Spatial
Representation for Pragmatic Navigation

Susan L. Epstein
Department of Computer Science
Hunter College and The Graduate School of The City University of New York
New York, NY 10021 epstein@roz.hunter.cuny.edu

Abstract

As described here, pragmatic navigation attempts to harness simple facts about a two-dimensional environment to facilitate travel through it without an explicit map. It relies upon predefined spatial representations whose explicit instances are learned during a sequence of trips through a fixed maze. Once learned, any of these instances can be applied to subsequent travel. Some of the representations are heuristic, as are the procedures that employ them. The resultant performance of an implementation, particularly when contrasted with traditional AI techniques, argues for path-finding guided by representations like those detailed here.

1. The Task

People often find themselves without a map in a foreign spatial region, such as a college campus or a small town, where they are expected to make their way between many different pairs of locations with increasing efficiency. During a variety of trips through that region, they construct not a detailed map but pragmatic representations that simplify path-finding (Gryl 1994a; Gryl 1994b). It is the thesis of this work that a robot can learn "its way around" a region efficiently and effectively with a variety of spatial representations instead of an explicit, detailed map. This paper describes a cognitive and computational model that supports successful, rapid robot navigation in two-dimensional, partially-obstructed space.

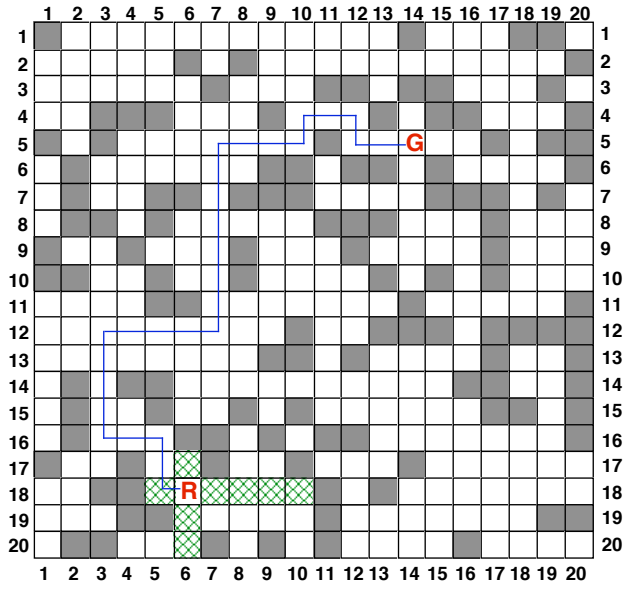

Figure 1: A path-finding problem. The robot must move to the goal in unidirectional steps through unobstructed locations. Its current legal moves are cross-hatched, and one possible solution path is shown.

Consider a perceptually-impaired robot operating in a grid with multiple internal obstructions, like that in Figure 1. The robot has no explicit, detailed map of its world, and is not permitted to construct one. At any instant in time, the state of the world is described to the robot only as the dimensions of the maze, the coordinates of the goal, the robot's own coordinates, the path it has thus far traversed, and how far the robot can "see" in four directions to the nearest obstruction or to the goal. The robot in Figure 1, for example, knows that it is in a 20 × 20 maze, that there is a goal at (5, 14), that its own location is (18, 6), that it has not yet moved, and that the view is clear as far north as 17, as far south as 20, as far east as 10, and as far west as 5. At each decision step the robot selects a single direction and distance to move. The robot of Figure 1 can choose only among the eight legal moves crosshatched in the diagram. A *problem* for the robot is represented as a pair ($start_R$, $start_G$), where $start_R$ is the robot's initial location and $start_G$ is the location of some stationary goal it must reach in a sequence of no more than n legal moves (the *decision-step* limit).. Such a problem is *solvable* if and only if there exists some path $<loc_0 move_1 loc_1 \ldots loc_{i-1} move_i loc_i \ldots loc_{p-1} move_p loc_p>$ where $start_R = loc_0$ and $loc_p = start_G$, $p \le n$, and $move_i$ is a legal move from loc_{i-1} to loc_i through only unobstructed locations, for $i = 1, 2, \ldots, p$. The *level of difficulty* of a solvable problem is the minimum value of p for which there is a solution. The intent is to provide the robot with a series of path-finding problems in the same maze, with the expectation that its performance will improve, both on problems it has encountered before and on new ones. One further constraint: the robot may not sense along the path as it travels, only at locations where it stops to make the next decision.

