When confronted with an intractable search space, people employ a variety of devices to make what they hope will be expert decisions. Some of this behavior is automatic; certain perceptions of the world trigger action without conscious reasoning. AI researchers have modeled this automaticity with *reactive systems*. Other portions of this behavior are heuristic; limitedly rational reasoning principles are applied in some combination. There is, however, another important mechanism people use, a kind of restricted search. *Situation-based behavior* is the serial testing of known, triggered techniques for problem solving in a domain. The principal contribution of this paper is the extension of FORR (FOR the Right Reasons), an architecture for learning and problem solving, so that it integrates situation-based behavior with reactivity and heuristic reasoning. To solve a complex problem, FORR structures a possibly incomplete and overlapping set of heuristic solution methods. Each method, although it represents some facet of domain expertise, may vary in reliability and speed. Additional contributions of this paper include a pragmatic, cognitively plausible approach to navigation with FORR; a careful study of how situation-based behavior, reactivity, and heuristics interact there; and empirical evidence that the resultant system is both effective and efficient.

Consider first how reactivity, situation-based behavior, and heuristics would behave separately on a path-finding example. In the grid world studied here, the robot knows its own coordinates and those of the goal. (Other kinds of grid worlds could be treated similarly.) The robot has no map; it can “see” only to the nearest obstruction in four orthogonal directions, and a decision constitutes a move in a straight line to any visible location. Fig. 1 shows a goal at (9,6) and three possible locations for a robot, with the legal moves from R1 striped. Unilateral application of reactivity, situation-based behavior, or heuristics would encounter difficulties here. Without record of its recent experience and the ability to learn, a purely reactive system could become mired in local cyclic behaviors. Without every relevant decision-making rule, a purely heuristic system could err, resorting to either random choice or search when no heuristic was applicable. And without efficient decision making, a purely situation-based system would incur heavy search costs.

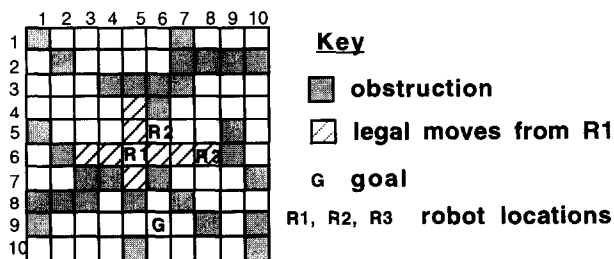


Fig. 1. Robot *R* seeks to move to goal *G*.

In contrast, FORR strikes an economical balance among reactivity, situation-based behavior, and heuristics. Rather than exhaustively deliberate on a complete model of the world, FORR responds in a variety of efficient ways to a partial, feature-based model. As a three-tiered architecture, it gives priority first to correct reactions, then to situation-based behaviors, and finally to heuristics, all with reference to the model learned during problem solving. Reactivity is given top priority because it is an inexpensive way to make the right obvious decisions and to avoid the wrong obvious ones. Although it is costly, situation-based behavior relies upon triggers to justify the appropriate expenditure of computational cycles on search, and relies upon restricted routines to search in a very limited manner. Heuristics thus become a last, albeit frequent, resort, applied when the right decision is not obvious and the problem solver cannot justify restricted search in the partial model.

Consider how FORR would deal with each of the robot locations in Fig. 1. If a robot at *R1* contemplated a move to (7,5), but had been there before and remembered that it was a dead-end, a correct reaction would be to eliminate that move from any further consideration. Now consider a robot at *R2* with no immediate reactions. If it had made little progress on its task, and recognized that it was aligned with the goal but there was an intervening obstruction, FORR could activate a time-limited search algorithm to circumnavigate that obstruction. If the search algorithm were able to realign the robot with the goal, say at (9,9) with the path

$\langle (6,6) (6,7) (7,7) (7,8) (8,8) (8,9) (9,9) \rangle$

then that entire path would be proposed and executed as a way of addressing the recognized situation. Finally, consider a robot at *R3*, one that has no immediate reactions and does not recognize any particular situation. Aligning the robot with the goal is generally a good heuristic, so (6,6) might be a good choice. Then again, if the robot is having difficulty on the trip, trying a location it has not yet visited, say (4,8) or (6,5), might be helpful. Of course, large steps speed travel, so (3,8), (6,3), and (8,8) might be good choices. Since two of those choices are in the general direction of the goal, a long-step heuristic might value them more highly. Achieving a consensus from heuristics such as these is nontrivial. (Happily, the dilemma at *R3* will be resolved in Section 4.)

*Pragmatic navigation*, as implemented here by FORR, aspires to provide robust, competent performance in two-dimensional space, performance that improves across time and is resilient to changes in origin and destination. Instead of pre-engineered expertise for a particular territory, a pragmatic navigator learns its way around a new territory from a series of trips through the territory, trips whose origins and destinations differ. Just as a person who travels efficiently through a campus does not retain a detailed description of each trip or remember the location of each tree and rock, a pragmatic navigator ignores much travel history and many topographical details. Instead of a detailed map, pragmatic navigation relies upon features that support efficient travel or make it more difficult, such as a door into a room or an extended wall. In a new territory, a pragmatic navigator initially performs as a competent novice, and then, as it learns features of the territory from traveling there, it uses that knowledge to improve its performance. As envisioned here, most features are heuristic rather than absolutely accurate; they are useful approximations that describe a territory and travel experience

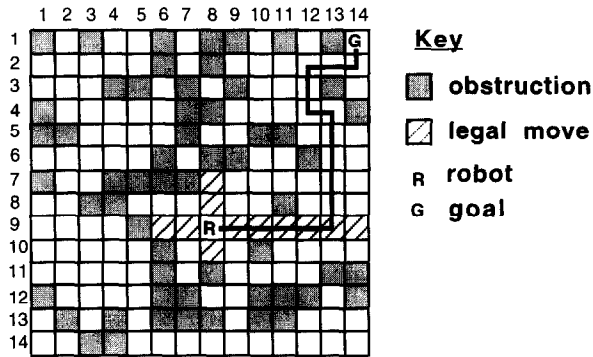


Fig. 2. A problem and its solution in a maze.

through it. Such representation deliberately sacrifices detail in exchange for efficient storage, retrieval, and computation.

It is the thesis of this work that expert navigation through unmapped territory can be achieved as the integration of correct reactions, situation-based behavior, and carefully balanced heuristics, many of which reference features learned about the territory during travel there. This approach works because it appropriately allocates control and responds to context, both in a cognitively plausible way. The first section of this paper frames navigation as a set of travel tasks for a perceptually-limited agent in unknown territory, explains why traditional AI search algorithms find that so difficult, and proposes pragmatic navigation as an alternative. The second section describes FORR, the underlying architecture for learning and problem solving that balances search, reactivity, and heuristics. Section 3 identifies territory features for pragmatic navigation and how they can be learned. Section 4 summarizes navigation principles for decision making, and offers examples of how FORR coordinates those navigation principles with learned knowledge. Section 5 formulates and reports on a series of experiments that demonstrate the strengths of pragmatic navigation and identify the FORR components responsible for its performance. Section 6 provides a full discussion of the results, including guidelines for applying these ideas in other domains. The final sections address related and future work. Algorithmic details are reserved to the Appendix.

## 1. The task and an approach

The pragmatic navigation task was first suggested as a problem that would challenge the power of search algorithms [25]. The robot's world is a *maze*, a discrete, rectangular grid with external walls and internal obstructions like the  $14 \times 14$  maze that is 30% obstructed in Fig. 2. (All the examples in this paper are taken from actual runs; the experiments themselves are on substantially larger mazes than this one.) A *location*  $(r, c)$  in a maze is the position in the  $r$ th row and  $c$ th column, addressed as if it were an array. A *problem* is to travel from an initial robot location  $R$  to some goal location  $G$  in

a sequence of legal moves, that is, to find a (not necessarily optimal) path to the goal. In Fig. 2 the problem is to travel from (9, 8) to (1, 14). In any state, the robot senses only its own coordinates, the coordinates of the goal, the dimensions of the maze, and the distance north, south, east, and west to the nearest obstruction or to the goal. The robot does not sense while moving, only before a move. The robot also remembers any useful knowledge acquired in previous trips through this maze. (This domain is similar to privet-hedge mazes such as the one grown for royalty at Hampton Court, and to the computer game Maze Wars.) The robot knows the path it has thus far traversed in the current problem, but it is not given, and does not construct, an explicit, detailed map of the maze like the one in Fig. 2. Rather, it learns features of that map as described in Section 3. As the problem level increases, this task, even with a map, is nontrivial.

Intuitively, a legal move passes through any number of unobstructed locations in a vertical or horizontal line. More formally, a *legal move* is a transition from a state where the robot is at  $(r, c)$  to one where it is at  $(r', c')$  such that exactly one of the following is true:

- $r = r', c < c'$  and unobstructed  $(r, c + 1), \dots, (r, c')$ ,
- $r = r', c > c'$  and unobstructed  $(r, c - 1), \dots, (r, c')$ ,
- $c = c', r < r'$  and unobstructed  $(r + 1, c), \dots, (r', c)$ ,
- $c = c', r > r'$  and unobstructed  $(r - 1, c), \dots, (r', c)$ .

The robot in Fig. 2 has 11 legal moves: north to (7, 8) and (8, 8), east to (9, 9) through (9, 14), south to (10, 8), and west to (9, 6) and (9, 7). A problem is *solvable* if and only if there exists some path

$$\langle R = loc_0 \text{ move}_1 loc_1 \dots loc_{i-1} \text{ move}_i loc_i \dots loc_{p-1} \text{ move}_p loc_p = G \rangle$$

such that  $\text{move}_i$  is a legal move from  $loc_{i-1}$  to  $loc_i$  for  $1 \leq i \leq p$ . The *level of difficulty* of a solvable problem is the minimum value of  $p$  for which there is a solution, that is, the minimum number of legal moves with which the robot can reach the goal. (Effectively, the level of difficulty of a problem is one more than the minimum number of left or right turns the robot must make to reach the goal.) Note that this is different from the Manhattan distance from  $R$  to  $G$ . Fig. 2 is a level-6 problem; one six-move solution for it, with Manhattan distance 16, is indicated there:

$$\langle (9, 8) (9, 13) (4, 13) (4, 12) (2, 12) (2, 14) (1, 14) \rangle.$$

Interchanging the robot and the goal produces another problem at the same level. The heading of the task is the subset of {north, east, south, west} that describes the direction from  $R$  to  $G$ . The heading of the task in Fig. 2 is {north, east}.

This task has several performance criteria. The robot is expected to solve multiple problems in the same maze. Lower-level problems should be easier to solve. The robot is expected to perform quickly. Speed can be measured as elapsed computation time, number of decisions, or path length (Manhattan distance) traveled. The robot is also expected to perform efficiently. Efficiency can be measured as the number of distinct locations visited or the percentage of repeated locations in a path. Finally, the robot is expected to learn; its performance should improve with experience. (There is an important distinction here between the number of decisions and number of moves. As

described below, the system does some search that may not be part of the problem solution but consumes resources. Thus, a *move* appears in the solution path, while a *decision* is a step taken during problem solving, whether or not it appears in the solution.)

As Section 5 demonstrates, the task formulated above is not amenable to traditional AI techniques. Depth-first search requires substantial backtracking; its paths are long and repetitive. Breadth-first search visits too many nodes; on most hard problems it approaches exhaustive search while it visits a high proportion of nodes in the search space and maintains a very large structure for open paths. Means-ends analysis is inapplicable because it requires knowledge about the vicinity of the goal to reason backwards. Best-first search with a “sensible” evaluation function, such as Euclidean distance to the goal, is easily misled by *deceptive* problems where proximity is not a valid indicator of progress. (The goal might be hidden behind a long wall so that the robot must move away from the goal to reach it eventually. For example, placing the robot at (9,4) and the goal at (8,5) in Fig. 2 produces a deceptive level-6 problem.) For a large maze, explicit search would be extremely inefficient, perhaps intractable. With FORR, pragmatic navigation offers an alternative.

## 2. The underlying architecture

FORR is a problem-solving and learning architecture that models the transition from general expertise to specific expertise, and capitalizes on methods that people use [10]. FORR approaches problem solving as a sequence of reasonable decisions. A *task* is a problem-solving experience where a sequence of moves is intended to reach a desired state, such as the robot moving from *R* to *G* in Fig. 2. A *problem class* is a set of related tasks, such as mazes that simulate furnished rooms or mazes that simulate office suites. A *domain* is a set of related problem classes, such as grid-world mazes. A FORR-based system begins with a domain and some domain-specific but problem-class-independent knowledge, such as “avoid dead-ends”. With task experience, a FORR-based program gradually acquires *useful knowledge*, problem-class-specific data that is potentially useful and probably correct. This useful knowledge, such as dead-ends in a particular maze, should enhance the performance of a FORR-based system.

Although most useful knowledge is tailored to a specific domain, there are broadly applicable useful knowledge items that FORR provides and learns by default. Those relevant to pragmatic navigation are calculated after each task. Average task length is the number of moves made. Total learning experiences is the number of tasks attempted. Openings, stored as a tree, are the first  $x\%$  of the moves recorded for a task. (Throughout this paper terms such as “few”, “recent”, or “ $x\%$ ” indicate parameters that are preset by the user. A full listing of the parameter values used in the experiments described here appears in the Appendix.)

FORR’s three-tier hierarchical model of the reasoning process is shown in Fig. 3. An *Advisor* is a domain-specific but problem-class-independent, decision-making rationale, such as “get closer to your destination”. Each Advisor is a “right reason”, implemented as a time-limited procedure. Input to each Advisor is the current state of the world,

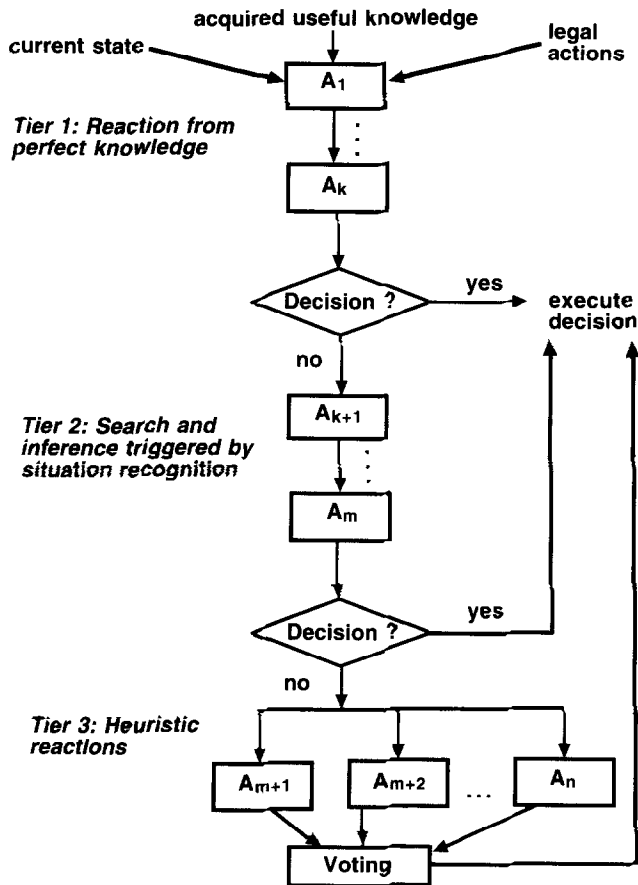


Fig. 3. How FORR makes decisions.

the current permissible actions from that state, and any learned useful knowledge about the current problem class. Each Advisor outputs any number of *comments* that support or discourage permissible actions. A comment lists the Advisor, the action commented upon, and a *strength*, an integer in  $[0, 10]$  that measures the intensity and direction of the Advisor's opinion. (Strengths above 5 are construed as support, below 5 as opposition.) Although there are no constraints on the nature of the comment-generating procedures themselves, a FORR-based system is intended to sense the current state of the world and respond with a rapid computation, that is, to avoid extensive search.

Tier-1 Advisors sense the current state of the world and what they know about the problem class; if they make a decision, it is fast and correct. They are consulted in a predetermined, fixed order. Each Advisor may have the authority to make a decision alone or to eliminate a legal action from any further consideration. Tier-1 Advisors are reactive and reference only correct useful knowledge. An important tier-1 Advisor in a FORR-based navigator would be "if you see the goal directly ahead, go to it". Only

when the first tier of a FORR-based system fails to make a decision does control default to the next tier. Tier 1 is like the sense–compute–execute loop of a carefully constructed and perfectly accurate reactive system.