This was originally posed in an AI context as an example of a particularly challenging search problem (Korf 1990). The alternative approach described here is a reactive learner that gradually acquires pragmatic knowledge about the space, and then applies it to

subsequent tasks. It originates from an architecture called FORR.

2. FORR

FORR (FOrr the Right Reasons) is a problem-solving and learning architecture intended to model the transition from general expertise to specific expertise (Epstein 1994). A FORR-based system begins with a *domain* of related problem classes, such as mazes, and some domain-specific but problem-class-independent knowledge, such as "avoid dead-ends." With problem solving experience, such as trips from one location to another, a FORR-based program gradually acquires *useful knowledge*, potentially applicable and probably correct data for a specific problem class that should enhance its performance.

FORR's hierarchical reasoning process is shown in Figure 2. FORR has tiers of *Advisors*, domain-specific but problem-class-independent, decision-making rationales, 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, the current permissible actions from that state, and any learned useful knowledge about the problem class under consideration. Each Advisor outputs any number of *comments* that support or discourage permissible actions. A comment lists the Advisor's name, the action commented upon, and a *strength*, an integer from 0 to 10 that measures the intensity and direction of the Advisor's

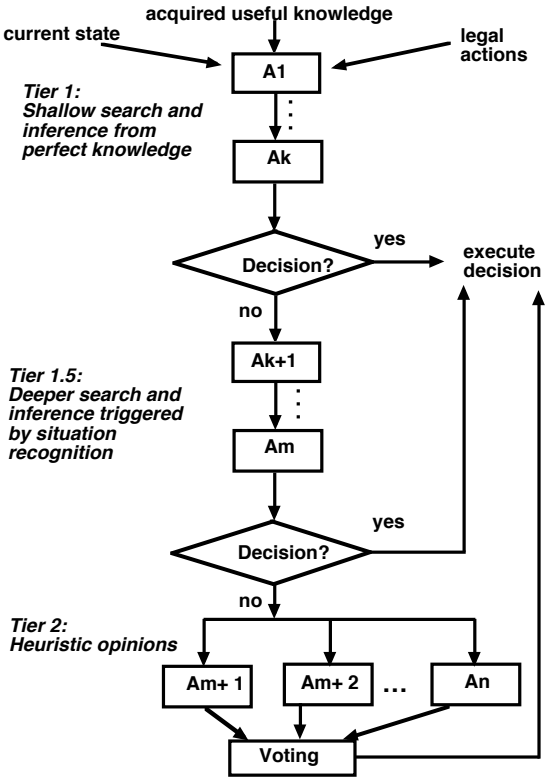

Figure 2: How FORR makes decisions.

opinion. Although there are no constraints on the nature of the comment-generating procedures themselves, FORR is intended to be used with Advisors that eschew extensive search.

To select an action at any point, a FORR-based system senses the current state of the world and forwards its perceptions to a hierarchy of Advisors. Tier-1 Advisors are consulted in a predetermined, fixed order. They may have the authority to make a decision unilaterally or to eliminate a legal action from any further consideration. Tier-1 Advisors are reactive, consulted in sequence, and reference only correct useful knowledge. Given the current state of the world and what they know about the problem class, any decision they make will be quick and correct. For path-finding, a good tier-1 Advisor is "if you see the goal, go to it." If the first tier of a FORR-based system fails to make a decision, control defaults to tier 1.5.

Each tier-1.5 Advisor has its own reactive trigger, plus a procedure that generates and tests a highly-constrained set of possible solution fragments. A solution *fragment* emerges from a tier-1.5 Advisor as a sequence of decisions, rather than a single reactive one, a digression from the "sense-compute-execute" loop. Tier-1.5 is prioritized too, but lacks any guarantee of correctness. If tier 1 has failed to make a decision, each tier-1.5 Advisor has the opportunity in turn to trigger. The first to trigger is immediately ceded control with limited time to generate and test its fragments. For path-finding, a good tier-1.5 Advisor is "if you are aligned with the goal but there is an intervening wall, go around it." If a tier-1.5 Advisor constructs what it believes to be a solution fragment, that fragment is executed and then, regardless of the outcome, control is returned to tier 1. If no tier-1.5 Advisor triggers or produces a fragment, the decision is made in tier 2.

Tier-2 Advisors are not necessarily independent, or even correct in the full context of the state space. Each of them epitomizes a heuristic, specialized view of reality that can make a valid argument for or against one or more actions. Tier-2 Advisors are reactive, but far less trustworthy those of tier 1, because neither their reasoning process nor the useful knowledge on which they rely is guaranteed correct. For path finding, a good tier-2 Advisor is "move in the direction of the goal." All of the tier-2 Advisors have an opportunity to comment before any decision is made. The decision they compute by tallying their comment strengths represents a consensus of opinion.

3. Learning pragmatic navigation knowledge

Pragmatic navigation knowledge is useful knowledge for a specific maze. It is learned from experience and, despite possible inaccuracies, is generally expected to enhance performance. Each kind of pragmatic navigation knowledge is prespecified, including its learning algorithm, learning time limit, and schedule (after a

decision, a task, or a set of tasks). Pragmatic navigation knowledge may also be learned during a decision, when the robot has recently been *constrained,* i.e., when any of the following is true:

• The area of the territory bounded 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.

• Half the recent moves were to locations visited at least once before.

The work reported upon here used x = 30% and y = 10%.

Figure 3 provides examples of pragmatic navigation knowledge for the maze of Figure 1, as implemented in a program called *Ariadne*. (Ariadne, daughter of King Minos, helped Theseus find his way through the labyrinth.)

From the dimensions of the maze, the robot can identify quadrants. A *gate* is a location that offers a transition from

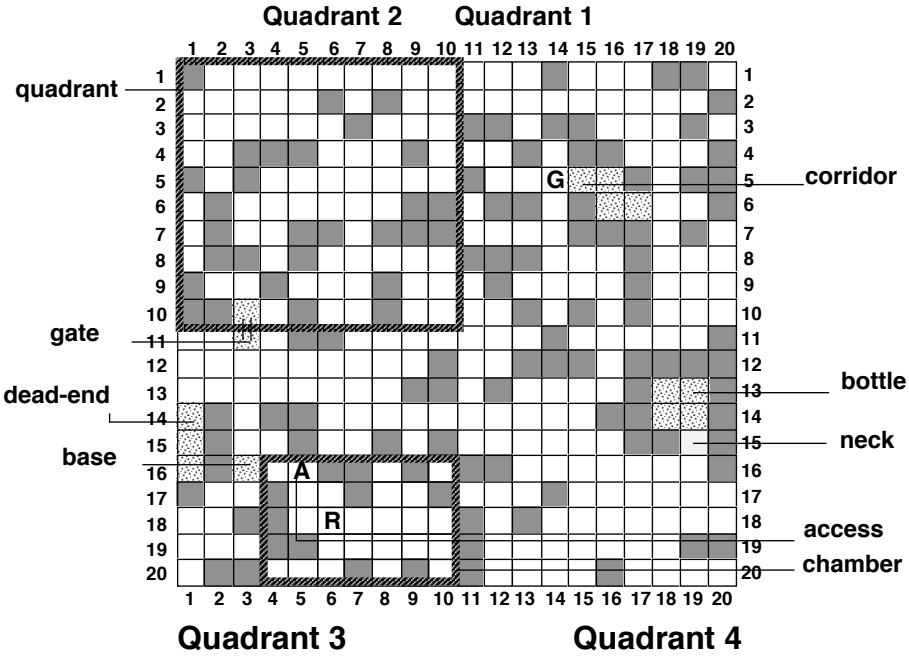

Figure 3: Some examples of pragmatic navigation knowledge.

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 during problem solving as a gate between the current quadrant and the previous one. A gate may not always be helpful; for example, (11, 3) is a gate between quadrants 3 and 2 in Figure 3, but it offers access to little of quadrant 2. Each gate is stored with its *extent* (rectangular approximation of the locations to which it affords access, computed as the view from the current location after passing though the gate) in a hash table whose key is the sorted pair of quadrant numbers. If the robot moves from (11, 3) to (10, 3), the extent of the gate at (11, 3) is computed from (10, 3) as 9 north, 4 east, 17 south, and 3 west.