Tier-3 Advisors, in contrast, are not necessarily correct in the full context of the state space. Each of them embodies a heuristic, specialized view of reality that can make a plausible argument for or against one or more actions. Tier-3 Advisors are reactive too, but far less trustworthy, because neither their reasoning process nor the useful knowledge on which they rely is guaranteed correct. Once control is relegated to tier 3, all tier-3 Advisors have an opportunity to comment before any choice is made. The decision they arrive at is the action with highest total strength. In a FORR-based navigator, a good tier-3 Advisor would be “minimize your distance to the goal”. Although a tier-3 Advisor is reminiscent of a heuristic rule in an expert system or a term in an evaluation function for a search algorithm, there are two important differences. First, because a tier-3 Advisor may rely on useful knowledge, its reactions to the same state may change with experience. Second, a tier-3 Advisor may have an inappropriate (incorrect or irrelevant) perspective for a particular problem class, and FORR may learn not only to disregard it, but also learn to refuse to allocate computational resources to it. As a result, appropriate control for decision making in tier 3 is neither obvious nor trivial. Further details appear with an example in Sections 4 and 5.

As the results in Section 5 indicate, tiers 1 and 3 alone do not solve the most difficult problems quickly and reliably enough; some search is necessary. FORR implements certain time-limited, situation-based searches with the Advisors of tier 2. Situation-based behavior is based upon psychologists’ reports about human experts in resource-limited situations [23]. For example, an emergency rescue team is called to the scene of an attempted suicide, where a person dangles from a sign after jumping from a highway overpass. Time is limited and the person is semiconscious. During debriefing after a successful rescue, the commander of the team describes how they immediately secured the semiconscious woman’s arms and legs, but then needed to lift her to safety. He mentally retrieved, instantiated, and tested four devices that could hold her while the team lifted, one device at a time. When a device failed in his mental simulation, he ran the next. When the fourth scenario ran several times in simulation without an apparent flaw, he began to execute it in the real world. Klein and Calderwood describe the predominance of this situation-based behavior in 32 such incidents, and cite additional evidence from studies of army commanders, business executives, juries in deliberation, judges setting bail, highway engineers, and nuclear power plant operators [23]. Its key features, for the purposes of this discussion, are that a situation triggers a set of procedural responses, not solutions, and that those responses are not tested in parallel.

Each tier-2 Advisor has a reactive trigger and a procedure that generates and tests a highly-constrained set of possible solution fragments. A *solution fragment* emerges from a tier-2 Advisor as a sequence of decisions, rather than a single reactive one, a digression from the “sense–compute–execute” loop. A tier-2 Advisor triggers when it recognizes that its method may be directly related to the current situation, for example, when the robot is aligned with the goal but there is an intervening wall. Execution of a tier-2 Advisor instantiates and tests one or more possible solution fragments, for example, paths to circumnavigate the intervening wall. Tier 2 is prioritized like tier 1, but lacks



any guarantee of correctness. Until one of them produces a solution fragment, each tier-2 Advisor that triggers is ceded control and given limited time to develop a solution fragment. Once some tier-2 Advisor constructs a solution fragment, that sequence is executed (one move at a time, subject to override from tier 1) and then, regardless of the outcome, control is returned to tier 1. Decisions during search are charged to the robot, whether or not any fragment step becomes a move because it is recommended by the Advisor as a result of the search. If no tier-2 Advisor triggers or produces a sequence of recommended steps, tier 3 will make the decision.

FORR is implemented in Common Lisp. To apply FORR to a domain, one specifies the domain, its problem classes, its useful knowledge, and procedures to learn it. Pragmatic navigation is implemented as the FORR-based system *Ariadne*. (*Ariadne*, daughter of King Minos of Crete, told Theseus how to find his way through the labyrinth that protected a great treasure.) The next two sections detail *Ariadne*'s useful knowledge as a set of features for two-dimensional space, and sketch *Ariadne*'s Advisors, navigation principles for path finding.

### 3. Learning to represent territory

Instead of a map, pragmatic navigation relies upon two kinds of features to describe a territory: those that support efficient travel (*facilitators*) and those make it more difficult (*obstructors*). These features constitute *Ariadne*'s useful knowledge, and are learned for a specific problem class from experience. Although it may be approximate, useful knowledge in FORR is expected to enhance performance. Other than the default items described in the preceding section, each kind of useful knowledge in a FORR-based program must be prespecified by the system designer, including its learning algorithm, learning time limit, and learning schedule (after a decision, a task, or a set of tasks). This section describes what *Ariadne* learns, when it learns, and how it does so.

#### 3.1. *Facilitators*

*Ariadne* identifies three kinds of facilitators: gates, bases, and corners. A gate has, in theory, the ability to provide a transition from one large segment of space to another. A base repeatedly appears as a counter-intuitive choice in successful paths. A corner offers the possibility of a new direction.

Since it knows the dimensions of the maze, *Ariadne* can calculate quadrants. A *gate* is a location that offers a transition from one quadrant of the maze to another. After each move, *Ariadne* tests whether its quadrant has changed, that is, if it has moved through a gate. If so, the robot's current location is learned as a gate between the current quadrant and the previous one. A gate may not always be helpful; for example, (8, 10) is a gate between quadrants 3 and 4 in Fig. 4, but it offers access to little of quadrant 3. Each gate is stored with its *extent* (rectangular approximation of the locations from which it can be reached) in a hash table whose key is the sorted pair of quadrant numbers. The extent of (8, 10) in Fig. 4, for example, is the rectangle with vertices (6, 10), (9, 10), (9, 5), and (6, 5). *Ariadne* only learns the gates it visits, so many locations in Fig. 4

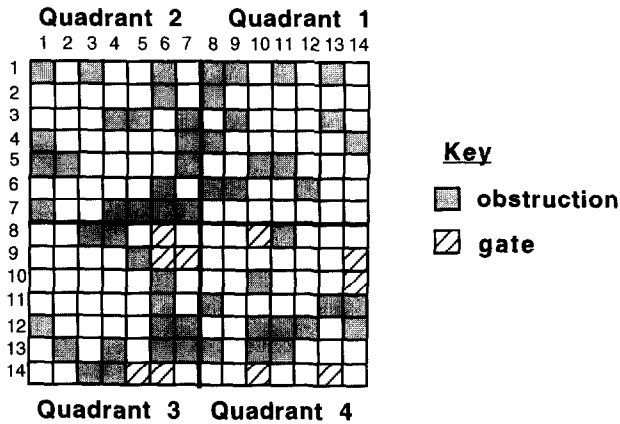


Fig. 4. After 10 tasks, gates learned for a simple maze.

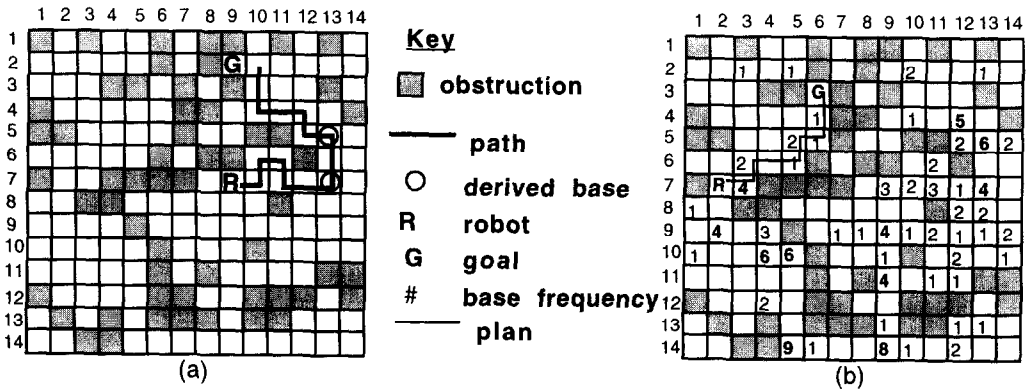


Fig. 5. (a) A solution path and the bases that arise from it. (b) A plan for a task formulated from learned bases.

that satisfy the definition of gate were not learned and are not marked. For example, during some trip, the robot moved into (14, 10) from quadrant 3, and therefore learned (14, 10) as a gate. Although (14, 9) is also a gate, it was never experienced as such, and therefore was not learned as a gate. (That position is, however, identified below as another kind of facilitator.) The subdivision of the maze into only four areas (the quadrants) was deliberate. Specifying  $n$  areas produces as many as  $nC_2$  gate categories to manage, while fewer areas provide too little transition information.

A *base* is a location in a maze that appears to have been a key to a successful path. In the author's home town, people regularly give directions beginning "First you go to the Claremont Diner." Although it served memorable cheesecake, the Claremont Diner burned down 15 years ago, and there is nothing particularly significant about

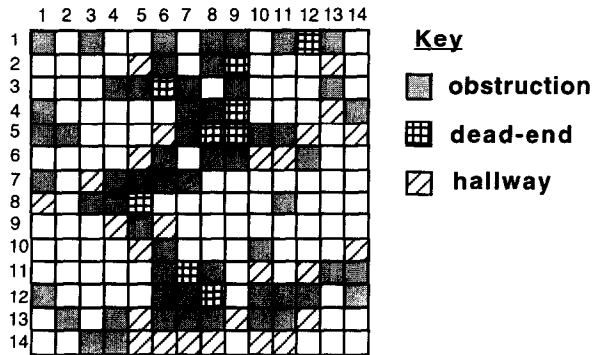


Fig. 6. After 10 tasks, corridors learned for a simple maze.

the car dealership that has replaced it. What is significant is that the diner was at a location that affords ready (not necessarily shortest path) access to other locations in a 10-mile radius. A base is such a location. Bases are learned after a successful task; the algorithm first corrects the path to eliminate loops and unnecessary digressions. A base is a location in that corrected path that was not in the heading from  $R$  to  $G$ . A base is not a dead-end or  $G$  itself, and solution fragments constructed by tier-2 Advisors that circumvent walls contribute only their most extreme positions opposite the original headings. Bases are stored in a hash table with their *frequency* (the number of times they have been identified in different problems). The bases learned from one solution path are circled in Fig. 5(a), where the heading was {north} and the eastern-most corners became bases.

Bases facilitate some primitive, high-level planning for Ariadne's decision making. A plan in Ariadne is a sequence  $\langle b_0 b_1 b_2 \dots b_{i-1} b_i b_{i+1} \dots b_n b_{n+1} \rangle$  where  $b_0$  is the robot's current location,  $b_{n+1}$  is the goal,  $b_i$  is a base for  $i = 1, \dots, n$ ,  $b_i$  is aligned vertically or horizontally with  $b_{i+1}$ , and either  $b_{i-1}$  is closer to  $b_0$  than  $b_i$  is, or else  $b_{i+1}$  is closer to  $G$  than  $b_i$  is. Plans are constructed from bidirectional search on aligned bases, with preference for bases of higher frequency, as if there were no obstructions. A plan fails when the robot is at some  $b_i$  and there is an intervening obstruction that prevents its move to  $b_{i+1}$ . An example of a plan Ariadne formulated for a task is shown in Fig. 5(b), where the bases learned after 20 level-6 tasks in the same maze are indicated by their frequency values. In Fig. 5(b), some bases, such as (14, 5) and (14, 9) are the keys to the only route between the eastern and western portions of the maze. Others, such as (7, 13) and (9, 9) lie at important intersections, like the Claremont Diner.

A *corridor* is a passageway of width one that either has a single exit (a *dead-end*) or is a *hallway*. A *pipe* is a straight hallway, that is, its endpoints lie in the same row or the same column. In Fig. 6, there is a hallway from (13, 5) to (14, 8) and a pipe from (14, 10) to (14, 11). Some pipes offer a view of the space after their far end. For example, from (14, 9) in Fig. 6 the robot can move not only to the ends of the pipe from (14, 10) to (14, 11), but also beyond it to (14, 12). Other pipes do not offer a view of the space after their far end, but since a pipe is not a dead-end, that promises

a turn in some other direction. For example, from (4, 10) in Fig. 6 the robot can see both ends of the (length one) pipe at (4, 13) but not beyond it; that promises a turn at the far end. Such a turn-promising far end of a pipe is called a *corner*. A corner can be helpful when the move to it is orthogonal to a direction in which the robot actually seeks to travel. If, for example, the robot were located at (4, 10) in Fig. 6 and the goal were to the south, moving east to the known corner at (4, 13) promises the ability to travel either north or south. A corridor is learned when, from the current state, the robot has only one or two moves. The two endpoints of a corridor serve as the keys to a hash table which also indicates whether or not they are for a dead-end. Corridors are enlarged and merged together as necessary. Like gates, only corridors that Ariadne experiences are learned.

### 3.2. Obstructors

Ariadne identifies four kinds of obstructors: chambers, bottles, barriers, and certain corridors. Chambers and bottles are circumscribing rectangular approximations of restricted spaces that are less narrow than a corridor, and have one or more exits. A barrier is a linear approximation of contiguous obstructed positions. A corridor may obstruct movement either because it is a dead-end, such as (2, 9) in Fig. 6, or because it is a non-corner pipe. In the latter case, movement into it necessitates an extra decision because its internal locations afford access only to each other. For example, in Fig. 6 from (14, 9) there is no reason to pause at (14, 10) or (14, 11) unless the goal lies there; if traveling east efficiently, one would go through that pipe, not pause within it.

Learning an obstructor is often triggered when the robot has been hampered in its ability to move through space. There are several measures of such confinement, all termed *recently constrained*:

- The area of the territory covered by the last  $x\%$  of the moves was less than  $y\%$  of the total maze area.
- All the legal moves have been visited at least once.
- The last  $x\%$  of the moves was less than  $y\%$  of the possible maze locations.
- $z\%$  of the recent moves were visited at least once before.

A *chamber* is an irregularly shaped space with an access point and an approximate extent. The extent is a bounding rectangle, a compact, heuristic approximation of the furthest in each direction one can go in the chamber. The *access point* is a location within the chamber that affords a view outside it, but need not be on the border of the extent. Fig. 7(a) shows a chamber with extent 1 north, 13 east, 5 south, and 9 west. When the robot moved to access point (4, 13) it saw beyond that extent to the south. In principle, all locations reachable from the robot's initial position really constitute a single large chamber, but the chambers that Ariadne learns are more limited and room-like. The learning algorithm for a chamber is triggered when the task has been underway for some time, the robot has been recently constrained and has been in its current location before, there are few legal moves to locations not yet visited on this task or the goal is remote, and the current location was not the result of a tier-2 fragment. The learning algorithm for a chamber first estimates the dimensions according to its current sensing, and then tries to move to a location where the chamber appears both

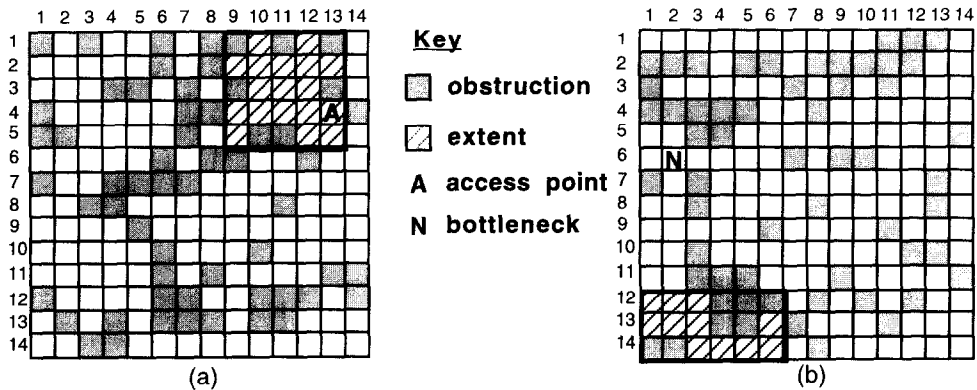


Fig. 7. (a) A learned chamber, with its extent and access point. (b) A learned bottle with its neck and extent.

higher and wider. From its current location the robot scans once horizontally, and then from the scanned location offering the largest view scans once again vertically. (If there are no horizontally-adjacent legal locations, the vertical scan is performed first.) If the procedure identifies a sequence of one or two locations (access points) that enlarge the extent, at least one of which is previously unvisited during this task, it records the chamber's extent and access point on a list. The scan for the chamber in Fig. 7 began from (4, 12). A new chamber may subsume an old one, in which case it replaces it on the list. Otherwise, chambers are not merged, and they may overlap or have more than one access point.