A *corridor* is a passageway of width one that either leads nowhere (a *dead-end*) or is a *hallway*. In Figure 3, {(14, 1), (15, 1), (16, 1)} is a dead-end and {(5, 15), (5, 16), (6, 16), (6, 17)} is a hallway that zigzags. A corridor is learned when, from the current state, the robot has only one or two moves. Only the endpoints of a corridor are recorded; each serves as the key to a hash table whose returned values are the other endpoint and whether or not it is a dead-end. Corridors are enlarged and merged together as necessary.

A *chamber* is an irregularly shaped space with an access point and an approximate extent that at worst overstates the chamber by a bounding rectangle. The *access point* of a chamber is any location within the chamber that affords a view beyond it. Figure 3's robot is in a chamber with access point (16, 5) and extent 16 north, 10 east, 20 south, and 4 west; from (16, 5) the robot can see west beyond its extent to (16, 3). All locations reachable from the robot really constitute one large chamber, but the chambers that Ariadne learns are more limited and room-like. A chamber is learned during problem solving 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 very few legal moves to locations not yet visited on this trip, and the current location was not the result of a tier-1.5 fragment. The learning algorithm for a chamber estimates the extent according to its current percepts, and then tries to move the robot to locations where the chamber appears 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 that enlarge the extent, at least one of which is previously unvisited during this trip, it records the chamber's extent and access point (the second location if there were two, otherwise the first) on a list. In Figure 3, for example, the horizontal scan from (18, 6) enlarges the northern view at (18, 5), and then the vertical scan finds the access point (16, 5). 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 space, learned after problem solving from analysis of the entire path after a trip is completed. A potential bottle begins with a location that was visited more than once. The bottle 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 (the outer boundaries of the locations it includes) computed, its *neck* (entry and/or exit point) is identified. Bottles are stored in a hash table as an extent and a neck. Figure 3 shows a bottle with extent 13 north, 19 east, 14 south, and 18 west and neck (15, 19).

Table 1: Ariadne Advisors that do not apply useful knowledge. Those that do apply it are detailed in the text.

Tier	Advisor	Rationale
1	Victory	If the goal is reachable by a legal move, go there.
1.5	Probe	Determine the current extent and try to leave it.
1.5	Other Side	Move the robot to the opposite side of the goal.
2	Adventure	Move to thus far unvisited locations, preferably toward the goal.
2	Been There	Discourage returning to a location already visited on this trip.
2	Contract	Take large steps when far from the goal, and small steps when close to it.
2	Cycle Breaker	Stop repeated visits to the same few spots.
2	Done That	Discourage moving in the same direction as before from a previously-visited location.
2	Giant Step	If recently confined, take a long step, preferably toward the goal.
2	Goal Column	Align the robot horizontally with the goal, if it is not already.
2	Goal Row	Align the robot vertically with the goal, if it is not already.
2	Mr. Rogers	Move into the neighborhood of the goal.
2	Plod	Take a one-unit step, preferably toward the goal.

A *base* is a location in the maze that appears to have beena 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 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 within a 10-mile radius. A base is such a location. Bases are learned after problem solving from analysis of a successful path that has been corrected to eliminate loops and unnecessary digressions. A base is an extreme location in that corrected path that was not in the heading from the robot to the goal, in other words, a counterintuitive move. For example, in Figure 1 the move on the solution path from (18, 6) to (18, 5) is away from the goal, so (18, 5) would be learned as a base. A base is not a dead-end or $start_G$ itself, and solution fragments constructed during search to circumvent an obstruction aligned with the goal contribute only their most extreme positions opposite the original headings. Bases are stored on a list.

Two other items of useful knowledge are learned automatically by FORR after problem solving and have no particular spatial significance: average task length in decisions steps and *openings*, previously successful path beginnings.

Pragmatic navigation knowledge is heuristic. No data is guaranteed to be correct, or applicable to any other problem. The learning algorithm for any item may reference useful knowledge or the ideas behind it to distinguish among kinds of items. For example, gates are not dead-ends. For the most part, however, one location might participate in several kinds of items, and thereby be considered by a variety of Advisors, without constraint. This means that, for example, a gate may also be a base or the neck of a bottle.

4. Applying pragmatic navigation knowledge

Ariadne applies acquired pragmatic navigation knowledge about a particular maze to solve problems in it. Each of its Advisors is a narrow decision-making rationale designed for path finding in general, rather than for some particular maze. Those Advisors that apply no pragmatic navigation knowledge are summarized in Table 1; further details are available on them in (Epstein 1995).

Tier 1 Advisors are perfectly correct, reactive procedures that decide quickly and mandate a single move. *No Way* is the only tier-1 Advisor in Ariadne that applies pragmatic navigation knowledge. It 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, unless the robot is already

in the dead-end (and therefore needs to get out).

Tier 1.5 Advisors are heuristics that do time-limited search in an attempt to produce a sequence of moves that they then mandate. (These Advisors simulate human, situation-based behavior (Klein & Calderwood 1991).) Each is defined by its trigger, which signals its applicability, and its search method. Ariadne has three that apply useful knowledge: Outta Here, Roundabout, Wander, and Super Quadro.

Outta Here attempts to leave a confined space that does not contain the goal. It triggers when the trip is well underway and either the robot's recent locations cover a relatively small fraction of the total area of the maze, or it believes itself 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 to its open end. 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 Figure 3 when the robot was at (18, 6), it would generate the path (18, 5) → (16, 5) → (16, 3). Outta Here is not guaranteed to find an access point, and may return the robot to a location it has already visited.

Roundabout attempts to circumnavigate a wall between the robot and the goal. It triggers when the robot is aligned with the goal, either horizontally or vertically. This is not a traditional wall-following algorithm; it establishes a primary

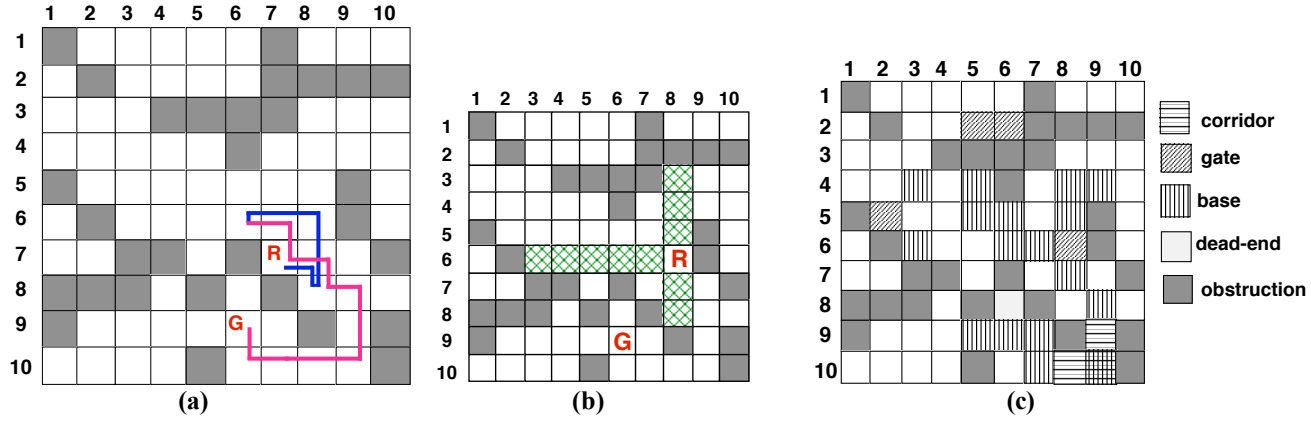

Figure 4: In one maze, (a) a solution path, (b) a simple decision situation, and (c) pragmatic navigation knowledge.

direction (toward the goal) and a secondary direction (orthogonal to the primary). Avoiding dead-ends, search repeatedly moves in the primary direction when possible, otherwise in the secondary direction until the goal is in view or 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 iterate after shifting the robot one or more steps in either secondary direction. Roundabout will proffer the first path that gets closer to the goal, even if it is not in sight at the end. In Figure 1 the portion of the solution from (5, 10) to (5, 13) was mandated by Roundabout.