A *bottle* is another useful knowledge description of a constrained subspace, similar to a chamber. Chambers, however, are learned deliberately by search, whereas bottles are learned without search from analysis of the entire path after a task is completed. A potential bottle begins with a location that was visited more than once, and is repeatedly extended in both directions along the path by immediately neighboring positions only if it includes several spots, is not corridor-like, and does not ultimately encompass more than  $x\%$  of the area of the maze. Once a bottle is identified and its extent computed, its *neck* (not necessarily contiguous entry and/or exit point) is identified. Bottles are stored in a hash table as an extent and a neck. Fig. 7(b) shows a bottle with extent 12 north, 6 east, 14 south, and 1 west, and neck (6, 2).

A *barrier* is a linear approximation of a wall that obstructs movement. Fig. 8 shows the barriers learned after 10 tasks in a simple maze. The barrier from (13, 10) to (11, 13), for example, is an approximation of the irregular wall in the lower right corner of the maze. Barriers are learned different ways, depending upon the search context in which they arise. There are four tier-2 Advisors whose search paths have clearly prioritized preferences for the direction in which they move. Two attempt to circumnavigate an intervening obstruction, one attempts to shift to the opposite side of the goal, and another follows along contiguous obstructions. All of these searches treat dead-ends as obstructed space. Whether or not these Advisors produce solution fragments, barriers are detected from the paths their searches followed. In each case,

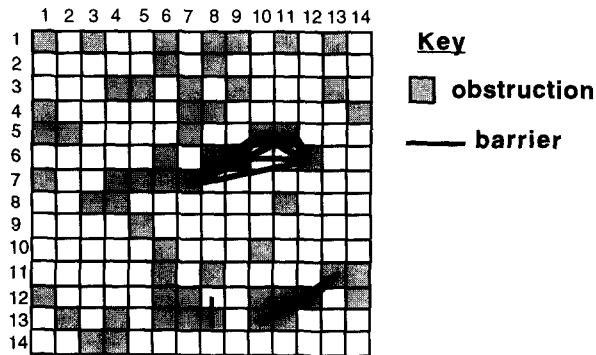


Fig. 8. After 10 tasks, barriers computed for a simple maze.

the learning algorithm is a function of the preferences in the rationale that produced the search path. Experience and Ariadne's recourse to tier 2 determine which barriers are encountered; thus there could have been some barriers learned to describe the irregular vertical wall between the eastern and western portions of the maze in Fig. 8, had the program encountered difficulty when required to avoid it. Each retained barrier is an object with endpoints, slope, intercept, and length.

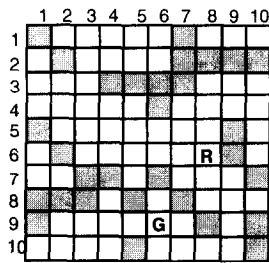
#### 4. Pragmatic navigation in action

An Advisor is a narrow decision-making rationale; in Ariadne, an Advisor is designed for maze navigation in general, rather than for some particular maze. As described in Section 2, an Advisor is characterized by the tier in which it resides: reactive Advisors in tier 1, situation-based Advisors in tier 2, and heuristic Advisors in tier 3. Recall too that input to an Advisor is always the same: in this domain, the dimensions of the maze, the location of the robot, the location of the goal, the legal moves for the robot, the trip history, and the features of the maze. Although every Advisor has access to the entire useful knowledge store, most apply no more than one or two items.


Table 1 lists Ariadne's 32 Advisors with the useful knowledge they reference. (A discussion on the origin of Advisors appears in Section 6.) Advisors in tiers 1 and 2 appear in their prioritized order. Full descriptions of the Advisors' behavior and their parameter values during the experiments appear in the Appendix. The four tier-1 Advisors are perfectly correct, reactive procedures that decide quickly. The eight tier-2 Advisors are situation-based procedures that do time-limited search in an attempt to produce a sequence of moves that they then mandate. Each tier-2 Advisor has a trigger that signals its applicability and a search method that attempts to compute solution fragments to address the identified situation. The solution fragments generated by a tier-2 Advisor are tested serially. The 20 tier-3 Advisors are reactive, time-limited heuristics that embody path-finding commonsense and do no search in the maze. Each may recommend or oppose any number of legal moves that have not already been

Table 1  
Ariadne's Advisors with tiers 1 and 2 in prioritized order

Tier	Advisor	Rationale	Useful knowledge
1	Victory	If the goal is reachable by a legal move, go there.	—
1	Can Do	Move adjacent to the goal.	—
1	No Way	Avoid dead-ends.	Corridors
1	Pipeline	Avoid internal locations in straight corridors.	Corridors
2	Roundabout	Circumnavigate intervening obstructions.	Barriers, corridors
2	Outta Here	Exit chamber or dead-end not containing the goal.	Chambers, corridors
2	Probe	Determine the current extent and try to leave it.	—
2	Patchwork	Repair plans.	Bases
2	Other Side	Move robot to the opposite side of the goal.	—
2	Super Quadro	Search for entry into the goal's quadrant or into a new quadrant.	Gates
2	Wander	Move far in an L-shaped path.	Barriers, average task steps, bases, corridors
2	Humpty Dumpty	Seek barriers.	—
3	Adventure	Move to thus far unvisited locations, preferably toward the goal.	Barriers
3	Been There	Discourage returning to a location already visited during this task.	—
3	Chamberlain	Move into a chamber that contains the goal; avoid a chamber that does not.	Chambers
3	Contract	Take large steps when far from the goal, and small steps when close to it.	—
3	Cork	Move into a bottle that contains the goal; avoid a bottle that does not.	Bottles
3	Corner	Move to the end of a pipe that promises a turn.	Corridors
3	Crook	Move to the end of a crooked corridor.	Corridors
3	Cycle Breaker	Stop repeated visits to the same few spots.	—
3	Detour	Move away from barriers that obstruct the goal.	Barriers
3	Done That	Discourage moving in the same direction as before from a previously-visited location.	—
3	Giant Step	If recently confined, take a long step, preferably toward goal.	—
3	Goal Column	Align the robot vertically with the goal, if it is not already.	—
3	Goal Row	Align the robot horizontally with the goal, if it is not already.	—
3	Home Run	Move toward bases.	Bases
3	Hurry	Take big steps early and small steps late.	Average task steps
3	Leap Frog	Execute opportunistic plans.	Bases
3	Mr. Rogers	Move into the neighborhood of the goal.	—
3	Opening	Begin as a previously successful path did.	Openings
3	Plod	Take a one-unit step, preferably toward the goal.	—
3	Quadro	Move into the goal's quadrant or into a new quadrant.	Gates



### Key

-  obstruction  
**G** goal  
**R** robot

Move	Comments (Advisor and strength)	Score
(3, 8)	Giant Step 8, Adventure 6	4
(4, 8)	Giant Step 8, Adventure 6	4
(5, 8)	Adventure 6, Plod 6	2
(6, 3)	Home Run 10, Giant Step 10	10
(6, 4)	Home Run 8, Mr. Rogers 6, Giant Step 10, Adventure 8	12
(6, 5)	Home Run 8, Mr. Rogers 7, Giant Step 10, Adventure 8	13
(6, 6)	<b>Home Run 8, Mr. Rogers 8, Giant Step 10, Adventure 8, Goal Column 10</b>	<b>19</b>
(6, 7)	Mr. Rogers 7, Adventure 8, Plod 8	8
(7, 8)	Mr. Rogers 9, Been There 4, Plod 8	6
(8, 8)	Mr. Rogers 10, Giant Step 10, Been There 4	9

Fig. 9. A state in the midst of problem solving, and how tier 3 votes on the next move. The strengths were converted from  $[0, 10]$  to  $[-5, 5]$  and then summed to produce the scores.

eliminated by No Way. Although every tier-3 Advisor captures a reasonable rationale for navigation, none should be trusted to decide alone. All of them vote together, and the simple ideas behind them support rapid computation. Given 10 seconds, none has ever run out of time on level 10 problems.

Pragmatic navigation is achieved by the execution of FORR's Fig. 3 decision process with all the Advisors of Table 1. Control is delegated to each tier in turn. A single decision may be mandated by tier 1, or a solution fragment submitted and executed from tier 2, or a top-ranked move selected by vote in tier 3.

The nature of the voting process in tier 3 is best explained with an example. Fig. 9 recaps the dilemma at *R3* from Fig. 1, a state in the middle of the fourth trip Ariadne made through this particular maze. The program has, by this time, some useful knowledge about this territory, and forwards it to all the tier-3 Advisors, along with a list of the 10 legal moves. Fig. 9 shows all 30 tier-3 comments for this state, produced by the seven tier-3 Advisors that chose to comment. The best supported move is clearly (6,6), for a variety of reasons: the position was key in a previous problem (Home Run), it is closer to the goal than several other of the legal moves (Mr. Rogers), it is a larger step than several other moves (Giant Step), it has not been visited before during the current task (Adventure), and it aligns the robot with the goal (Goal Column). Although (6,6) is not "on the way" to the goal, once Ariadne repositioned the robot there during this task, Roundabout (a tier-2 Advisor that circumnavigates an intervening obstruction) triggered and drove the robot to (9,9), from which the problem was quickly solved. Fig. 9 demonstrates FORR's thesis: that many "right" reasons may indeed combine to make good decisions. Thus Ariadne does not seek to construct a flawless logical explanation for its decisions, only to make satisficing ones.

For the problem in Fig. 10, Table 2 demonstrates how the satisficing nature of pragmatic navigation can have surprising results. Table 2 is an annotated version of three



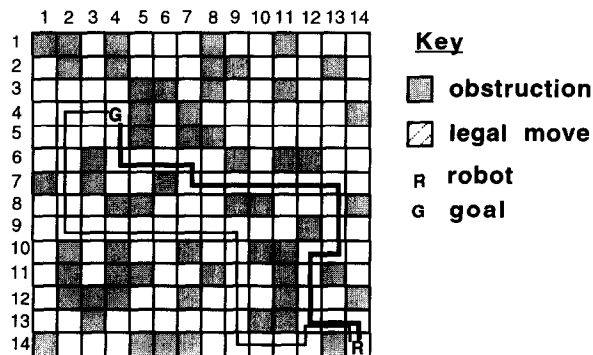


Fig. 10. A level-8 problem.

Table 2

Three solutions to the problem in Fig. 10

Generator's path	Second Ariadne solution	First Ariadne solution
8 decisions	10 decisions	42 decisions
26 path length	22 path length	58 path length
(14, 14)	(14, 14)	(14, 14)
(13, 14)	(13, 14)	(13, 14)
(13, 12)	(13, 12) identical to generator ((12, 12)) Patchwork fragment	(13, 12) (10, 12) (14, 12) (10, 12) a good, but indecisive choice ((10, 13) (7, 13)) Wander tries to help
(14, 12)	(12, 12)	(10, 13)
(14, 9)	(10, 12) a shortcut!	(7, 13)
(9, 9)	(10, 13)	(3, 13) overshooting
(9, 2)	((7, 13) (7, 7) (6, 7) (6, 4) (4, 4)) Patchwork forces a perfect plan	(3, 12) partial correction
(4, 2)	(7, 13)	(4, 12) alignment
(4, 4)	(7, 7)	(4, 8) too far toward the goal
	(6, 7)	... Other Side's fragment to (5, 3), interrupted by Victory at (6, 4)
	(6, 4)	(4, 4)
	(4, 4)	

solutions to Fig. 10's level-8 problem. The first column is the problem generator's solution, the third column is Ariadne's initial try at it, and the second column is Ariadne's solution after 10 different level-8 learning tasks, including this one, in the same maze. In Ariadne's second solution, as soon as the robot moved to (13, 14) it formulated a somewhat baroque plan from the bases it had learned:

((13, 14) (13, 12) (12, 12) (12, 5) (9, 5) (9, 13) (7, 13) (7, 7) (6, 7) (6, 4) (4, 4)).

Ariadne executed the first step of the plan, and then Patchwork triggered at (13, 12). The movement from (12, 12) to (12, 5) is illegal, and Patchwork, in its allotted search time, did not find a patch to the plan, so it forced only the fragment  $\langle (12, 12) \rangle$ . Ariadne moved to (12, 12) and then the ordinary decision-making process resumed. Ariadne forged ahead to (10, 12), then shifted to (10, 13). Patchwork triggered, recognized that the original plan was now fully executable from the newly-accessible (7, 13), and completed the task. The resulting solution entailed more decisions, but produced a shorter path length than the problem generator's solution. (The problem generator's solution is optimal only in number of turns, not in path length.) In contrast, Ariadne's first solution began well, if a bit indecisively. Wander's fragment helped, but the more primitive tier-3 Advisors led the robot to a premature alignment with the goal at (3, 13), an alignment that knowledge helped avoid the second time around.

## 5. Empirical design and results

The data described here was produced when Ariadne generated a maze and tested the performance there of different reasoning agents. Because FORR is nondeterministic, results from 10 runs (i.e., 10 randomly-generated mazes) were averaged to produce an *experiment*. Throughout this section, cited differences are statistically significant at the 95% confidence level unless otherwise stated.

Experiments were performed for problems at some fixed level of difficulty  $n$ . A problem at level  $n$  is generated by selecting a random, unoccupied location for  $R$ . On the first iteration, the algorithm *extends* the robot's location, marking all locations reachable by a single legal move. Each newly marked location becomes an element of the *fringe*. On subsequent iterations, each element of the fringe is extended and removed from the fringe but remains marked, and a new fringe is formed. If any iteration fails, the robot location  $R$  is discarded and the process begins anew. After  $n$  iterations, every element of the fringe is a possible location for  $G$  and one is selected at random and marked. The problem generator's solution is thus the sequence of marked locations. The number of locations that would have been visited by a breadth-first search algorithm is estimated as the total number of locations ever marked in the maze while the problem is generated.

### 5.1. The ablation experiments

The first set of experiments demonstrates Ariadne's ability after learning to solve problems it has never before experienced, and explores which components of the program are necessary to achieve such performance. These experiments were performed on *random mazes*, that is,  $20 \times 20$  mazes that were 30% obstructed, with problem levels 4, 6, 8, and 10. Dimensions and obstruction frequency were selected to provide enough possible problems at the specified level of difficulty. Although the generation of square mazes with some fixed percentage of obstruction has no routines designed to formulate hard problems, it offers ample challenges. Note, in Fig. 10 for example, the lengthy, irregular wall running from (1, 4) to (6, 9), and the deceptive nature of the distance

between (8, 3) and (7, 4), complexities that the problem generator exploits quite nicely. In each maze, the full version of Ariadne was given 20 learning problems ( $R$  and  $G$  pairs) at a fixed level of difficulty. Then 10 newly-generated testing problems for the same maze and level of difficulty were offered to all the agents with learning turned off. After learning, the following agents were tested:

- Ariadne.
- The *Reactive agent*, an ablated version of Ariadne that used only the tier-1 Advisors to simulate correct, reactive decision making alone. If more than one legal move was left after tier 1, this agent made a randomly-selected move.
- The *Reactive+Search agent*, an ablated version of Ariadne that used only the tier-1 and tier-2 Advisors to simulate reactive decision making with situation-based behavior but without heuristic reasoning. If no decision was made after tier 2, this agent made a randomly-selected move.
- The *Reactive+Heuristic agent*, an ablated version of Ariadne that used only the tier-1 and tier-3 Advisors to simulate correct and heuristic reactive decision making without situation-based behavior.
- The *No-Learn agent*, an algorithm that applied all the Advisors but made no useful knowledge available.
- The *No-Plan agent*, an algorithm that applied all the useful knowledge and all the Advisors except those involved with planning (Leap Frog, Home Run, and Patchwork).

Separate experiments also tested a random agent that selected random legal moves (equivalent to blind search), and a best-first agent that did best-first search with the Euclidean distance to the goal as an evaluation function.