Wander 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. It 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 stochastic, with probability

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

Super Quadro attempts to change the robot's quadrant. It triggers when the trip 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 (18, 5) in Figure 3, depending upon its recent experience, Super Quadro could generate the path (16, 5) → (16, 3) → (10, 3) into the extent of, and through, the (11, 3) gate. Super Quadro has no heuristics for preferring one gate to another, so its solution fragment may not always be constructive.

Tier-2 Advisors are reactive, time-limited heuristics that embody path-finding commonsense and do no forward search in the maze. Each may recommend or oppose any number of legal moves that have not already been eliminated by No Way. Although every tier-2 Advisor captures a reasonable rationale for navigation, none should be trusted to decide alone. All 17 vote together. The simple ideas behind them support rapid computation; given 10 seconds, none of Ariadne's tier-2 Advisors has yet run out of time.

Opening encourages the reuse of previously successful path beginnings. Even if the goal is in a different location, its heuristic may work 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. *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,

Table 2: Tier 2 votes on Figure 4(a).

Move	Comments	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)	**Home Run 8, Mr. Rogers 8, Giant Step 10, Adventure 8, Goal Column 10**	**19**
(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

Chamberlain encourages moving to an access point (to support a subsequent exit). *Cork* is the bottle version of Chamberlain. When the robot is outside the bottle, it discourages moves into the neck of a bottle whose extent indicates that it cannot contain the goal, and encourages moves into the neck of a bottle whose extent indicates that it can contain the goal. When the robot is inside the bottle, Cork reverses this advice. *Quadro* is a simplistic 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* encourages moves to bases and, with lesser strengths, moves to locations near bases. *Leap Frog* constructs high-level plans as a result of bidirectional search on aligned bases, as if there were no obstructions, and supports moves to the bases in its plans. Late in a trip (as measured by the average task length), *Hurry* proportionately encourages the moves with the five longest steps.

Figure 4(a) shows Ariadne's somewhat torturous solution to a level 4 problem during learning in a scaled-down maze. Ariadne starts out well, but when unable to move south or west toward the goal from (8, 8), it goes to (6, 8). The decision situation at (6, 8) is detailed in Figure 4(b). At this point, tiers 1 and 1.5 have already waived their right to decide on the next move, and (6, 3) is a learned base from a previous trip. Ariadne's actual comments appear in Table 2, where the total score for a move is computed by converting the comment strength range from [0, 10] to [–5, 5] and then adding the strengths. The move to (6, 6) will be selected because it is near a base, it gets closer to the goal than some of the other moves, it is a relatively long step, it has not yet been visited on this trip, and it is in the goal column. Once Ariadne moves to (6, 6), Roundabout computes a fragment that puts the robot in sight of the goal, and the trip ends as Victory moves there.

Because of this trip, (8, 9) is learned as a base. The problem generator's solution to Figure 4(a) is

$$(7, 7) \rightarrow (7, 9) \rightarrow (10, 9) \rightarrow (10, 6) \rightarrow (9, 6).$$

Figure 4(a) was Ariadne's fourth learning trip in this particular maze, and it had not yet encountered (8, 9) as a base. The next time Ariadne solves the same problem, it recognizes (8, 9) as a base, moves there from (8, 8), then goes to (9, 9), starts Roundabout, and ultimately produce a solution as short as the generator's path but in smaller steps (and therefore more decisions). Figure 4(c) shows Ariadne's useful knowledge about this maze after 10 level-4 learning problems in it.

5. Experimental design and results

Ariadne's performance is evaluated in a series of runs. A *run* for a fixed, randomly-generated maze consists of 10 learning problems given to a program, followed by 10 testing problems with learning turned off. A problem of either kind is terminated when the agent reaches the goal or when it has made 100 decisions. Because Ariadne is non-deterministic, results from 5 runs are averaged to produce an *experiment*. Experiments were performed for problems with levels of difficulty 6, 8, 10, and 12 in 20 × 20 mazes that were 30% obstructed internally. Effectively, the level of difficulty of a problem is the minimum number of (left or right) turns the robot must make to reach the goal.

Performance during testing is evaluated in a variety of ways: the number of test problems solved within the decision-step limit *n (solutions),* the Manhattan distance of a successful solution path *(path length),* elapsed time per trip in seconds on a Sun Sparc10 *(speed),* and the percentage of solutions that are at least as short as the one anticipated by the problem generator *(power)*.

Ariadne solves about 94% of all problems below level 12 in less than 100 decision steps. Speedup-learning is measured by reusing learning problems during testing. In recent testing on problems at levels 6, 8, 10, and 12, a non-parametric sign test showed that speedup paths are significantly shorter at the 95% confidence level. Ariadne solved 76% of level 12 problems on the first pass, and 86% when it resolved them with access to useful knowledge but with further learning turned off. On newly-generated testing problems, Ariadne solves some with learned pragmatic navigation knowledge that it cannot solve without it. With

learned pragmatic knowledge, it also constructs more solutions at least as good as the problem generator's than those constructed without pragmatic navigation knowledge and does so in less time, again at the 95% confidence level.

6. Discussion

In a sufficiently large and complex maze, people and many standard search techniques find these problems extremely difficult. The branching factor is large when there is enough obstruction to make the goal hard to see, but not too much, so that there are many choices at each decision point. Typical AI search strategies will explore most of the nodes, revisiting some of them many times. The robot's knowledge is so limited that search dependent upon an ordinary evaluation function is difficult to construct. For example, closer to the goal is not necessarily better; there may be a very long wall there. Best-first search, with Euclidean distance as its evaluation function, solves only 70% of level 10 problems [Pazzani, personal communication]. Depth-first search requires fairly elaborate backtracking and loop prevention; very few problems would be solvable with depth-first search under the $n = 100$ step limit imposed here. Breadth-first search, while it will always solve the problem, does so at the cost of visiting a high proportion of nodes in the search space and maintaining a very large structure for open paths. Indeed, the data indicate that explicit, breadth-first search in these mazes is nearly exhaustive (96.2%) for level-12 problems. Means-ends analysis is not possible because the robot knows little, if anything at all, about the immediate vicinity of the goal. For a very large maze, then, explicit search with a map would be extremely inefficient, perhaps intractable.

The thoughtful approach is to learn something about the maze on repeated trips through it. Branting and Aha recently suggested 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 (Branting & Aha 1995). Their mazes, although larger than those tested here, are less complex, since they assume that obstructions are rectangular objects. Such a maze is unlikely to present many dead-ends, narrow-necked chambers or bottles, or effective barriers between large regions. We would expect Ariadne to perform quite well without learning in those mazes. Alternatively, a variety of reinforcement learning techniques have sought convergence to an optimal path through repeated solution of a single problem, and obtained it after hundreds of thousands of repetitions [Moore and Atkeson, 1993; Sutton, 1990]. In contrast, Ariadne has no mechanism that would guarantee optimality, and will quickly settle upon the same route in most cases.

Pragmatic navigation proves to be a robust alternative. A single of item of pragmatic navigation knowledge can be applied differently by different Advisors, even within the same tier. For example, bases were conceived of as touchstones, intended for Home Run as a way to guide a single reactive decision. When simple planning was implemented in Leap Frog, bases were harnessed for that as well.

The application of pragmatic navigation knowledge, however, benefits from an empirical component. Leap Frog, for example, assumes orthogonally-aligned bases will have no intervening obstructions, and then builds a set of plans bridging the gap between $start_R$ and $start_G$ with a sequence of bases. Initially it was unclear whether these plans should be high-level (take the most distant aligned base from the search node) or low-level (take the nearest aligned base to the search node). During testing, low-level plans were clearly superior.

Although Ariadne draws no explicit map, one might fear that the amount of useful knowledge would eventually approach the amount required for a map. This does not appear to be the case. Apart from gates, there is no connectivity knowledge in pragmatic navigation. (In Figure 4(a) with 70 locations, however, there are 174 adjacencies.) In a particular maze there are not so many corridors, chambers, bottles, and gates to be discovered. Bases, which we believe to be Ariadne's most powerful item of useful knowledge, are a form of distinguished location. Although they may cluster together, bases tend to repeat, both because of maze topology and because they are learned from new solutions, which rely on old bases. Travel in a difficult problem almost always entails visits to both bases and non-bases, but to date bases have never exceeded 21% of the reachable locations, even after repeated learning in the same maze.