The learning problems established a useful knowledge base for those Advisors that depend on it. All agents using such Advisors had equal access to the learned knowledge. A problem of either kind was terminated when the reasoning agent reached the goal or when it reached the decision step limit which included all exploration during tier-2 search. This limit was set to 200 on level 4, 300 on level 6, and 400 on level 8. On level 10, it was set to 1000 to give the agents ample opportunity to solve each problem. The 1000-decision limit permitted more experience, so that Ariadne acquired additional useful knowledge to support better comments.

In preliminary testing on 10 random mazes to determine the efficacy of the random agent, Ariadne was permitted 20 learning trips per maze with a 1000-decision cutoff, and then Ariadne and the random agent were tested on 10 trips per maze with a 200-decision cutoff. Although Ariadne solved 99% of the level-4 testing problems in the 10 randomly-generated mazes, the random agent was only able to solve 36%. In addition, Ariadne's solutions to solved problems were significantly shorter (17.53 as opposed to 271.86) and entailed fewer decisions (23.10 instead of 166.06) than the random agent's. The random agent was therefore eliminated from the full ablation experiment.

Table 3 reports the results for the ablated agents and Ariadne averaged across the 10 runs in each experiment. In Table 3, a "location" is a distinct square in the grid. "Distance" is the Manhattan distance along the path to the goal. Since a step may move through one or more locations, path length varies among problems of the same difficulty. "Decisions" is the number of steps taken during tier-2 search or solution,

Table 3

The performance of Ariadne and ablated versions of it after learning in a particular  $20 \times 20$  maze in Ariadne's world. Results are averaged over runs in 10 mazes

Agent	Distance	Decisions	Moves	Locations	Triggers	Time	Solved
4-step problems (generator path length 12.47)							
Reactive	124.32	75.78	46.41	24.69	—	1.58	81% (0%)
Reactive+Search	31.99	51.07	16.38	15.38	4.64	0.51	96% (0%)
Reactive+Heuristic	47.15	18.66	16.82	9.39	—	1.72	99% (0%)
No-Learn	25.10	53.98	13.01	9.91	3.18	0.78	91% (0%)
No-Plan	16.95	25.14	10.49	9.82	1.65	0.43	98% (0%)
Ariadne	16.38	23.50	10.02	9.36	1.44	0.44	98% (51%)
6-step problems (generator path length 19.19)							
Reactive+Search	53.06	102.60	28.91	24.00	8.78	0.99	88% (0%)
Reactive+Heuristic	62.63	37.48	28.56	13.10	—	3.97	95% (0%)
No-Learn	51.34	119.46	26.98	17.81	7.29	1.59	86% (0%)
No-Plan	30.14	47.98	18.71	17.05	2.61	0.95	99% (0%)
Ariadne	26.36	39.04	15.87	14.58	2.20	0.91	97% (38%)
8-step problems (generator path length 25.62)							
Reactive+Heuristic	115.37	99.14	41.64	19.42	—	18.00	84% (0%)
No-Plan	43.74	73.59	26.72	24.47	5.56	2.22	95% (0%)
Ariadne	41.35	83.89	25.98	23.12	4.59	2.20	93% (14%)
10-step problems (generator path length 31.23)							
No-Plan	96.31	79.68	39.31	24.14	14.94	8.88	98% (0%)
Ariadne	72.48	60.44	35.13	23.70	13.95	5.95	97% (19%)

while “moves” is the number of steps in the solution. The number of distinct locations actually visited during those moves is reported as “locations”. “Triggers” measures the reliance of the system on tier 2; it is the number of passes through Fig. 3 during which any tier-2 Advisor executed. Distance, moves, and locations are computed only over solved problems. (This tends to make the ablated agents look somewhat better than they actually are, because they solve the easier problems.) “Time” is execution time per testing problem, in seconds. The percentage of testing problems solved by Ariadne is also listed, with the number solved as well or better than the problem generator's solution in parentheses.

An ablated agent was eliminated from testing on all subsequent levels if it solved fewer than 90% of the problems within the decision-step limit at any given level. On level 4, the Reactive agent solved a surprising 81% of the testing problems, but with solutions that were significantly longer, entailed many more decisions than Ariadne's, and required far more execution time. The Reactive agent succeeded typically when large portions of the randomly-generated maze were unreachable from the robot's starting point, the way the upper right corner is in Fig. 1. In such a maze the substantial random component of the Reactive agent's behavior was more likely to be effective. Mazes where Pipeline

Table 4  
Percentage of the maze visited by Ariadne  
and by breadth-first search

Level	Breadth-first	Ariadne
4	40.26	3.56
6	64.62	5.58
8	86.25	8.83
10	95.63	9.19

was able to veto many choices also supported this agent well. Nonetheless the Reactive agent was eliminated after level 4.

On level 6, Reactive+Search agent and the No-Learn agent produced significantly longer paths that required many more decisions. (Note the increased number of triggers for No-Learn, as it substitutes search for knowledge.) Both were eliminated after level 6. Despite its long paths, the Reactive+Heuristic agent was retained because it was able to solve 95% of the level-6 problems, but on level 8 its inadequacy became clear. There, its paths were far longer and its solution time substantially greater. On level 10, Ariadne is clearly faster overall than No-Plan, and produces shorter paths for the problems it solves, although once it failed to solve a problem on which No-Plan succeeded.

There is one important benefit of planning. As discussed in Section 4, the problem generator produces a problem whose difficulty is predicated on number of steps rather than path length. In reality, however, it is often possible to find a shorter solution with more turns, and Ariadne frequently does so. Observe that none of the ablated agents ever found a solution as good or better than the problem generator's. In contrast, Ariadne often did so, more than half the time on level 4, 38% of the time on level 6, 14% on level 8, and 19% on level 10. Inspection indicates that most of the rest of Ariadne's solutions are near optimal, with an occasional outlier or two driving up the average distance.

To measure the efficacy of Ariadne's problem solving, compare it with two standard AI techniques: breadth-first search and heuristic search with an evaluation equal to the Euclidean distance from the robot to the goal. Table 4 compares the fraction of the locations accessible to the robot from its starting position visited by breadth-first search with that visited by Ariadne. Ariadne only visits a small fraction of the locations. In contrast, breadth-first search solved the same problems while exploring an increasingly large fraction of the maze. This somewhat understates the cost of a physically executed breadth-first search, whose many repetitive subpaths go uncounted here. By comparison, in a separate experiment of 10 runs where Ariadne first learned on 20 problems, best-first search with only the Euclidean distance to the goal as its evaluation function was able to solve only 26% on level 10 within 1000 steps, and averaged path lengths of 87.65 on these solved problems, versus Ariadne's 62.99 path length with a 93% success rate. Best-first search averaged 164.48 seconds per problem, Ariadne 4.23 seconds.

In summary, as the problems become more difficult, Tables 3 and 4 show that several things happen: breadth-first search reaches an increasing percentage of the accessible unobstructed locations, the search-oriented tier-2 Advisors trigger more frequently, and

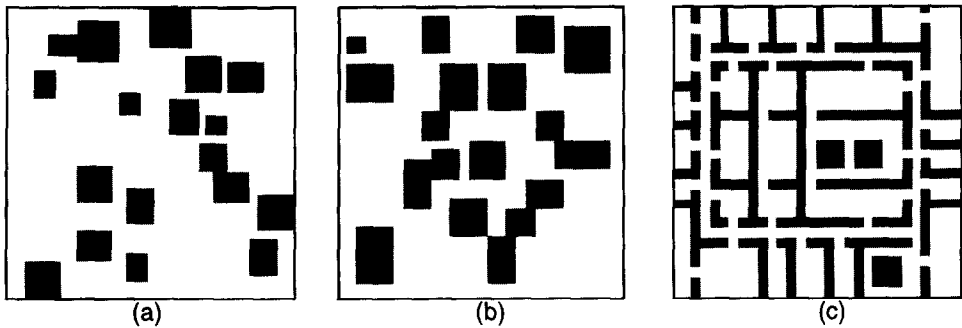


Fig. 11. Some non-random environments: (a) warehouse, (b) furnished room, (c) office. Grids are omitted for clarity.

the ability of the ablated agents to solve the problems becomes markedly inferior. FORR with tier 2 offers a measure of reliability and achievement the other versions lack. Although this work was predicated on the acceptability of suboptimal solutions, the successful paths of the ablated agents are extremely long. With all of FORR's tiers in place, Ariadne gets the robot to the goal more often, more quickly, and considers fewer alternatives along the way.

### 5.2. Performance in non-random environments

Although random mazes present interesting challenges, real navigators face environments which are not random. To test Ariadne's robustness, three other classes of mazes intended to model more realistic worlds were constructed. These maze classes represent furnished rooms, warehouses, and office suites on a  $40 \times 40$  grid. The non-random mazes must be quite large to produce enough even somewhat challenging problems.

The *warehouse* maze models a single room, much like the typical basement, garage, attic, or storeroom. The generator for a warehouse maze places *objects* (rectangular obstructions) at random on an empty grid. Parameters specify the size and number of these objects, the amount of space around them, and the minimum distance an object must be from the outer wall of the grid. Objects may not overlap, but may be contiguous if the parameters permit. An example of a warehouse maze appears in Fig. 11(a).

The *furnished room* maze models a room whose obstructions are objects that do not require a particular regularity. Thus a classroom with regular rows of desks or a room with furniture surrounding a television would not be encompassed by this model. Like the warehouse maze, parameters specify the size of the objects and the amount of space between them. Objects are placed in a furnished room maze in two passes: first along (but not touching) the perimeter, and then in the center section. Although precise locations are chosen at random, the objects along the perimeter are placed sequentially so that there is some measure of balance. Objects along the perimeter are also allowed to overlap. As a result the objects often resemble various pieces of furniture, such as a sectional couch. An example of a furnished room appears in Fig. 11(b).

The *office maze* models a single floor in an office building. Such an environment should make good use of space and make all subspaces (the *offices*) accessible. Although several layouts are possible, the generator here produces only mazes with an outer rectangle of offices (those that could have windows) bordered within by a rectangular hallway. The space framed by the hallway is an inner core of connected offices. Parameters determine the size of the hallway, and the number of offices along each wall, most with a door to the hallway. As in the real world, corner offices are somewhat larger, and offices adjacent to a corner office open only onto the corner office (rather than the hallway) to model private spaces accessible only through adjacent offices. Offices within the inner core are slightly larger than those on the outer rectangle, and have more doors. Office mazes include a few objects placed as if an office were a furnished room. An example of an office maze appears in Fig. 11(c).

Ariadne was developed for random mazes. Without any changes in the useful knowledge or the Advisors, we tested the program in mazes like those in Fig. 11. In each problem class on each run a new maze (e.g., a warehouse or a furnished room) was generated. Ariadne learned on 20 different problems in that maze, and then was tested on 10 previously-unseen problems in the same maze. Level-8 problems were run on the offices and warehouses, but it was difficult to find a furnished room with enough problems at any higher level than 4, presumably because the required gap between the perimeter and the furniture provides ready access to most locations. Although these new mazes were 4 times larger than those in earlier experiments, the decision-step limit remained at 1000. Ariadne solved all the furnished room problems easily, 49% of the time as well or better than the problem generator's solution. It also solved all the warehouse problems readily, 41% as well or better than the problem generator.

The office mazes presented a greater challenge. Only 5% of Ariadne's solutions were as good or better than the problem generator's, and in several cases Ariadne did not solve the test problem within the 1000 decision-step limit. (In contrast, Ariadne devoted an average of 42.83 decisions to furnished room problems, and 84.79 to warehouse problems.) Inspection indicated that in most cases the solution was near at hand, but the corner offices had been sufficiently deceptive to demand additional search. One obvious solution would be to create a representation for linked chambers, where the access point for one chamber is in the extent of the next. Although this is readily programmable, it would not have been appropriate to add it to Ariadne to facilitate solutions in offices. Other than that, this testing in non-random environments appears to be evidence that Ariadne's knowledge representation is adequate for most two-dimensional grid worlds, and that its satisficing approach scales.

## 6. Discussion

### 6.1. Why does *FORR* work?

FORR works because it allocates control appropriately. Control here is whether to react, to search, or to guess intelligently. When a quick and obvious reaction is appropriate, tier 1 responds correctly. Tier 1 prevents foolish errors and guarantees easy right

answers. When pre-identified needs that should be addressed with a particular search routine arise, tier 2 does so. Tier 2 provides appropriate, more thoughtful, more costly responses. Rather than reproduce the safeguards of tier 1 in every tier-2 Advisor, tier 1 monitors tier-2 fragments during their execution, and can interrupt them if necessary. When tier 1 has prevented obvious mistakes and search should not be an option, tier 3 formulates a guess as a compromise among reasonable heuristics. In this way, search is minimized, decisions are timely and well-founded, and no egregious errors are committed.

FORR also works because it responds to context appropriately. Context here is how what one knows affects how and what one decides. A FORR-based system knows the right reasons (Advisors) for doing things in the domain, and how those reasons should relate to one another (their assignment to and sequence in the tiers). A FORR-based system also knows what is worth learning (useful knowledge), how to learn it, and how to apply it (again, the Advisors). Useful knowledge can be represented in many ways, acquired in many ways, and applied in many ways. That flexibility brings to bear what the FORR-based system knows exactly when it needs it. Decisions are based not only on general, prespecified principles but also on information acquired from experience.

It is important to note that the Advisors of Table 1 and the useful knowledge of Section 3 were all developed only for random mazes. The other problem classes are quite recent and no changes of any kind were made to adjust for them. Domain knowledge (here, useful knowledge, Advisors, and learning methods) is a powerful tool. Of course, a one-domain demonstration of an architecture does not entirely validate it. The interested reader is referred to Hoyle, a FORR-based program for learning to play two-person, perfect information, finite-board games [9,10]. To date Hoyle has learned to play 18 different games as well or better than the best human opposition. The games Hoyle plays are relatively simple; none of them has a search space larger than several billion states. People, however, find these games quite challenging and in many ways play them very much like Hoyle [36]. Although Hoyle's Advisors are all reactive (tiers 1 and 3 here), a set of tier-2 Advisors is currently under development.

## 6.2. *Extension to other domains*

How would Ariadne extend to other path-finding domains? Path finding on a printed map would work from correct knowledge and provide the opportunity to reason backward from the goal's location as well as forward from the robot. This suggests additional Advisors that direct the robot's approach with the facilitators and obstructors already in place, such as tier-2 bidirectional planners that would target positions one or two known legal steps from the goal. Path finding in large scale space would probably require a 3-part process: long-distance travel through a network of highways plus two local searches like those currently performed. One local search would join the starting robot location *R* to the chosen highway route; the second would join the chosen highway route to the goal location *G*. The long-distance segment could either rely on standard graph search algorithms or be done in a FORR-based system. Since a highway network is structured to facilitate travel, a FORR-based long-distance solution would require additional useful knowledge (such as highway interchanges) and Advisors to exploit it.



Ariadne imposes the same spatial descriptions of experience on both random grids and more realistic ones. The program learns instances of those descriptions and applies those instances to navigate. Because the random domain is deliberately impoverished, without landmarks or tasks that reflect some regularity (such as a central distribution point or a docking station as a consistent *R*), these descriptions must be very general. Ariadne could, however, readily be augmented to learn other descriptions, such as landmarks. One would only need to add a category of useful knowledge, a method to learn it, and Advisors to reference it. In other words, a richer domain would make this task easier but not invalidate the fundamental approach.

If unlimited computation time were available, if failure were intolerable, or if optimality were essential, FORR's approach would not be appropriate. The right reasons and useful knowledge for many domains, however, are readily accessible, and FORR's modularity facilitates application development. Two substantial FORR-based systems are currently under development. The first, for transportation resource scheduling, assigns trucks and drivers to pick up and deliver about 1000 loads in the course of a work day. Human experts at this task are observed to use a host of tier-3-like heuristics, but we believe that tier-2 Advisors will make a substantial contribution. Initial results with a preliminary version are quite promising. The second system constructs a three-dimensional model of a protein from its formula. Here the Advisors are fewer and more complex, and weight learning (described in Section 6.7) will be particularly important.

### *6.3. How a search space engenders a FORR-based system*

It is possible to reason about the traversal of a search space without complete knowledge of its geography. In this sense, Ariadne's visible space provides a metaphor for state space search. Unintelligent search is blind, just like the ablated version that sought the goal without using Ariadne's limited vision. In many AI artifacts, intelligence has been simulated with an evaluation function that measures proximity to the goal. Deceptive problems foil this approach because the "obvious" evaluation function overlooks subtleties in the problem space. (Recall that "deceptive" was defined here as "when proximity is not a valid indicator of progress". Indeed, the term "deceptive" was borrowed from work in genetic algorithms to address similar difficulties.) The purpose of this section is to suggest some additional ways to think about a search space, and to exploit knowledge about it in a FORR-based system.

The kind of knowledge available about a domain should inspire an Advisor, and also typically determines its tier. To instantiate the ideas behind Ariadne in another domain, the reader should consider what can be learned or observed, at what cost, and how it is best exploited. The experiments described above took great care to monitor the cost of search. The surprise in Ariadne is that in non-random environments like furnished rooms and warehouses it is rarely necessary to plan to reach the goal. For example, there is little point in a tier-2 hill-climbing Advisor for distance to the goal location; the robot often behaves that way anyway, and, when it does not behave that way, there is usually a set of good reasons for that behavior. The reader is therefore encouraged to begin with Brooks' recommendations [4], but to leaven them with learning and situation-

based search. Save search for clearly defined situations with efficient algorithms for ameliorating them.

If you decide to construct a FORR-based system, do so gradually, testing it on a suite of increasingly difficult problems. Begin with tier-1 and tier-3 Advisors, and add Advisors only as the need for them arises. Monitor the Advisors for the resources they require, remembering that tier-1 and tier-3 Advisors should be relatively inexpensive. Monitor too the frequency with which they comment. Irrelevant Advisors will not comment; they are unnecessary and should be quickly dropped. Take pains not to make an Advisor problem dependent, for example, designed for a particular maze; a valid rationale should be broadly applicable. Although the rationales behind tier-3 Advisors need not partition the reasons for good decision making in the domain, try to minimize overlap and to keep each premise simple. This minimizes the possibility of repetitive Advisors, which effectively give one Advisor the ability to vote more strongly than the others, a poor choice.

Individual Advisors often encapsulate what people consider commonsense, such as Mr. Rogers' "get closer" or Giant Step's "take long steps". The kinds of mistakes made during problem solving by a FORR-based system under development typically suggest new Advisors, or corrections to the rationale behind existing ones. It is also possible in some domains to automate the discovery of tier-3 Advisors and gradually phase them into decision making. Pattern-oriented, tier-3 Advisors for game playing have been learned from visual cues [13] and have been shown to improve performance there [14]. These Advisors were based on a limited vocabulary of shapes that was instantiated during extensive play experience and then generalized. Extendibility of the vocabulary becomes a crucial issue here. In a path-finding domain, to learn Advisors one would begin with a vocabulary of spatial terminology (such as "connect" and "constrained") and then postulate Advisors that would combine those terms in useful ways. Such a program could learn to manage corner office suites as a sequence of connected components.

For those attempting to extend the principles of Ariadne or FORR to other applications, the remainder of this section categorizes Ariadne's Advisors by the kind of knowledge they apply to maneuver through the state space of the maze world. Full details on Ariadne's individual Advisors appear in the Appendix by tier; the purpose of this section is to characterize their approach to intelligent search. Many of the Advisors, the reader will note, are actually about search and not restricted the maze world.

#### *6.3.1. Correct conditions as good as shallow search*

A correct condition can be used to generate good single actions instead of searching for them. Rather than examine all the legal actions, an Advisor predicated on a correct condition can generate one or more choices from the problem description and then test to see if that action is among the legal moves. Ariadne's Victory, for example, makes a single move that reaches the goal. Most domains have some intrinsic version of Victory, such as a decision that immediately wins a game. Ariadne's Can Do, as a second example, forces a move to a location vertically or horizontally adjacent to the goal, setting Victory up for the next decision. Some domains have a version of Can Do; for example, in chess Can Do is checkmate, since the actual win occurs when the king is captured. Victory does not look at all possible next states to see if the robot reaches

the goal in one of them, nor does Can Do look two moves away in the search space. Instead Victory interprets visibility as accessibility, and seeks the goal-achieving move among its legal options. Similarly, Can Do interprets adjacency as near-accessibility, and seeks a one-from-the-goal move among its legal options. An Advisor that correctly exploits a condition as good as shallow search assumes that a chosen operator will execute properly, that is, no other agent or aspect of the environment will prevent the Advisor from achieving its purported post-condition. An Advisor that correctly exploits a condition as good as shallow search belongs in tier 1, where it offers quick and easy access to nearby solutions.

### 6.3.2. Constrained subspaces and hindrances

A constrained subspace can be used to curtail unnecessarily repetitive search. Informally, a *constrained subspace* is a set of states in a state space that afford ready access to each other, and very little access to other portions of the space. In the maze world, a chamber, a bottle, and a corridor represent constrained subspaces. In other, less visually-oriented domains, a constrained subspace would be a set of states whose neighbors are primarily (but not exclusively) themselves, so that search would cycle among them. An example of a constrained subspace in another domain is a set of game states where two players shuttle the same piece or two between a small set of locations. No intelligent agent would deliberately search within a constrained subspace unless it believed that a goal state lay there; it would seek an operator that left the constrained subspace instead.

More formally, let  $S$  be a set of states in a state space and let  $S_s$  denote the set of immediate neighbors of  $s \in S$ , those accessible by a single operation on  $s$ . Let an *exit* from  $S$  be a state  $s \in S$  such that  $S_s - S \neq \emptyset$ , that is, a state not all of whose neighbors lie in  $S$ . A subspace  $S$  is constrained if and only if it has relatively few exits proportional to  $|S|$ . By definition, Ariadne's constrained subspaces have at least one exit, such as the exit from a dead-end or the access point of a chamber. Bear in mind, however, that visible space in Ariadne is only a useful metaphor for this concept; constrained subspaces can occur in any domain.

Learning the extent of a constrained subspace often requires search. Thus Ariadne has a tier-2 (search-based) Advisor called Probe that delineates what it perceives to be a chamber, a constrained subspace with at least one exit, its access point. Such search is appropriately triggered when the agent senses confinement in the search space, and terminated when an exit is identified. Confinement is measured in Ariadne by how little of the territory the robot can see or how few distinct locations it has been to recently. It is also possible to induce a confined space with a post-solution critique of the path through the search space. This is the essence of the learning algorithm for bottles.

Once a constrained subspace and its exit(s) are identified, an Advisor can exploit it by refusal to move through an exit into a constrained subspace that does not contain a goal state. In Ariadne, for example, an intelligent agent has no reason to enter a dead-end nor to pause in a pipe not believed to contain the goal. Unless the goal is in the pipe, locations within the pipe cannot ever provide a new option, and can only increase the number of steps in any solution. This is why Ariadne treats a pipe as a set of locations to be avoided. Advisors that prune search with knowledge about constrained

subspaces can appear in any tier. In tier 1, No Way keeps the robot from entering a dead-end, and Pipeline keeps it from pausing in a pipe. Since an agent may begin in a constrained subspace, or heuristically stumble into one, Advisors to leave a constrained subspace that does not contain a goal state are also necessary. This is what Outta Here does, for example, searching for an exit from a chamber or dead-end in tier 2, and what Chamberlain and Cork do in tier 3 by their reactions to knowledge about chambers and bottles, respectively.

Yet another kind of knowledge that prunes search is a *hindrance*, a set of states in a state space which share a property whose presence guarantees that the goal is not achieved in the state. In the maze world, the presence of a barrier between the robot and the goal constitutes a hindrance. An example of a hindrance in another domain is the inability to move a particular playing piece to a particular place on a game board, such as the king's location in chess. In any domain, search may visit states with hindrances until quite close to the goal, but the elimination of a particular hindrance can be quite productive. Since all the hindrances that can arise during problem solving may not be predictable from the initial state, an intelligent agent cannot merely decompose the problem into a set of subgoals that are non-hindrance conditions. Ariadne, for example, does not know where all the barriers lie. An intelligent agent can address particular hindrances as they arise, either reactively, or with situation-based search from the current state, or with situation-based search for plan revision, or even by deliberately seeking out hindrances and then avoiding them. Ariadne's Detour, Roundabout, Patchwork, and Humpty Dumpty are examples of these approaches, respectively. Because knowledge about hindrances is likely to be heuristic, Advisors that rely on them reside in tier 2 or tier 3.

### 6.3.3. Transition regions

Although many of us are accustomed to imagining state space as a vast and somewhat formless graph, it may well have a shape that suggests solutions. A *transition region* is a set of locations that permits movement from one portion of a space to another. Ariadne represents its search space in quadrants, which suggests transition regions between them. Fig. 12 identifies transition regions *A* through *N* among the quadrants for the maze from Fig. 1. For clarity, the horizontal and vertical pairs of transition regions are shown separately, in Figs. 12(a) and 12(b), respectively. A location is in a transition region if a single decision from it would move the robot to another quadrant. For example, from any of  $E = \{(6, 3), (6, 4), (6, 5)\}$  in quadrant 3 the robot could move east to region *F* in quadrant 4 in one step.  $Q_i$  denotes the portion of quadrant *i* not in a transition region.

The summary graph in Fig. 12(c) links two transition regions if they have a location in common or if a single decision could shuttle the robot between them. The summary graph makes it quite clear, for example, that travel to regions *B* or *N* is of limited value, and that travel from *N* to the rest of the fourth quadrant ( $Q_4$ ) will require movement through region *D* in the first quadrant. Because the summary graph can substantially reduce the number of nodes in the search space (from 64 to 17 in this case), it is a powerful planning tool. Its construction, however, requires exhaustive search through the state space. Instead, a single representative of a transition region can expedite movement

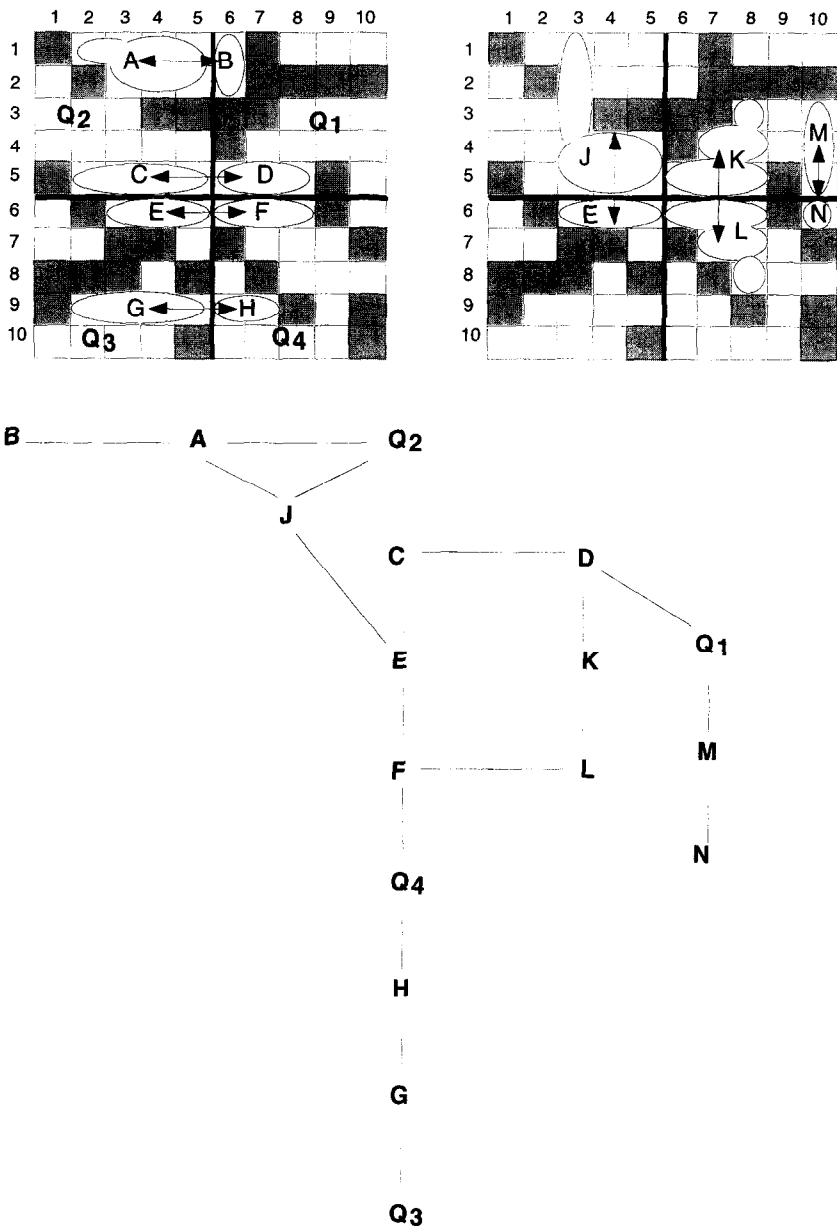


Fig. 12. The horizontal (a) and vertical (b) pairs of transition regions. (c) The summary graph for the same maze

through the state space. Ariadne's gates are a heuristic approximation of the summary graph for a maze. For example, learning (5,7) as a gate in Fig. 12 would describe the relationship among regions *C*, *D*, and *F*, as well as how to reach (5,7) through the overlap of its extent with *Q*<sub>1</sub> and *Q*<sub>4</sub>. Thus one way to approach a problem in the

maze world would be to search for an intervening transition region, the way  $Q_1$  must precede movement from  $N$ ; Quadro is a reactive version. A tier-2 planning Advisor for movement through a sequence of quadrants would also be a reasonable idea, one not implemented here.

#### 6.3.4. Other classifications of (search) space

It is also possible to classify a state space according to a set of conditions that characterize the relationship between the states visited thus far during search and the goal state. If search has not been productive, it can be constructive to search for a state whose classification is different. In Ariadne, for example, a tier-2 Advisor called Other Side monitors the robot's position in space with respect to the goal. If the robot has remained, say, consistently south of the goal, Other Side may devote resources to moving it north of the goal. Wander is a tier-2 Ariadne Advisor whose classification mechanism is based on the notion of moving away from a small set of states. Advisors like these, that use knowledge derived from costly search procedures, should be triggered only when there is some likelihood of their success, and assigned to tier 2.

When decisions are primarily heuristic, the same superficially “right” reasons may repeatedly draw the agent to the same non-productive states. Therefore, in a satisficing approach like FORR, more general classifications describing what the agent has decided in the past can be quite useful. Advisors with such a historical perspective are typically in tier 3. Ariadne's Been There, Done That, Cycle Breaker, and Adventure have counterparts in many domains. Been There discourages revisiting a state; Done That discourages repeating the same transition out of a state. Cycle Breaker is a tier-3 Advisor to suggest alternatives that are different in the current context, while Adventure suggests states entirely new in the current task. A similar, exploratory heuristic has been applied to Go [32].

It is also possible for an Advisor to take a historical perspective on a set of tasks. Ariadne's Opening supports the reuse of the beginning of a previously successful path. It has a counterpart in most domains, most obviously for openings in game playing. Home Run and Leap Frog are both dependent on bases, historically useful generalizations over states in the space. (A base is a generalization because it ignores the location of the goal.)

Still other classifications compare alternative actions and select extremes. Ariadne's Giant Step and Plod, for example, advocate extreme step size, the former large steps and the latter small ones. A similar heuristic was employed in AM and Eurisko [29,30].

#### 6.3.5. The role of reactivity

Reactivity is an important component in both Ariadne and FORR. Inexpensive sensing and simplistic reasoning can actually combine to produce a remarkable number of fast and wise decisions. Although the ablation experiment proved that these were not enough to solve the most difficult problems efficiently, they did prove to be an important component. Advisors that favor simple environment sensors are typically reactive heuristics for tier 3, as are all such Advisors in Ariadne: Goal Row and Goal Column concern alignment with the goal, Mr. Rogers and Contract react to distance from the goal, Corner and Crook react to what is visible.

#### 6.4. Cognitive plausibility

There is no claim here that Ariadne is a cognitive model of a human navigator, only that many of its features are cognitively plausible. Several of the Advisors do model principles of naive geographic reasoning [8] all of us readily recognize: Plod's tentativeness, Adventure's curiosity, and Hurry's anxiety. In addition, cognitive scientists have found empirical evidence for Contract's rationale [18] and the rationales of several of the other Advisors [20]. A literature search, however, reveals no tests of human subjects on problems as difficult as these.

The integration of reactivity, heuristics, and search as situation-based behavior is based upon the detection of all three approaches in people [23]. There is increasing evidence that FORR, Ariadne's underlying architecture, accurately captures aspects of human problem solving. In a variety of domains, from circuit design to game playing, psychologists report that people integrate multiple, parallel, possibly conflicting strategies to make decisions [2,6,37]. During problem solving, people describe multiple, possibly inconsistent, rationales relevant to a single goal [36]. The brain appears to use a modular architecture to accomplish integration of information [7,46].

The use of a learning architecture, rather than a prespecified one, is supported by evidence that people evolve superior performance. Human expertise develops only from repeated experience at problem solving [15–17,33]. This expertise is applicable to a related set of problem classes. For example, an expert path finder in unfamiliar territory will immediately wonder about dead-ends and not about the color of obstructions. Experts rely upon a foundation of domain knowledge for a source of focus and direction to provide a baseline level of competence. Thus the architecture need not begin with total ignorance; it is reasonable to provide it with general domain knowledge and the methods to specialize it, for example, by learning feature instances.

Ariadne's facilitators and obstructors form its *cognitive map*, its representation of its world. These features are consistent with the ways humans represent space. People use constructed representations of the visual world to make inferences about space, representations that are based upon, but more abstract and general than, perception [44]. These representations systematically distort visual perception to facilitate recall [45]. They are integrated with many other kinds of information to form a model that the human user does not require to be complete or consistent. Ariadne's reliance upon multiple representations and its tolerance for inconsistent and even incorrect information, geographers tell us, is much like the naive geography that people rely on [8]. Even the use of levels (number of turns) for degree of problem difficulty is supported by results that people gauge distance that way [38].

There are two ways to view Ariadne's task as resource-limited. If CPU time is a scarce resource, then the agent that makes the fewest passes through FORR's decision structure in Fig. 3 is best. If traveling time or fuel is a scarce resource, then the agent that constructs the shortest paths is best. On both metrics the full FORR agent achieves a synergy that the ablated versions lack. The behavior of human subjects asked to traverse a variety of routes on a campus was also explained by distance, time, and number of turns (here, problem level) [18].

### 6.5. *Situation-based behavior as a compromise on search*

The initial impulse behind reactive programming was to avoid search, to make instead decisions that were local in time (unconfirmed by search) and local in space (restricted to current perception). When one augments the reactive Advisors of tier 1 and tier 3 with tier 2, it is easy to forget that. Tier 2 Advisors are kept within the search minimization philosophy in two ways. First, FORR only allocates each Advisor, in any tier, a limited amount of computing time. Solution fragments that take too long to construct will not arise. Second, tier-2 Advisors have hand-coded routines intended to address their particular subgoals. These routines generate and test solution fragments, but the proposed partial solutions are highly constrained, to address the situation identified by the trigger while preserving resources. This constraint saves the tier-2 Advisor from a combinatoric explosion. For example, as detailed in the Appendix, Roundabout's search is quite deep, but it is also severely curtailed by knowledge; that is why it is effective.

As discussed in Section 2, tier-2 Advisors are intended to simulate behavior observed by psychologists studying time-limited decision making [23]. The decision makers described how they first recognized a particular category of situation (in Ariadne, the trigger), and then applied a procedure tailored to that situation type (in Ariadne, the search) that proposed a kind of solution deemed appropriate to that situation category. In FORR's interpretation, such situation-based behavior is not case-based reasoning (CBR), although they have much in common. In CBR, experiences are indexed and stored. Later, one or more potentially relevant cases are retrieved, and an attempt is made to modify their solutions to solve the current problem [24]. Although situation-based behavior is triggered by an abstraction of the current state that could have been used as an index for CBR, situation-based behavior does not retrieve specific solutions to be modified, only procedures intended to generate solution fragments. Situation-based behavior and CBR both constrain solution generation, but CBR does it by searching from old solutions, while situation-based behavior does it by the knowledge inherent in its procedures. Klein and Calderwood emphasize that the human experts they study do not perceive their problem solving as reminding. (This is not a claim that CBR has no parallel in people, only that it is less likely to be used when resources are very limited.)

Situation-based behavior is not the same as planning either. A plan is a set of actions intended to reach a specific goal [43]. The Advisors of tier 2 are not planners because they actually execute their behavior, even if they do not eventually recommend it. For example, Wander can investigate as many as eight L-shaped paths (one longest step in each direction plus a possible second step) before it chooses one to execute. Rather than planners, situation-based Advisors are procedures that reactively seize control of a FORR-based program's resources for a fixed period of time. When that time elapses, the situation-based Advisor either returns control to tier 3 or returns a sequence of actions whose execution it requires. Tier 3 constitutes a reactive decision maker, much like Pengi [1]. The principal difference is that Pengi's problem is living in its world; it is not held to an explicit decision standard like "solved in 1000 decision steps".



Situation-based behavior is a resource-grabbing, heuristic digression intended to produce a solution fragment, not a production rule or a macro-operator. Although the trigger of a tier-2 Advisor could be the condition of a production rule, the comment generator's response is too complex (particularly since it can be overridden by tier 1) to be the action part. A macro-operator is a generalization across the variables entailed in a successful procedure, whereas a situation-based Advisor is a procedural generalization over several kinds of behavior appropriate to a situation.

Finally, situation-based behavior sheds some light on the ongoing debate about representation and reactivity [22]. Ariadne's conceptual knowledge includes "the last 30% of the moves have been in no more than 5% of the locations in the maze" and "a wall lies between the aligned robot and the goal". This paper demonstrates that, at least in this domain, the representation of conceptual knowledge as a context is an essential component in a reactive learner.

#### *6.6. The interaction among reactivity, heuristics, and search*

Ablation shows the interdependence among reactivity, heuristics, and situation-based behaviors. Without tier 2, FORR is a reactive system augmented by learned useful knowledge. The results with the Reactive+Heuristic agent, however, simply are not good enough. This agent regularly gets stuck in regions where Giant Step cannot extricate it; it needs maneuvers like Wander's L-shaped path to get out. It also regularly gets close to the goal but cannot reach it because of an intervening wall; it needs Roundabout's determined circumnavigation to get closer. The situation-based Advisors of tier 2 make a clear contribution when combined with tier 1 as Reactive+Search, but they have a limited repertoire of behaviors. The situation-based Advisors are insufficient on their own. They trigger significantly more often with Reactive+Search than with Ariadne, because most of their triggers measure lack of recent progress, something the robot experiences more often with Reactive+Search. Tier 2 is, effectively, a device to execute subgoals. The subgoal is the opposite of the trigger, for example, Wander tries to "get out of here", and Roundabout tries to "get around the wall". Subgoals are detected by the program, but their nature is predetermined by the programmer.

There is a complex relationship among the tiers. Tier 1 offers the commonsense inherent in any problem-solving task. Tier 3 tries to avoid search and effectively sets up the situation-based Advisors in tier 2 so that they can trigger. For example, Goal Row and Goal Column move the robot where Roundabout can trigger. In turn, the situation-based Advisors of tier 2 set up the heuristic reasoners in tier 3. For example, Wander puts the robot where many tier-3 Advisors are more likely to make newly constructive comments. Another important side-effect of the search in tier 2 is the acquisition of useful knowledge, from which Advisors in every tier can benefit.

#### *6.7. The role of learning in pragmatic navigation*

As the data indicate, learning is essential to an agent facing hard problems with limited resources. The ablated No-Learn agent failed to solve many of the hardest

problems, while the full version could perform quite well after only 20 learning trials. Even on level 6, No-Learn was slower to solve problems than Ariadne because its lack of knowledge forced it to meander more about the maze.

One hallmark of Ariadne's pragmatic navigation, as Table 1 indicates, is the variety of procedures for the same useful knowledge. Corridors demonstrate the flexible use of knowledge; the same data in a single representation is applied by seven different Advisors. Barriers demonstrate the flexible acquisition of knowledge; the experience of four different Advisors provides fodder for learning them.

Nor is there apparently any danger of learning so much useful knowledge that a detailed, static map of the maze would have been a more efficient representation than Ariadne's learned heuristic features. After 20 level-8 problems in  $20 \times 20$  random mazes, for example, there were on average only 32.5 barriers, 80.5 bases, 1.6 bottles, 3.9 chambers, 49.3 corridors, and 18.0 gates. As a graph, however, such a maze would have 280 nodes and approximately 1485 edges.

Perhaps the most interesting challenge is that of balancing the Advisors' comments. Even among randomly-generated mazes there are different kinds. Some may conceal the goal behind wall-like structures, others place it at the end of a tortuous path. When one compares in Table 3 the number of decisions used to solve a problem with the number of triggers, it is clear that most of Ariadne's decisions were made in tier 3, and that the program's overall performance could still be improved. The key to this, we believe, is to have the program learn to value tier-3 Advisors appropriately. AWL is an algorithm that FORR uses to learn weights to apply to the voting in tier 3 [11]. AWL, however, was formulated for a domain (game playing) where the learner necessarily has access to a model of expertise (its opposition). As described here, Ariadne is an autonomous learner. Although initial testing indicated that weights trained on the problem generator's paths would improve performance, we are currently adapting AWL to retain Ariadne's autonomy.

#### *6.8. The role of planning in pragmatic navigation*

The simple planning in Ariadne is a surprisingly effective tool. Although the plans themselves are naive, they are inexpensive to generate. The three plan-related Advisors (Home Run, Leapfrog, and Patchwork) together average only 0.1517 seconds of computation per problem during level-10 testing. On average Ariadne constructs 14.58 (not necessarily distinct) plans for a level-10 problem during testing, from an average of 77.0 available bases. Because bases are strengthened with each re-identification, and three different Advisors use them, over time Ariadne develops habitual routes, just the way people do. It is not uncommon during testing, or even late in the learning stage, to see Ariadne formulate a reasonable plan fairly early and execute it in its entirety. This also means that, unless some fortuitous choice occurs, the program is unlikely to develop shortcuts on its own; those would have to be taught.

Because a plan in Ariadne is a sequence of positions from the robot's current location to the goal, a plan is not reactive in Schoppers' sense [39]. Ariadne's planning is, instead, a reaction to the fact that it currently has no applicable plan and that it has the knowledge (the bases) from which to attempt to construct one. The preference for

planning from strategic locations (bases) instead of considering all possible choices is similar to the approach in [40]. Ariadne's plans are deliberately structured to support legal travel (their orthogonal moves) and to be readily patched by a circumnavigation routine. Using bases to plan also makes the overly optimistic assumptions that Ariadne's routes are correct and that the robot has had adequate travel experience throughout the maze. This is why not too much time is invested in plan construction or correction; it is generally more efficient to replan.

The component ripe for development in Ariadne is planning. No Advisor considers barriers, gates, or corridors in plan construction, or links chambers and bottles together to facilitate travel into the corner offices of the office mazes. An Advisor that reasoned efficiently with these useful knowledge items could make a significant contribution, as long as it was appropriately wary of the heuristic nature of the information.

### 6.9. *The real world*

Although Ariadne would require adaptation before it could direct a real-world robot, pragmatic navigation embodies principles that roboticists would do well to consider. When territory is distant (a planet), dangerous (deep sea terrain), or dynamic (a warehouse), correct and complete maps may be unavailable and a feature orientation becomes quite practical. The nature of the maze already permits irregularly-shaped territory, since the mazes may have unreachable sections along their edges, as in Fig. 1. For the real world, the fineness of the grid would also have to be determined. Given the heuristic nature of useful knowledge and the inaccuracies of measuring devices and robotic controls, however, one really does not want a very fine grid: approximations in pragmatic navigation are tolerable and tolerated.

The primary issue in adaptation would be the sensors. Their inaccuracies would require modifications to allow a margin of error. Continuous, rather than discrete, sensing would be more costly. Permitting the robot to sense and travel in only four orthogonal directions was more limiting than the equipment on most robots, but not far from human proclivities [19]. The equidistant placement of the four sensors was judged appropriate for a task where forward motion was often no more productive than lateral or backward movement. A more generous number of sensors and directions should do better, although it could require increased computation and would necessitate revision of some of the algorithms.

Finally, in many real-world problems the goal is not completely stationary; it may drift or attempt avoidance maneuvers. Standard AI search methods have no provision for this. No real difficulty is foreseen if  $G$  were to move, say, one unit in a randomly chosen direction after each two of Ariadne's moves. The instability of  $G$  should be a nice fit with the opportunistic planner.

## 7. Related work

This work has some clear counterparts with Korf's analysis of heuristic search in the tile puzzles [25]. His minimin search "strategy of least commitment" is shared by

Ariadne's Plod, but it of necessity lengthens the number of steps in a solution. For example, if the correct, unobstructed move is from (1, 1) to (1, 10), plodding will take 9 moves to get there, although an immediate move to (1, 10) would be legal. Ariadne's Mr. Rogers uses the Euclidean distance heuristic much the way Korf tried heuristic node ordering in the tile puzzles, and with the same disappointing results. Once you get close in this domain, too, there may be better ways to reach the goal. Korf's permission to backtrack with loop prevention is analogous to some of the decision making in Roundabout. Ariadne has no foolproof loop prevention, but Been There, Cycle Breaker, and Done That discourage loops. In Ariadne's graph (rather than tree) search space, Korf indicates that one cannot expect locally optimal solutions. If the search horizon were limited only by how far the robot could see ahead, the Reactive+Heuristic agent should solve few problems, since it sees only one step ahead. That agent's better than expected performance is attributable to its useful knowledge about a particular maze and its general maze-traveling knowledge in the tier-3 Advisors. It is possible to dictate the level of difficulty in Ariadne's problems, but with the tile puzzles there remains some uncertainty about how the level of difficulty impacts upon the ability of the problem solver.

Although Ariadne's maze problems may be reminiscent of recent machine learning work in reinforcement learning, it is important to note that the program's task and fundamental approach are significantly different [31,42]. Such programs seek convergence to an optimal path through repeated solution of what, according to the definition in Section 1, would be a single problem. In contrast, Ariadne has no mechanism that would guarantee optimality, and will quickly settle upon the same route in most cases. Ariadne constructs satisficing paths for a set of problems, applying knowledge learned from one problem to the others, instead of from one problem-solving attempt to another attempt at the same problem. The complexity of a maze problem for the reinforcement learners is a function of both goal strength and the number of state-action pairs (the number of reachable locations and directed one-step movements from them). The complexity of a problem for Ariadne, on the other hand, is the number of turns required, independent of the size of the maze and the strength of the goal. Memory use is different, too. Ariadne learns abstractions, while the reinforcement learners refine estimates for the value of each one-step move attempted from each state.

A second, recently suggested learning approach was a case-based planning method for the grid world that operated in a set of abstraction spaces, and stored both detailed and abstracted solution paths [3]. These mazes were somewhat simpler versions of Ariadne's warehouses, which present few dead-ends, narrow-necked chambers or bottles, or effective barriers between large regions. Once again, Ariadne would find them relatively easy.

One way to characterize an approach to learning navigation for a robot is by the degree to which it is engineered, that is, to which the robot's abilities are preprogrammed. Work in this area ranges from the *tabula rasa* approach in [34] to the thoroughly prescribed [5]. Although Ariadne's initial knowledge about what to learn and how to learn it is certainly engineered, conflicts among knowledge-based principles are left to the voting in tier 3, that is, no tier-3 Advisor is deliberately formulated to outweigh any other.

A second way to characterize an approach to learning navigation for a robot is by the degree to which it reflects what is known about human perception and behavior. PLAN is a connectionist model that integrates path finding into general cognition [5]. PLAN learns a topological map of experienced landmarks, a route map of place sequences and directions between them, and a survey map of global information. All three maps are based on a visual scene rather than an aerial view. The resultant system has strong biases to human sensory orientation (particularly in its refusal to see more than 180° about its position) and limited short-term memory, neither of which is necessary for intelligent robot navigation. The authors claim that such maps are therefore based on experience, but the experience they record is purely visual or rote experience. In contrast, Ariadne retains knowledge that represents both momentary experience (e.g., a gate) and reasoning about a sequence of experiences (e.g., a base). Although developed independently, some of PLAN's features are quite similar to Ariadne's: its gateways are similar to Ariadne's gates, its regions similar to Ariadne's bottles and chambers. PLAN makes its global overview explicit, and plans hierarchically from regional maps. PLAN's expertise, however, appears to be in maps whose scale and level of difficulty are far below those Ariadne traverses with ease, and the authors offer no statistics on either its efficiency or its effectiveness.

A third way to characterize an approach to learning navigation for a robot is by its representation of its world. Hayes has constructed a rigorous approach to reasoning about space [21]. He too envisions representations of space into pieces (like Ariadne's bottles, chambers, and quadrants) that have boundaries and connectors (like Ariadne's bottlenecks, access points, and gates). There is, as yet, no implemented version, however. TOUR [27] and Qualnav [28] have landmarks that are sensorily distinctive. Ariadne's grid world, as originally postulated by Korf, did not provide such landmarks. As a substitute, Ariadne has gates and bases predicated upon theories about what might be useful in travel. Ariadne does not store routes or route fragments from completed problems either, whereas TOUR and PLAN both keep a topological network of routes between places that can be manipulated to find a path. Unlike these systems, Ariadne's global overview is implicit in the useful knowledge it learns; it plans naively and opportunistically. Although it would be simple enough to learn a route-fragment graph on Ariadne's gates and bases, the reactive approach described here is preferred, for both its computational efficiency and its limited storage.

SSH (Spatial Semantic Hierarchy) is a four-level ontological hierarchy to provide spatial knowledge for real-world robots [26]. Ariadne's features lie primarily at SSH's topological level as learned places (e.g., gates and bases) and regions (e.g., bottles and chambers). Kuipers envisions this level as a representation for exploration and path-finding problems. Ariadne is thus complementary to his work.

There has been some work on learning heuristics for problem solving. Lenat's Eurisko sought transformations of existing heuristics to find new, useful ones [30]. Such an approach affords access only to interesting concepts linked to others by some attractive path of available transformations. Since transformation is a syntactic process, this approach does best when the representation in some way reflects the semantic content of the concepts, as it did in AM [29] but less so in some of the domains Eurisko addressed. Prieditis' Absolver II sought an abstraction of a problem that could

be sped up and then used as an admissible heuristic [35]. Absolver II removed details from the problem description in a variety of grammatical transformations that also required a good problem representation. The heuristics it learned were intended to prune search, but to be used one at a time. One of the problems Absolver II addressed was a “Rooms World” problem for which it computed a heuristic that minimized the number of rooms a robot visited in the course of a box moving task. This is analogous to entering chambers only when necessary, for which Ariadne relies on Chamberlain and Outta Here. In contrast, Ariadne’s heuristics are for the most part prespecified, although the information on which they rely is gathered during problem solving. Although this makes them applicable to a variety of problems and problem classes, it also makes them vulnerable to incorrect generalizations induced from experience.

Finally, although optimum paths in space have a variety of well-documented algorithms [41], they eventually face the  $O(n^2)$  complexity that comes from the underlying graph among the  $n$  points in the space. If Ariadne were to construct a map based on its heuristic knowledge, search there would be  $O(n^2)$  too. Instead, Ariadne employs the kind of spatial descriptions of experience people formulate in a satisficing approach to subvert that cost. The 24 Advisors in tiers 1 and 3 must react to a finite set of at most  $2n - 1$  legal moves and the available useful knowledge  $u$  within some fixed time  $t$ . The 8 Advisors in tier 2 spend at most some finite time  $t'$  in search. For a problem with a solution step limit  $l$ , this means that Ariadne spends at most  $l(24(2n - 1)ut + 8t')$  time in search of a solution. The key is to control  $u$ , that is, to keep the cost of referencing useful knowledge (facilitators and obstructors) sublinear. The result is occasionally optimal, and usually quite good, performance.

## 8. Conclusion

Human navigators react quickly and correctly to clearly productive or unhelpful opportunities, such as “there’s the goal” or “not that dead-end again”. They entertain a variety of heuristics, such as “closer is better” or “when far away, take a long, straight step”. They also digress into a search mode tailored for a particular situation, for example, “this obstacle is between me and the goal, so I have to get around it”.

Ariadne is an implemented pragmatic navigation system that epitomizes this kind of naive geographic reasoning, as if it were learning the way around a new campus or town. Ariadne learns a variety of features about a new environment, and uses that knowledge to solve multiple travel tasks there. Ariadne solves multiple problems in the same territory. It solves the easier ones in fewer steps and with fewer resources than more difficult problems. Compared to traditional search techniques, it solves these problems quickly, as measured in elapsed problem-solving time, number of decisions, and path length. It also performs efficiently, as measured by number of distinct locations visited and the percentage of repeated locations in a path. And Ariadne learns, so that it applies previous experience to hitherto unseen problems, both in random mazes and in a variety of more realistic world models. Ariadne does some primitive planning, and controls the allocation of search resources to acquire knowledge.

Pragmatic navigation as detailed here was shaped by an underlying architecture, FORR, a robust, adaptive integration of reactivity, heuristics, and search. Instead of a detailed map, pragmatic navigation represents the world as a collection of probably correct, possibly useful features. Instead of a uniform learning technique, each feature has one or more learning methods to acquire territory-specific instances of it. Instead of executing a long-range plan, pragmatic navigation selects one reasonable part of a journey at a time. Instead of planning with heavy search costs in a fine-grained representation, pragmatic navigation opportunistically constructs naive plans that suggest, rather than mandate, path selection. Despite the heuristic nature of Ariadne's knowledge and decision making, empirical results demonstrate that, after relatively few learning tasks, Ariadne is able to solve novel, difficult problems quickly.

Ariadne's clear ability to outperform ablated versions of itself demonstrates that no subset of FORR's reasoning methods suffices in this domain. Reactivity, heuristics, learning, planning, and time-limited search based on situation recognition are each essential to good performance, and FORR connects them all in an appropriate manner. This paper advocates pragmatic navigation as a satisficing, cognitively plausible, developmental approach to path finding. Ariadne demonstrates that such an approach is viable, efficient, robust, and applicable to a broad range of environments.

## Acknowledgments

David Sullivan's preliminary work on Tonka convinced me that a FORR-based version of Ariadne could be a success. An earlier, far more primitive, version of Ariadne was reported upon in [12]. Since then, Barry Schiffman provided much insight and many constructive suggestions, as well as the generators for mazes that represent furnished rooms, offices, and warehouses. Thanks too to Mike Pazzani for the suggestion to compare Ariadne with best-first search, and to Jack Gelfand, Alice Greenwood, Pat Hayes, Ben Kuipers, and Barbara Tversky for their shared wisdom.

## Appendix A. Ariadne's Advisors by tier

### A.1. Tier 1

Advisors appear in the order in which they are consulted.

*Victory* has absolute authority; if the goal is reachable by a legal move, Victory makes it.

*Can Do* forces a move to a location vertically or horizontally adjacent to the goal. On the next decision cycle, Victory will turn toward the goal and reach it.

*No Way* vetoes unnecessary moves into a dead-end. This Advisor checks each legal move to see if it resides in the extent of a dead-end that could not contain the goal. (Recall that the extent is a bounding rectangle, so this is a conservative approach.) If so, No Way eliminates the move from further consideration by any other Advisor, unless the robot is already in the dead-end and therefore needs to get out.

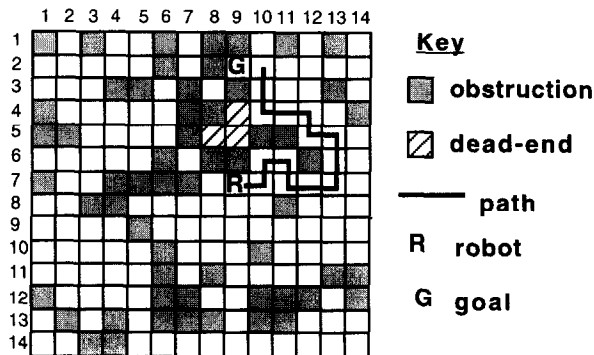


Fig. A.1. Roundabout circumnavigates an obstruction.

*Pipeline* vetoes a move to any location within a pipe if the location beyond the opposite exit is in view, unless the robot is in the pipe or the location is aligned with the goal. For example, in Fig. 6 (see p. 285) from (14, 9) Pipeline would veto moves to (14, 10) and (14, 11); if traveling east efficiently, one would go through that pipe, not pause within it.

## A.2. Tier 2

Advisors are discussed in the order in which they are consulted.

*Roundabout* is a tier-2 Advisor that attempts to circumnavigate an obstruction between the robot and the goal. It triggers when the robot is in the same row or column as the goal but there is an intervening obstruction. When the robot is so aligned, Roundabout attempts to go around the obstruction between it and the goal. When, for example, Ariadne faced the situation in Fig. A.1, Roundabout took the robot around the intervening horizontal obstruction before stopping at (2, 10) where the goal was in sight. Roundabout is not a traditional wall-following algorithm; it establishes a primary direction (toward the goal) and a secondary direction (orthogonal to the primary and not toward a barrier between the robot and the goal). In Fig. A.1, the primary direction was north and the secondary direction was east. Avoiding known dead-ends, such as  $\{(4, 9) (5, 9) (5, 8)\}$  in Fig. A.1, search repeatedly moves toward the goal, in the primary direction when possible, otherwise in the secondary direction or their opposites until the goal is in view or until backup (permitted opposite to the primary and secondary directions) would exceed the original alignment coordinates. While time permits, if this search fails to produce a solution fragment, the algorithm will shift the robot laterally, first in the secondary direction, then in the opposite of the secondary direction, and attempt to circumnavigate from there. Although it iterates with increasingly large shifts until it succeeds, Roundabout is time-limited and heuristic like any tier-2 Advisor. Its search may fail to circumnavigate the obstruction, perhaps getting closer to the goal than it had been when it started but without actually bringing the goal into sight. All of Roundabout's search paths, successful or not, serve as input for barrier learning.



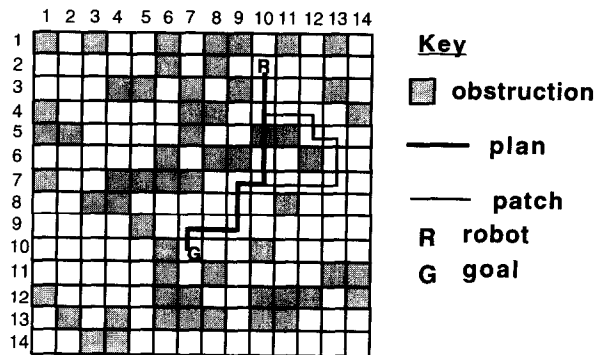


Fig. A.2. Patchwork repairs a plan.

*Outta Here* is a tier-2 Advisor that attempts to leave a chamber or dead-end if it does not contain the goal. *Outta Here* triggers when the task is well underway and either the robot's recent locations cover a relatively small fraction of the total area of the maze or Ariadne believes the robot to be in a dead-end or chamber not containing the goal. If the robot is in a dead-end, *Outta Here* marches out with a sequence of steps that lead through its exit. If the robot is in a chamber, *Outta Here* scans the way the chamber-learning routine does (and may learn a chamber as a side-effect) before it returns a sequence of up to three steps that move the robot out through the access point of the chamber. If *Outta Here* were to trigger in Fig. 7(a) (see p. 287) when the robot was at (2, 12), it could generate the path  $\langle (4, 12) (4, 13) (10, 13) \rangle$ . *Outta Here* is not guaranteed to find an access point, however, and may return the robot to a location it has already visited during the current task.

*Probe* is a tier-2 Advisor that determines the extent of the robot's confinement and then attempts to leave that space. Its trigger is the learning condition described for chambers. *Probe* scans to improve the view, that is, it tries to move the robot to an access point for the current chamber and then orthogonally leave the chamber. Chambers are learned during these scans as a side-effect of this search. From (4, 12) in Fig. 7(a), depending upon its recent experience, *Probe* actually generated a path out and thereby learned the chamber shown. *Probe* is not guaranteed to learn a chamber.

*Patchwork* is a tier-2 Advisor that attempts to repair a current plan with the same circumnavigation routine as Roundabout, if the plan fails because of an obstruction. Fig. A.2 shows a typical *Patchwork* plan that naively disregards the obstruction at (5, 10). When *Patchwork* triggered in Fig. A.2, it repaired the indicated plan by inserting the fragment  $\langle (4, 11) (4, 12) (5, 12) (5, 13) (6, 13) (7, 13) \rangle$  to move from (4, 10) to (7, 10). *Patchwork* triggers when there are valid current plans and 30% of the expected number of decisions has been exhausted. This prevents premature, overly elaborate solutions when simpler approaches would suffice. When *Patchwork* succeeds in repairing one step in a plan and the result contains at least one location new during this task, *Patchwork* forces that fragment, that is, it executes all the initial steps of the plan that need no correction, one circumnavigation sequence as a patch, and the next set of steps

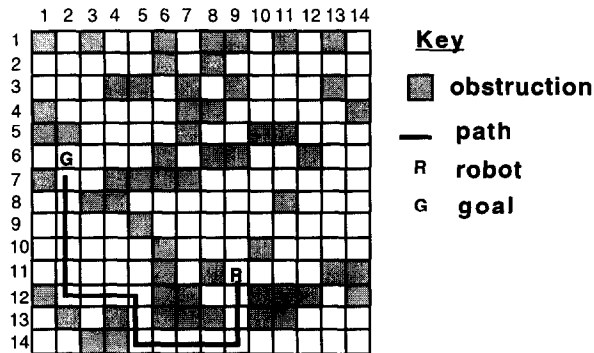


Fig. A.3. Other Side shifts the robot's orientation to the goal.

that need no correction. In this way Patchwork moved the robot in Fig. A.2 directly to the goal.

*Other Side* is a tier-2 Advisor that attempts to shift the robot from one side of the goal to the other. In decision cycles, *Other Side* is costly, so its trigger describes an appropriate but also fairly desperate state. *Other Side* triggers when it is late, there are no current plans, the other side of the goal is within the edges of the maze, the current location is not the result of a just-executed fragment, the robot has been in this location before, and throughout the task the robot has been on one side (left, right, above, or below) of the goal. *Other Side* then attempts to move the robot to the other side (right, left, below, or above, respectively) of the goal. An example of an *Other Side* path appears in Fig. A.3, from an early learning task on this maze. The robot began the task at (9, 12) and kept trying to penetrate the irregular barrier running north-south to reach the goal. When *Other Side* finally triggered, the robot was at (11, 9) and had always been to the right of the goal. *Other Side* will abandon its directional attempt if the goal comes into view, as it did from (12, 2). If, for example, the goal had been at (4, 2) instead, *Other Side*'s path would have gone from (12, 2) to (11, 2) then halted at (11, 1) to the left of the goal instead, still a constructive contribution.

*Super Quadro* is a tier-2 Advisor that attempts to change the robot's quadrant. It triggers when the task is well underway and the robot has been in its current quadrant (or its current quadrant and the goal's quadrant) for some time. *Super-Quadro* scans to find a move into the extent of a gate that would change the robot's quadrant. It tries to find a sequence of orthogonal steps to a location whose quadrant is different, preferably the goal quadrant if it has not been there recently. From (5, 5) in Fig. A.4, taken from an early learning task where Ariadne had not yet found its way to the goal through (2, 4), *Super Quadro* generated the path  $\langle (5, 6) (5, 11) \rangle$ . This takes the robot into the goal's quadrant, but cannot lead it to the goal. (Ariadne eventually found the right solution.) *Super Quadro* has no heuristics for preferring one gate to another (since that would require even further search), so its solution fragments, like this one, may not always be constructive.

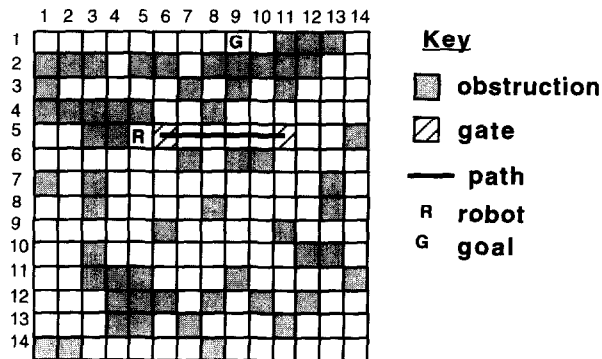


Fig. A.4. Super Quadro responds to its trigger.

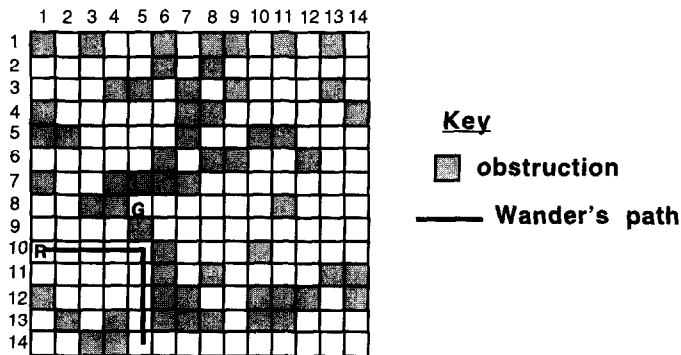


Fig. A.5. A situation that triggered Wander.

*Wander* is a tier-2 Advisor that attempts to find an L-shaped path that leads it to a new location, one as far from the robot's current location as possible. *Wander* triggers only when the robot's behavior is judged constrained and repetitive, and the current location was not just the result of a solution fragment. Wandering becomes less likely as more bases are identified; its trigger is therefore stochastic, with probability

$$1 - \frac{|\text{bases}|}{0.1|\text{unobstructed maze locations}|}.$$

*Wander* tests up to eight L-shaped paths that are pairs of longest possible steps in two orthogonal directions. As a side-effect, *Wander* learns dead-ends on the first leg of the L. On the second leg of the L, it does not pass a location that would align the robot with the goal unless the robot has already visited that location during this task, nor does it move toward a barrier between it and the goal. *Wander* often produces several solution fragments but, like any tier-2 Advisor, it can only recommend one. It prefers to recommend one in the direction of the goal that ends in a location farthest from the

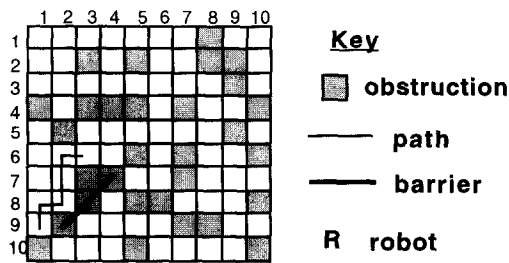


Fig. A.6. Part of a Humpty Dumpty path and the barrier learned from it.

robot's current location, has been rarely visited, and has not been visited recently. When Wander triggered in Fig. A.5, the robot had begun at (8,2) and made little progress. Now Wander forced the path  $\langle (10,5) (14,5) \rangle$ . Wander is the least reliable of the tier-2 Advisors, but it often makes a dramatic impact when one is sorely needed.

*Humpty Dumpty* is a tier-2 Advisor that goes in search of barriers, hindrances that might be avoided by judicious move selection. It triggers if the robot has been to most of the locations available through the current legal moves and there are few known barriers in its vicinity. The fragment *Humpty Dumpty* returns is one that traverses the immediate area along contiguous obstructions. It has the least goal-directed algorithm in tier 2, but the data it collects serves as important input to several other Advisors. A *Humpty Dumpty* path fragment and the barrier extracted from it appear in Fig. A.6.

### A.3. Tier 3

Advisors in tier 3 are consulted simultaneously. Their order of presentation here is only to facilitate discussion.

Goal Row and Goal Column attempt to align the robot with the goal. *Goal Column* favors moves that align the robot vertically with the goal only if it is not already; *Goal Row* favors moves that do so horizontally. Giant Step and Plod advocate large and small steps, respectively. Late in a task, *Hurry* proportionately encourages those moves with the five longest steps. When the robot has recently been confined to a small area, *Giant Step* encourages the five longest moves with proportionate strengths. *Plod* advocates single unit moves, more strongly (with a higher comment strength) for those toward the goal and when the robot is close to *G*.

Mr. Rogers and Contract address the Euclidean distance to the goal. *Mr. Rogers* supports, with strengths proportional to their result, moves into the goal's neighborhood, that is, those that produce the shortest distances to the goal. *Contract*, on the other hand, advocates large steps when the robot is far from the goal, and small steps when the robot is close to it. Closeness is measured relative to the maze's diagonal, its maximum possible distance. Both Advisors address proximity, but Mr. Rogers seeks to be close to the goal, while Contract uses proximity to determine step size, whether or not those steps draw the robot closer to the goal.

Been There, Done That, Cycle Breaker, and Adventure all address prior behavior during the current task. *Been There* discourages returning to a location already visited during this task, with more discouragement (a lower comment strength) for more frequently visited locations. With careful representation, such Advisors can respond as if they were reactive. One way to make *Been There* reactive, for example, would be to note locations already visited in the current task in an array and “sense” there. *Done That* discourages moving in the same direction one did before from a previously visited location. When the robot repeatedly visits the same few spots in its most recent moves, *Cycle Breaker* attempts to intervene by suggesting an alternative that, while not necessarily new, is novel in the current context. *Adventure*, in contrast, encourages innovation. *Adventure* recommends moves to locations thus far unvisited during this task, with greater strengths for those in the direction of the goal.

*Chamberlain* discourages a move into the extent of a chamber if the goal is not there, and encourages such a move if the goal might be there. If the robot is already in a chamber where the goal is not, *Chamberlain* encourages moves to its access point (to support a subsequent exit). *Chamberlain* is less powerful than *Probe* because it does not search and can only recommend a single move. *Cork* is the tier-3 version of *Chamberlain* for bottles. Any location from which a bottle’s neck can be seen provides an exit from the bottle. When the robot is outside the bottle, *Chamberlain* discourages moves into a bottle whose extent indicates that it cannot contain the goal, and encourages moves into the neck of a bottle or into the bottle itself whose extent indicates that it can contain the goal. When the robot is inside the bottle, *Cork* reverses this advice.

*Detour* avoids moving toward barriers that lie directly between a location and the goal. If a barrier already intervenes, *Detour* encourages moves to avoid it.

*Quadro* is a simpler version of *Super Quadro*. It encourages, with decreasing strengths, moves to known gates into the goal’s quadrant, moves into the extent of a known gate into the goal’s quadrant, moves to known gates into another quadrant, and moves into the extent of a known gate into another quadrant.

*Home Run* and *Leap Frog* both use bases. *Home Run* encourages moves to bases and, with lesser strengths, moves to locations next to bases. *Leap Frog* advocates a move to the legal location closest to the end of any current plan.

*Corner* advocates a move to the far exit of a pipe that turns, more strongly when the pipe is in the direction of the goal, less so when it is opposite the direction of the goal. *Crook* advocates a move to the near exit of a corridor that is neither straight nor a dead-end, again more strongly when the corridor is in the direction of the goal, less so when the far exit of the corridor is opposite the direction of the goal.

*Opening* encourages the reuse of previously successful path beginnings when applicable. Although such moves may seem odd if the goal is in a different location, the heuristic works well if the old path was successful because it began by moving to an area that offered good access to other parts of the maze.

#### A.4. Parameters for Advisor applicability

One common component of the triggers for tier-2 Advisors is a measure of lateness relative to available resources. Each task in a FORR-based system can be terminated if

the decision limit is met or the computation time allocated to a problem is exhausted. Depending upon the trigger, lateness may be measured as the number of actual moves in the partial solution, the percentage of the decision limit or computation time thus far exhausted (whether or not it resulted in moves), or the ratio of actual moves to the average task length. The following parameters were applied during the experiments described here:

- Giant Step: The most recent 30% of the history is confined to no more than 10% of the maze area.
- Humpty Dumpty: The most recent 30% of the history includes at least 80% of the legal actions, there are fewer than 3 known walls in the robot's vicinity, and 50% of the decision limit has been exhausted.
- Opening: The first 15% of the moves constitute the opening.
- Other Side: 60% of the decision limit has been exhausted.
- Outta Here: The most recent 30% of the history is confined to no more than 10% of the maze area.
- Wander: The most recent 30% of the history is confined to no more than 10% of the maze area.

## References

- [1] P.E. Agre, D. Chapman, What are plans for?, *Robotics and Autonomous Systems* 6 (1990) 17–34.
- [2] G. Biswas, S. Goldman, D. Fisher, B. Bhuvu, G. Glewwe, Assessing design activity in complex CMOS circuit design, in: P. Nichols, S. Chipman, R. Brennan (Eds.), *Cognitively Diagnostic Assessment*, Lawrence Erlbaum, Hillsdale, NJ, 1995.
- [3] L.K. Branting, D.W. Aha, Stratified case-based reasoning: Reusing hierarchical problem solving episodes, in: *Proceedings 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montreal, Que., Morgan Kaufmann, San Mateo, CA, 1995, pp. 384–390.
- [4] R.A. Brooks, Intelligence without representation, *Artificial Intelligence* 47 (1991) 139–160.
- [5] E. Chown, S. Kaplan, D. Kortenkamp, Prototypes, Location, and Associative Networks (PLAN): towards a unified theory of cognitive mapping, *Cognitive Science* 19 (1995) 1–51.
- [6] K. Crowley, R.S. Siegler, Flexible strategy use in young children's tic-tac-toe, *Cognitive Science* 17 (4) (1993) 531–561.
- [7] E. DeYoe, D. Van Essen, Concurrent processing streams in the monkey visual cortex, *Trends in Neuroscience* 11 (1988) 219–226.
- [8] M.J. Egenhofer, D.M. Mark, Naive Geography, Technical Report 95-8, National Center for Geographic Information and Analysis, 1995.
- [9] S.L. Epstein, Prior knowledge strengthens learning to control search in weak theory domains, *Internat. J. Intelligent Systems* 7 (1992) 547–586.
- [10] S.L. Epstein, For the Right Reasons: the FORR architecture for learning in a skill domain, *Cognitive Science* 18 (3) (1994) 479–511.
- [11] S.L. Epstein, Collaboration and interdependence in limitedly rational agents, in: *Proceedings AAAI Fall Symposium on Rational Agency*, AAAI Press, Cambridge, MA, 1995.
- [12] S.L. Epstein, On heuristic reasoning, reactivity, and search, in: *Proceedings 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montreal, Que., Morgan Kaufmann, San Mateo, CA, 1995, pp. 454–461.
- [13] S.L. Epstein, J. Gelfand, J. Lesniak, Pattern-based learning and spatially-oriented concept formation with a multi-agent, decision-making expert, *Computational Intelligence* 12 (1) (1996) 199–221.
- [14] S.L. Epstein, J. Gelfand, E.T. Lock, Learning game-specific spatially-oriented heuristics, Constraints, to appear.

- [15] K.A. Ericsson, N. Charness, Expert performance: Its structure and acquisition, *American Psychologist* 49 (8) (1994) 725–747.
- [16] K.A. Ericsson, R.T. Krampe, C. Tesch-Römer, The role of deliberate practice in the acquisition of expert performance, *Psychological Review* 100 (3) (1993) 363–406.
- [17] K.A. Ericsson, J. Smith, *Toward a General Theory of Expertise—Prospects and Limits*, Cambridge University Press, Cambridge, UK, 1991.
- [18] R.G. Golledge, Path selection and route preference in human navigation: a progress report, in: *Proceedings 2nd International Conference on Spatial Information and Theory (COSIT '95)*, Semmerling, Austria, Lecture Notes in Computer Science, Springer, Berlin, 1995, pp. 207–222.
- [19] A. Gryl, Analyse cognitive des descriptions d'itinéraires, in: *Proceedings Premier Colloque Jeunes Chercheurs en Sciences Cognitives, La Motte d'Aveillans, France, 1994*.
- [20] A. Gryl, Analyse et modélisation des processus discursifs mis en oeuvre dans la description d'itinéraires, Ph.D. Thesis, Université Paris XI Orsay, 1995.
- [21] P.J. Hayes, The second naive physics manifesto, in: J.R. Hobbs, R.C. Moore (Eds.), *Formal Theories of the Commonsense World*, Ablex Publishing, Norwood, NJ, 1988, pp. 1–36.
- [22] P.J. Hayes, K.M. Ford, N. Agnew, On babies and bathwater: a cautionary tale, *AI Magazine* 15 (4) (1994) 15–26.
- [23] G.S. Klein, R. Calderwood, Decision models: some lessons from the field, *IEEE Trans. Systems Man Cybernet.* 21 (5) (1991) 1018–1026.
- [24] J.L. Kolodner, Introduction to the Special Issue on Case-Based Reasoning, *Machine Learning* 10 (3) (1993) 1–5.
- [25] R. Korf, Real-time heuristic search, *Artificial Intelligence* 42 (1990) 189–211.
- [26] B. Kuipers, R. Froom, W.-Y. Lee, D. Pierce, The semantic hierarchy in robot learning, in: J. Connell, S. Mahadevan (Eds.), *Robot Learning*, Kluwer Academic Publishers, Boston, MA, 1993, pp. 141–170.
- [27] B.J. Kuipers, Modeling spatial knowledge, *Cognitive Science* 2 (1978) 129–153.
- [28] B.J. Kuipers, T.S. Levitt, Navigation and mapping in large-scale space, *AI Magazine* 9 (2) (1988) 25–43.
- [29] D.B. Lenat, AM: an artificial intelligence approach to discovery in mathematics, Ph.D. Thesis, Department of Computer Science, Stanford University, 1976.
- [30] D.B. Lenat, EURISKO: a program that learns new heuristics and domain concepts, *Artificial Intelligence* 26 (1983) 61–98.
- [31] A.W. Moore, C.G. Atkeson, Prioritized sweeping: reinforcement learning with less data and less time, *Machine Learning* 13 (1) (1993) 103–130.
- [32] B. Pell, Exploratory learning in the game of Go: initial results, in: D.N.L. Levy, D.F. Beal (Eds.), *Heuristic Programming in Artificial Intelligence 2—The Second Computer Olympiad*, Ellis Horwood, Chichester, UK, 1991, pp. 137–152.
- [33] J. Piaget, B. Inhelder, *The Child's Conception of Space*, W.W. Norton, New York, 1967.
- [34] D. Pierce, B.J. Kuipers, Learning to explore and build maps, in: *Proceedings 12th National Conference on Artificial Intelligence (AAAI-94)*, Seattle, WA, AAAI/MIT Press, Cambridge, MA, 1994, pp. 1264–1271.
- [35] A.E. Prieditis, Effective admissible heuristics, *Machine Learning* 12 (1–3) (1993) 117–141.
- [36] M.J. Ratterman, S.L. Epstein, Skilled like a person: a comparison of human and computer game playing, in: *Proceedings 17th Annual Conference of the Cognitive Science Society*, Lawrence Erlbaum, Pittsburgh, PA, 1995, pp. 709–714.
- [37] M.J. Ratterman, S.L. Epstein, Game-playing expertise in young children (in preparation).
- [38] E.K. Sadalla, S.G. Magel, The perception of traversed distance, *Environment and Behavior* 12 (1980) 65–79.
- [39] M.J. Schoppers, In defense of reaction plans as caches, *AI Magazine* 10 (4) (1989) 51–60.
- [40] S.J.J. Smith, D.S. Nau, T.A. Throop, Total-order multi-agent task-network planning for contract bridge, in: *Proceedings 13th National Conference on Artificial Intelligence (AAAI-96)*, Portland, OR, MIT Press, Cambridge, MA, 1996, pp. 108–113.
- [41] E. Stefanakis, M. Kavouras, On the determination of the optimum path in space, in: *Proceedings 2nd International Conference on Spatial Information Theory (COSIT '95)*, Semmerling, Austria, Lecture Notes in Computer Science, Springer, Berlin, 1995, pp. 241–257.

- [42] R.S. Sutton, Integrated architectures for learning, planning, and reacting based on approximating dynamic programming, in: *Proceedings 7th International Conference on Machine Learning*, Austin, TX, Morgan Kaufmann, San Mateo, CA, 1990, pp. 216–224.
- [43] A. Tate, J. Hendler, M. Drummond, A review of AI planning techniques, in: J. Allen, J. Hendler, A. Tate (Eds.), *Readings in Planning*, Morgan Kaufmann, San Mateo, CA, 1990, pp. 26–49.
- [44] B. Tversky, Spatial mental models, *The Psychology of Learning and Motivation* 27 (1991) 109–145.
- [45] B. Tversky, Distortions in cognitive maps, *Geoforum* 23 (2) (1992) 131–138.
- [46] L. Ungerleider, M. Mishkin, Two cortical visual systems, in: D. Ingle, M. Goodale, R. Mansfield (Eds.), *Analysis of Visual Behavior*, MIT Press, Cambridge, MA, 1982, pp. 548–586.