Based on preliminary empirical evidence, there is every reason to believe that Ariadne will scale up, i.e., that it will continue to perform well in much larger and more tortuous mazes than these. Hoyle, a FORR-based game-learning program, progressed from expertise in spaces with several thousand nodes to spaces with several billion nodes after the addition of only a few tier-2 Advisors (Epstein 1992). Ariadne has already performed well on preliminary tests in 30×30 mazes and continues to improve as we refine its Advisors and its learning algorithms.

Even if one could identify all the relevant useful knowledge for pragmatic navigation, there is no obvious way to integrate it properly. For example, it is our intuition that bases are much more helpful than chambers, so that when they disagree, the comments of Leap Frog and Home Run should be valued more than those of Chamberlain. In some mazes, however, the relative significance of Advisors differs because the topology of the maze makes one approach consistently (on a variety of problems) more successful than another. *AWL* is an algorithm that learns weights for second-tier Advisors [Epstein, 1994b]. It was devised to exploit empirical evidence that the accuracy of second-tier Advisors varies with the problem class. AWL learns weights to apply to comment strengths, so that voting results from Hoyle's Advisors match the way Hoyle's external exponent played, i.e., AWL fits Hoyle to the expert. In Ariadne, of course, there is no external expert, but there are procedures that smooth and eliminate repetitions from successful paths after a trip is over, *without search*. We are

now testing AWL to fit Ariadne's voting to these improved paths, and expect improvement.

This work has several similarities to the way people describe routes in urban and suburban Paris (Gryl 1994a; Gryl 1994b; Ligozat 1992). In particular, the data indicate that when people select routes they do not change direction a lot, move initially toward their goal, prefer main axes, and tend to avoid neighborhoods with limited access. Ariadne does not change direction a lot, either, because it has Advisors like Giant Step and Hurry with a propensity for long, one-directional steps. Plod, Mr. Rogers, and Giant Step all move the robot toward the goal, too, particularly early in a trip. Cork and Chamberlain deal in a human-like manner with the constricted neighborhoods of bottles and chambers. Finally, Ariadne's "vision" in only four directions permits only movement along the north-south and east-west axes.

Ariadne is ongoing work. There are still many proposed Advisors and items of useful knowledge on the drawing board. More elaborate planning is a central focus, as is a variety of training methods. Meanwhile the program's efficient and successful performance argues for navigation based on spatial representation of useful knowledge rather than detailed maps.

Acknowledgments

This work was supported in part by NSF#9423085. The author thanks Jack Gelfand, Glenn Iba, and Michael Pazzani for their thoughtful discussions.

References

Branting, L. K. & Aha, D. W. 1995. Stratified Case-Based Reasoning: Reusing Hierarchical Problem Solving Episodes. In Proceedings of the 14th International Joint Conference on Artificial Intelligence, 384-390. Montreal: Morgan Kaufmann.

Epstein, S. L. 1992. Prior Knowledge Strengthens Learning to Control Search in Weak Theory Domains. *International Journal of Intelligent Systems*, 7 : 547-586.

Epstein, S. L. 1994. For the Right Reasons: The FORR Architecture for Learning in a Skill Domain. *Cognitive Science*, 18 (3): 479-511.

Epstein, S. L. 1995. On Heuristic Reasoning, Reactivity, and Search. In Proceedings of the 14th International Joint Conference on Artificial Intelligence, 454-461. Montreal: Morgan Kaufmann.

Gryl, A. 1994a. Analyse cognitive des descriptions d'itinéraires. In Proceedings of the Premier Colloque Jeunes Chercheurs en Sciences Cognitives, La Motte d'Aveillans, France:

Gryl, A. 1994b. Opérations cognitives mises en oeuvre dans la description d'itinéraires, Mémoire de DEA, Paris IX, Orsay.

Klein, G. A. & Calderwood, R. 1991. Decision Models: Some Lessons from the Field. *IEEE Transactions on Systems, Man, and Cybernetics*, 21 (5): 1018-1026.

Korf, R. E. 1990. Real-Time Heuristic Search. *Artificial Intelligence*, 42 (2-3): 189-211.

Ligozat, G. 1992. Strategies for Route Description: an Interdisciplinary Approach. In Proceedings of the ECAI 92 Workshop W19, Spatial Concepts: Connecting Cognitive Theories with Formal Representations, Vienna: