A Fresh Look at Sensor-Based Navigation: Navigation with Sensing Costs

Zahy Bnaya and Ariel Felner
Information Systems Engineering
Deutsche Telekom Labs
Ben-Gurion University
Be'er-Sheva, Israel
{zahy,felner}@bgu.ac.il

Eyal Shimony
Computer Science,
Ben-Gurion University
Be'er-Sheva, Isrel
shimony@cs.bgu.ac.il

Gal A. Kaminka and Efi Merdler
The MAVERICK Group
Computer Science, Bar-Ilan University
Ramat-Gan, Israel
galk@cs.biu.ac.il

Abstract

Most work on navigation minimize travel effort or computational effort of the navigating agent, while assuming that unknown components of the environment are sensed by the agent at no cost. We introduce a framework for navigation where the agent needs to minimize a global cost function which includes both the travel cost and the sensing cost. At each point in time, the agent needs to decide whether to perform sense queries or to move towards the target. We develop the SN (Sensing-based Navigation) framework that utilizes heuristic functions to determine when and where to sense the environment in order to minimize total costs. We develop several such heuristics, based on the expected total cost. Experimental results show the benefits of our heuristics over existing work, and demonstrate the generality of the SN framework.

Introduction.

Navigation—finding a path through an unknown environment to a known goal—is a fundamental task in AI and robotics. The agent starts at a given position, and knows the position of the goal. The task is to efficiently reach the goal (as defined by a cost function). At every step, the task of the navigator is to find a path from its current location to the goal, until the goal is reached. In many circumstances the environment may change dynamically, and may be only partially known to the agent.

Previous papers on navigation falls into two categories: Some focus on minimizing the travel cost (Felner *et al.* 2004), while assuming a static environment and no global sensing abilities. Others allow for a changing environment, but assume that all changes are immediately visible to the agent (Stentz 1994; Koenig & Likhachev 2002a). Both categories ignore realistic navigation tasks for robots which allow for some a-priori knowledge of the environment (e.g., a map of roads), but where changes to the environment can only be sensed at a cost (e.g., by physically operating sensors).

We introduce a novel, more general problem formulation for navigation, which accounts for sensing costs, as well as physical travel cost. The task of the agent is to minimize a total cost function which includes both components. At each point of time, the agent needs to decide whether to move towards the goal, or perform sensing queries. We introduce the *Sensing-based Navigation* (SN) framework. SN models the task as navigation in a given weighted graph, where some of the edges may become untraversable, with some known probability. First, SN generates a path to the goal based on its current knowledge. Then, SN uses a policy, which allows the agent to decide between trusting just the default sensing and following the shortest path found (risking traveling into untraversable regions) on the ine hand, and sensing before moving, but paying the sensing cost, on the other hand.

This problem has several realistic scenarios, for example, an agent who buys a map of the city (G), knowing that some of the roads can be blocked by the police, However, the agent does not know which of the roads are blocked. The agent can perform some actions to get that information (such as calling the city info center, or sending a scout to a given place). These actions have costs which can be factored into its global cost function for the entire task. Another example could be a robot equipped with sensor devices, trying to navigate to a known goal in an uncertain terrain. The robot can choose to follow the path in its original map or operate the sensor devices to explore the path. Both traveling the path and using the sensor devices cost energy which the robot tries to minimize. This problem could also be relevant to on-line routing algorithms in big networks, instead of routing a packet through an existing routing table which could be out-dated, the sender of the packet can decide to check a certain path and fine-tune its current routing tables.

As computation of an optimal policy has a worst-case exponential run-time, SN uses heuristics to make this choice. We provide several heuristics based on the expected total cost and compare them to two brute-force heuristics which are equivalent to existing approaches to navigation. One hexisting heuristic never senses distant objects and thus navigates by following a pre-computed path until it fails. Another existing heuristic always senses all edges in its path ahead of time. We evaluate the benefits of using the expected cost heuristics across different types of sensing costs and travel costs, in systematic experiments.

Related Work

Much of the literature on navigation deals with physical robots that move in real environments. Surprisingly, it is usually assumed that sensing is free, and therefore only travel cost is considered. Some works deal with local navigation, where sensors have limited range, and thus can only sense within the robot's immediate environment, while others assume global sensing. See (Burgard *et al.* 2005) for an extensive survey; we focus here on closely related work.

Many navigation algorithms ignore the sensing cost (assuming it is sensing free and without range limits). The *agent-centered search* framework (Korf 1990; Koenig 2001) covers an important class of navigation algorithms, which interleave exploration and exploitation and address the combined plan-execution (travel) cost and the planning computation cost. These methods restrict the planning to the part of the domain around the agent; a step is selected and taken and the process repeats. These methods assume that information about unknown parts of the environment, or changes to it, are available with no sensing cost. In contrast, our framework considers both sensing and travel costs.

Another class of algorithms are the *incremental search* algorithms which handle dynamically-changing graphs. LPA* (Koenig & Likhachev 2002b) is activated every time the graph is changed, in order to find the current shortest path from the same given start and goal nodes. LPA* utilizes old data explored by previous runs, and thus computational effort is saved. D* (Stentz 1994) and D*-lite (Koenig & Likhachev 2002a) are intended for dynamically changing environments (e.g., where edges are continuously added and deleted). They modify LPA* to the case that the robot moves along the path and calculates a new shortest path from its new current location. These algorithms assume that graph changes are perceived by the agent with no cost.

An approach similar to our work is the *Shortest path discovery* framework (Szepesvri 2004). In that work, the task is to find the shortest path between two vertices, while assuming that edge costs are revealed by performing a special query. This work does not deal with online navigation as it is an offline shortest-path search task. It tries to minimize computation cost and the number of edge queries.

Our work is also closely related to the *Canadian traveler problem* (Bar-Noy & Schieber 1991). In the *Canadian traveler problem* (CTP), a traveler that has to travel from some location s to location t in some environment, represented as a weighted graph $G(V, E)$. However, this graph is unreliable, some of the roads might be blocked (due to snowfall for example). Finding such a blockage can only be done upon physically reaching an adjacent node. The task is to devise a traveling strategy which yields an optimal expected travel cost. The main extension in our current problem is in the way blocked roads are revealed. In our problem, finding a blockage can still be done by physically travelling to an adjacent node. However, our model allows agents to have the additional capability of remote sensing (e.g. the traveler can use his mobile telephone to call a friend who lives near the unreliable road to get additional information about its condition), thereby generalizing the CTP.

Partially Observable Markov Decision Processes (POMDP) offer a model for cost optimization of graph problems under uncertainty, where the objective is to find a policy (mapping from belief states, or observation histories, into actions) that provides optimal expected reward (Puterman 1994). While it is possible to model our problem as a POMDP (e.g., as unbounded-horizon POMDPs (Hansen 2007)), these problems are PSPACE-hard in the general case. Although some approximation techniques, such as point-based approximations, show promise (Pineau, Gordon, & Thrun 2003; Shani, Brafman, & Shimony 2006), the size of the state-space of our problem is beyond the current state of the art of POMDP approximation. This necessitates specialized algorithms as we propose. The action model we use is deterministic, and the uncertainty in world comes from the state of the graph alone. Like traditional POMDPs, our observation model could be stochastic.

A body of work exists on computing or approximating the value of information (Heckerman, Horvitz, & Middleton 1993), which is another way of looking at the problem under the decision-theoretic setting. These are not directly applicable to our problem; fitting them to our domain is one of the methods employed in this paper.

Problem Description

We concretely describe our problem as follows. We are given a connected directed graph $G = (V, E)$, a source vertex of the agent ($s \in V$), and a target vertex ($t \in V$). The task of the agent is to travel from s to t. For example, the agent is given a map of a city and its current position, and needs to plan its path to a given goal location.

However, we allow the input graph G to undergo a number of changes that are not known to the agent. In particular, in this paper we allow some of the edges in E to become *blocked* and thus become untraversable. Note that it is sufficient to deal only with blocking of *edges*, since a blocked vertex would have all of its incident edges blocked. Our framework can be easily extended to arbitrary edge weight distributions, thus allowing edges whose weight can change arbitrarily, rather than become completely blocked.

The current traversable graph is a subset of G called the *current graph* (CG) in this work. CG includes just the vertices and edges from G that are currently traversable. We assume that CG is otherwise fixed throughout the problem solving process, but that CG is *not* known to the agent. We also assume that for each edge e in G there is a *blocking probability* $p(e)$ (known to the agent) that e is blocked. For simplicity, we assume in this paper that the blocking events are independent, but this can be easily generalized.

Once the agent is situated in a vertex v it can perform actions of the following two types:
1) Sense: The agent can query the status of any given edge $e \in G$. This action is called $sense(v, e)$. We assume that sensing provides a correct answer, but incurs a *sensing cost* $c(v, e)$. As described below, the cost of sensing edges is domain dependent and might be a function of many attributes, such as the distance from v to e etc. In addition, we assume that when an agent is situated in a vertex it can see the status of each of its outgoing edges at no cost. For example, a road traveler can see with his eyes that the road it wishes to enter is blocked. This can be considered as *default sensing* or *local sensing*. Thus, we assume that default sensing is done any time the agent enters a new vertex. **2) Move:** The agent can move along any of the known traversable edges, outgo-

ing from v. This action incurs a *travel cost* which equals the weight of the edge.

The task of the agent is to reach the target while minimizing a total cost $C_{total} = C_{travel} + C_{sensing}$.

SN: Sensing-based Navigation

Sensing-based Navigation (SN) is a general scheme for navigating in physical graphs, while allowing for interleaved sensing actions at additional sensing cost. It allows plugging in different policies for deciding if and when to sense a remote edge.

In all of our algorithms, the agent maintains a belief graph (BG), initially a copy of G. Each edge e is annotated with the probably of traversablity, $p_t(e) = 1 - p(e)$. After a sensing query (either remote or local) edges are re-annotated according to the sensing outcome. If the outcome is that the edge is traversable it is re-annotated with $p_t(e) = 1$. If the outcome is that the edge is blocked, it is dropped from the graph ($p_t(e) = 0$). The belief graph is thus a special case of a POMDP *belief state*.

The agent first plans a path P on BG from v to t with any known path finding algorithm such as Dijkstra's algorithm or a heuristic search algorithm (e.g., one based on A*). We adopt the *free space assumption* from (Koenig, Smirnov, & Tovey 2003), meaning that every edge in BG is assumed to be traversable for this purpose. The agent can then attempt to traverse P, without sensing. However, since edges in BG may turn out to be blocked (i.e., they do not exist in CG), this may result in wasted travel cost if a blocked edge on P is physically reached. Alternatively, the agent may decide to interleave sensing actions into its movements to update BG prior to continuing along the path. If BG changes, the agent recalculates the shortest path to the goal without extra travel cost but at the additional cost of the sensing it performed.

SN works as follows. After path P is planned the agent first checks that the next edge is not blocked (local sensing). Then, it considers some remote sensing of edges in P. The decision on which edges to sense is done by a decision policy procedure. If no blocked edges are detected by either of these sensing queries, it moves one step along P and the process is repeated. If a blocked edge in P is detected, a new path to the goal is calculated and the process is repeated.

The pseudo-code for SN is presented in Algorithm 1, and works in three phases:

Calculate shortest path: In this phase a shortest path is calculated on BG from the current vertex v to the target vertex t. It is invoked by the *main* procedure in line 5.

Traverse the path: In this phase (invoked by the *main* procedure in line 6) the agent travels step by step along the path (line 14) until the target is reached. However, before the physical move, two checks are performed. First, if the next edge to follow is found (by the default local sensing) to be *blocked* (in CG) then the rest of the path becomes invalid. This might happen if the agent decided not to remotely sense this edge in previous steps. Control is passed back to procedure *main* and *CalculatePath* is invoked again on the new BG but from the new location to which the agent returned in line 11. The second check is to verify that the rest of the path is valid by performing some remote sensing. This is done by invoking the *VerifyPath* procedure (lines 12–13). Only then when no blocked edges are detected does the agent move along the first edge of the path (line 14).

Algorithm 1 Pseudo-code for SN

```
procedure main (Graph $G$, vertex $s$, vertex $t$)
{01}  $BG = G$;
{02}  $v = s$;
{03}  Update $BG$ based on local sensing;
{04}  while($v \neq t$) do
{05}     $P$=CalculatePath($BG$,$v$,$t$);
{06}     v=TraversePath($P$);
{07}  endwhile

vertex function TraversePath (Path $P$)
{08}  While $P \neq \emptyset$ do
{09}     $(x, y)$=pop first edge from $P$.
{10}     if($(x, y)$ is blocked)
{11}        return($x$);
{12}     if (VerifyPath($P$)==FALSE)
{13}        return($x$);
{14}     Traverse($x, y$) and place Agent at $y$;
{15}     Update $BG$ based on local sensing;
{16}  endwhile
{17}  return($t$);

bool function VerifyPath (Path $P$)
{18}  foreach($e \in P \mid e \in BG$)
{19}     if(shouldSense($e$))
{20}        r=sense($e$);
{21}        if(r==TRUE)
{22}           mark($e$) as probability 1
{23}        else
{24}           delete $e$ from $BG$;
{25}           return(FALSE);
{26}     endif
{27}  endfor
{28}  return(TRUE);
```

Verify the path: In this phase the agent verifies that edges of the calculated path are traversable. It is invoked by the *TraversePath* procedure in line 12. An edge in BG with traversability probability $p_t < 1$ is verified by performing a *sense* query for this edge (line 20). It is not mandatory in the framework to sense all the edges, and SN can choose to skip sensing some of these edges. The decision on whether to sense an edge is done by calling *ShouldSense* (line 19). The different sensing policies presented below differ in their decision of which edges of the path to sense while in this phase, and thus *ShouldSense* is a key function: different implementations determine the actual behavior of SN. We describe several such implementations below.

If one of the edges of the path are sensed as blocked, the path becomes invalid, BG is updated accordingly (lines 24–25) and the control is passed to procedure *main* where *CalculatePath* is invoked again for the new updated BG.

Brute-Force Sensing Policies

We now turn to describe the sensing policies. Perhaps the two simplest heuristics for *ShouldSense* are those that ignore the sensing costs altogether. These approximate many existing approaches to navigation.

The brute force *Never Sense* (NS) policy never senses any remote edges (*ShouldSense* always returns *False*). As local edges are sensed automatically and freely (with default sensing), NS approximates the approach of local navigation techniques in robotics, where a robot uses its body-mounted sensors to form an egocentric view of its surroundings, and navigates using this information. No sensing cost is ever incurred by NS, but it may lead to increased travel costs. NS works as follows. From any given location, the agent calculates the shortest path on BG and starts traversing that path without any sensing performed. The possible pitfall here is that the agent can reach an edge on the calculated shortest path in BG that turns out to be blocked. In this case the edge is removed from BG and a new shortest path is calculated on BG from its current location to the target vertex. This is done until the target is reached.

In contrast, the brute-force *Always Sense* (AS) policy takes the opposite direction. In AS, the agent queries **all** remaining edges in the path before it moves through it (*ShouldSense* always returns *True*). As soon as it discovers that an edge is known to be blocked, it recomputes the shortest path to the goal. Only after the entire path is proved to be traversable does the agent perform a sequence of moves along the calculated path. The AS policy optimally minimizes the travel cost, but may incur a large sensing cost. This behavior approximates the approach taken by global navigation algorithms (Stentz 1994; Koenig & Likhachev 2002a), which are designed to deal with dynamically-changing environments, but make the assumption that all changes in the environments are immediately available to the agent, at no cost.

Expected Total-Cost Policies

We now present heuristics to approximate the expected costs. The idea in considering whether and what to sense is traditionally based on *value of information*, or at the least some tractable approximation thereof (e.g., *myopic/single step value of information*), for a sensing operation. In the single-step assumption, we compute value of information (of sensing a given edge e) under the assumption that no additional information is gathered before the agent acts. However, in SN this type of approximation is infeasible, because "local sensing" information is gathered during traversal, even without additional sensing operations.

Residual Value of Information (RVOI) Suppose that the agent were given the state of the entire graph CG, except for that of edge e under consideration. What would be the value of knowing whether e is blocked in this case? This is essentially a computation of the *residual* value of information of knowing the state of edge e. Now suppose that there are k edges in BG whose exact status is unknown. There are 2^k different possible ways to turn BG into a fully known graph. Ideally, we should consider each of these possibilities in turn, calculate the value of information of sensing edges for it (as described below for the *improved expected* cost policy) and make the decision on whether to sense based on the entire set of 2^k graphs. Since this may be intractable, we further approximate the value of sensing by randomly generating n (=100 in our experiments) such graph samples. We calculate the costs and make the decisions based on these sampled graphs. We call this method the *residual value of information policy* (RVOI).

Expected Total Cost with Free Space Assumption A more computationally efficient heuristic adopts the *free space assumption* for the purpose of computing the value of sensing. The idea is to assume (for the sake of the decision making about sensing) that every edge in BG is traversable, except for the edge e under consideration. Thus, unlike RVOI above, we make our calculations using one graph, the one with the free space assumption.

To demonstrate the dilemma of whether to sense or not consider Figure 1 which shows an abstract situation in the course of running SN, where it needs to decide on a sensing query. The figure refers to the following components: (1) v is the current location of the agent; (2) $e = (x, y)$ is the edge under consideration; (3) in BG segments $v \rightarrow x$ and $y \rightarrow t$ are the shortest paths from v to x and from y to t, respectively; and (4) if e is blocked then segment $d(v)$ is the shortest path from v to t (in BG) and $d(x)$ is the shortest path from x to t.

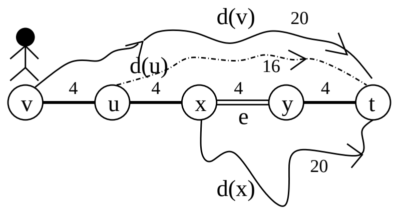

Figure 1: The expected cost heuristic.

In Figure 1, if e is traversable, the shortest path is (v, u, x, y, t) and its cost is 16. We label this path as $v \rightarrow t$. If e is blocked, the shortest path is $d(v)$ and its cost is 20. The agent may choose not to sense edge e from v and to take the risk and travel to x. If it then realizes (from x) that e is traversable the travel cost will be 16. However, if it finds e to be blocked it will need to use segment $d(x)$ with cost of 20 and the total travel cost will be 28.

Alternatively, the agent might choose to sense e from v, and pay the sensing cost. Then, if e is traversable the total cost will be 16 plus the sensing cost and if e is blocked, the total cost will be 20 (segment $d(v)$) plus the sensing cost.

The *expected cost* heuristic calculates the expected cost of these options is as follows:
$E(\neg sense(v,e)) = (\mathbf{1} - \mathbf{p}) \times (v \rightarrow t) + \mathbf{p} \times (v \rightarrow x + d(x))$
$E(sense(v,e)) = (\mathbf{1} - \mathbf{p}) \times (v \rightarrow t) + \mathbf{p} \times d(v) + SC(v,e)$
where $SC(v,e)$ is the cost of sensing e from v and p is the probability that edge e is blocked. The action with the minimal expected cost is chosen. In our example, suppose that $SC(v,e) = 2$ and that $p = 0.5$. The expected cost for sensing e from v is $E(sense(v,e)) = 0.5 \times 16 + 0.5 \times 20 + 2 =$

20. Likewise, the expected cost of not performing the sensing is $E(\neg sense(v,e)) = 0.5 \times 16 + 0.5 \times 28 = 22$. According to the expected cost policy the agent will choose to perform the sensing.

Improved Expected Total Cost Suppose that if e is blocked, the alternative path from v to t deviates from the original shortest path in vertex u (labeled $d(u)$ with cost 16 in the figure), and suppose that $SC(u,e) = 1$ (while $SC(v,e) = 2$). As shown above, it is worthwhile to perform the sensing in vertex v. However, observe that the outcome of the sensing will not influence the next action of the agent. This is because for both outcomes (whether e is traversable or blocked) the next move of the agent will be to go to vertex u. Only in u does the outcome of the sensing determine to where the agent will continue: to x if e is traversable, and the detour to t otherwise. Since the sensing cost from u is cheaper, the total cost will decrease if the agent waits and only performs the sensing at vertex u.

The *improved expected cost* policy accounts for such cases. This policy also calculates the expected cost of performing the sensing of e in all the vertices that appear later in the shortest path. It will choose to perform the sensing at the current vertex only if the expected cost of this sensing now is smaller than not sensing e at all, but is also smaller than waiting and performing this sensing at a later vertex.

Experiments

To evaluate SN, we have performed systematic experiments on a number of different sensing cost functions, using the different policies described earlier.

A sensing cost function for real world applications relies on many attributes of the environment and on the agent's capabilities. For example, in a city, a car driver might call up a roadside assistance agency which charges a constant amount per call and in each call only a fixed amount of information is given. In this case, the sensing cost is a predefined constant. If the sensing is done by a robot, then the sensing cost is proportional to the distance between the robot and the sensed item. Note also, that the total composite cost made up of travel and sensing cost might be defined in different units of measurements, and we need to bring them together into a common unit; use the constant c as a scaling coefficient below. In the experiments we define two possible sensing cost functions inspired by real-world settings:

- **Constant cost**. Here we assume that the cost of each sensing query is $SC(v,e) = c$ (e.g., a phone call). If $c = 0$ then the expected cost policies converge to the *always sense* policy, since there is nothing to lose by sensing. Similarly, if $c = \infty$ then the expected cost policies converge to the *never sense* policy, since any travel cost is smaller than a single query.

- **Distance cost**. Here we assume that the cost is proportional to the distance from the current location of the agent to the sensed item. In this work we used the distance from v to the closest vertex adjacent to $e = (x,y)$. Formally, $SC(v,e) = c \cdot \min(dist(v,x), dist(v,y))$.

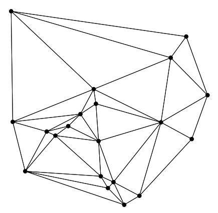

Figure 2: Delaunay graph of 20 nodes.

Our simulated environments were generated from Delaunay graphs (Okabe, Boots, & Sugihara 1992), over 1000 points generated with uniformly distribution in a square of 100x100. Our experimental setting aims to simulate graphs that could correspond to a small city. A Delaunay triangulation was constructed by creating a line segment between each pair of points (u,v) for which there exists a circle passing through u and v that encloses no other point. This causes each point to be joined by a line segment to each of its nearest neighbors, but not to other points.

Figure 2 illustrates a 20-node Delaunay graph. The experiments were performed as follows. First, a Delaunay graph G of 1000 vertices was created by placing the vertices in a square of size 100×100 units. Then, two of its vertices were randomly selected as the start and target vertices. Now each edge of the graph was blocked with a constant fixed probability BP (of 0.1, 0.3 , 0.5 and 0.6) to attain the current graph CG (varying edges probabilities would not change the behavior). We ran the different policies on each of these 100 cases and report the average cost below.

Results

Tables 1-2 give the results of the different techniques and for different sensing cost functions. Sections in the tables, correspond to a different cost and a different scaling factor c. Within each section, the tables are separated by double-lines into groups of three columns, titled *travel*, *sense*, and *total* which refer to costs. Each such group of three is associated with a value of the blocking probability BP (0.1, 0.3, 0.5, 0.6). There are five rows of values in each part, corresponding to the five policies discussed in the paper: *Never sense* (NS), *Residual Value of Information* (RVOI), *Expected cost* (EXP), *Improved expected cost* (I-EXP) and *Always sense* (AS). The important column is of course the *total cost* column.

In general, the *I-Expected* policy outperformed the *expected* and RVOI policies which are based on the same principle. The RVOI policy tends to grossly underestimate the value of information of sensing operations resulting in a much worse travel cost and turned to behave almost similar to the cheap and simple NS policy. Thus, the current version of RVOI is an impractical policy due to its large computational overhead.

Constant sensing cost: Section 1 of table 1 corresponds to a small constant cost of 0.01 (with a map size of 100×100). In this case, the relative cost of sensing is much cheaper than the cost of traveling. Thus, there is almost no reason not to sense a questionable edge. Indeed, the AS

Constant sensing cost of c=0.01												
Policy	**travel**	**sense**	**total**	**travel**	**sense**	**total**	**travel**	**sense**	**total**	**travel**	**sense**	**total**
BP	p=0.1			p=0.3			p=0.5			p=0.6		
Never	57.54	0.00	57.54	70.06	0.00	70.06	115.56	0.00	115.56	286.45	0.00	286.45
RVOI	56.24	0.36	56.60	62.81	0.92	63.73	112.80	0.26	113.06	286.45	0.00	286.45
Exp	56.05	0.33	56.38	61.53	1.21	62.73	74.90	3.37	**78.27**	110.05	7.90	117.94
I-Exp	56.05	0.31	56.36	61.26	0.98	62.24	79.74	2.49	82.23	123.50	5.81	129.31
Always	55.96	0.36	**56.32**	60.33	1.85	**62.18**	72.09	7.86	79.95	97.31	20.33	**117.64**

Constant sensing cost of 0.1												
Policy	**travel**	**sense**	**total**	**travel**	**sense**	**total**	**travel**	**sense**	**total**	**travel**	**sense**	**total**
BP	p=0.1			p=0.3			p=0.5			p=0.6		
Never	57.54	0.00	**57.54**	70.06	0.00	70.06	115.56	0.00	115.56	286.45	0.00	286.45
RVOI	57.08	0.60	57.69	65.53	5.00	70.53	112.50	2.17	114.67	286.45	0.00	286.45
Exp	56.37	1.43	57.80	61.89	9.33	71.22	80.26	31.77	112.03	122.51	79.23	201.74
I-Exp	56.37	1.37	57.74	62.15	7.84	**69.99**	81.89	21.25	**103.14**	134.81	54.91	**189.72**
Always	55.96	3.56	59.52	60.33	18.49	78.82	72.09	78.65	150.74	97.31	203.34	300.65

Constant sensing cost of 3												
Policy	**travel**	**sense**	**total**	**travel**	**sense**	**total**	**travel**	**sense**	**total**	**travel**	**sense**	**total**
BP	p=0.1			p=0.3			p=0.5			p=0.6		
Never	57.54	0.00	57.54	70.06	0.00	70.06	115.56	0.00	**115.56**	286.45	0.00	**286.45**
RVOI	57.54	0.00	57.54	70.03	0.06	70.09	115.55	0.54	116.09	286.45^*	0.00^*	286.45^*
Exp	57.54	0.00	57.54	69.94	0.03	69.97	116.63	2.45	119.09	293.15	20.30	313.45
I-Exp	57.54	0.00	**57.54**	69.94	0.03	**69.97**	116.55	2.64	119.19	293.28	19.15	312.43
Always	55.96	106.79	162.75	60.33	554.64	614.97	72.09	2359.42	2431.51	97.31	6100.18	6197.49

Table 1: The best sensing policy in each group is highlighted in **bold** * marks an approximate value, based on NS.

policy resulted in the lowest total cost for almost all values of BP. Note however, that the *I-Expected* policy did relatively very well. When the blocking probability BP increases more work is needed by all policies as more dead-ends are introduced. The NS policy incurs the extra work in traveling. Since the relative travel cost is high its total cost increases dramatically. The AS and the *I-Expected* policies incur much of the extra work in sensing cost and their total cost does not increase significantly because of the cheap sensing.

The second section of table 1 corresponds to a constant cost of 0.1. In this case, the relative cost of sensing is within the same range of the cost of traveling. The table shows that for all values of BP AS and NS incur relatively similar total costs. In the cases where $BP > 0.1$ the *I-Expected* policy outperforms all other policies in its total cost. For $BP = 0.1$ it was nearly equal to the NS policy.

Finally, the third section of the table corresponds to an extremely expensive constant cost of 3. In this case, it is not worthwhile to sense and the table clearly shows that in all cases of BP, NS significantly outperforms AS in terms of total cost. With BP of 0.1, 0.3 both the *Expected* and *I-Expected* policies are equal or even better than the NS policy which means that they choose to sense rarely. It is surprising to note that when $BP \geq 0.5$ the expected cost policies are a little worse than the NS policy, not just in their sensing costs, but also in their travel costs, resulting in an inferior total cost. The pitfall is the *free space assumption*. These policies assume that all the other edges exist, but with a large blocking probability this is a bad assumption. The travel cost of *I-Expected* is larger because if a given sensing returned that the edge is blocked, then a detour is suggested but that detour can later prove invalid.

Costs proportional to distance: Table 2 also shows results for the distance cost function. The top part corresponds to coefficient of 0.01 while the bottom part corresponds to a coefficient of 0.04. In all the possible cases, NS and RVOI were better than AS, but both expected costs policies systematically outperformed the two brute force polices in their total costs. In this case the expected cost policies tend to sense nearer edges. This proves useful as the error of the free space assumption is smaller.

Summary and Conclusions

We introduced the sensing-based navigation problem where the task is to navigate to a target when trying to minimize a compound total cost of both travel and sensing. We presented SN, the Sensing-based Navigation algorithm, a general framework that solves this problem. We then proposed several heuristic policies for SN, and have investigated their use in hundreds of trials, and under a variety of conditions.

The *improved expected* cost policy turned out to outperform all other policies on the vast majority of test cases. Only in the constant sensing cost where the sensing is extremely cheap or extremely expensive together with extreme large values of p did one of the brute force policies do better. However, even in these cases the *I-Expected* cost policy—based on the free space assumption—is not significantly worse. This is encouraging as this policy is computationally very cheap.

Future work will continue in the following directions. More sensing cost variation can be used. For example, we can assume that the sensing cost is proportional to the weight of the edge. Similarly, inspired by rules of physics, the sens-

Sensing cost of distance * 0.01												
Policy	**travel**	**sense**	**total**	**travel**	**sense**	**total**	**travel**	**sense**	**total**	**travel**	**sense**	**total**
BP	p=0.1			p=0.3			p=0.5			p=0.6		
Never	57.54	0.00	57.54	70.06	0.00	70.06	115.56	0.00	115.56	286.45	0.00	286.45
RVOI	57.29	0.29	57.58	65.51	2.74	68.25	114.50	2.56	117.06	286.45	0.00	286.45
Exp	56.49	1.20	57.69	63.03	13.14	92.17	83.89	48.35	132.24	142.20	141.27	283.47
I-Exp	56.57	0.52	**57.08**	64.06	2.12	**66.18**	95.40	6.31	**101.71**	190.88	14.65	**205.52**
Always	55.96	10.02	65.98	60.33	48.68	109.01	72.09	207.35	279.43	97.31	549.76	647.07

Sensing cost of distance * 0.04												
Policy	**travel**	**sense**	**total**	**travel**	**sense**	**total**	**travel**	**sense**	**total**	**travel**	**sense**	**total**
BP	p=0.1			p=0.3			p=0.5			p=0.6		
Never	57.54	0.00	57.54	70.06	0.00	70.06	115.56	0.00	115.56	286.45	0.00	286.45
RVOI	57.48	0.14	57.62	68.29	1.59	69.88	109.66	1.57	111.24	269.49	0.00	269.49
Exp	57.17	0.39	57.55	65.02	6.22	71.24	98.09	35.94	134.03	204.11	126.40	330.51
I-Exp	57.16	0.30	**57.46**	65.93	2.29	**68.22**	100.03	8.19	**108.22**	221.61	24.47	**246.08**
Always	55.96	40.10	96.06	60.33	194.71	255.04	72.09	829.39	901.47	97.31	2199.04	2296.35

Table 2: Results with different distance sensing costs. The best sensing policy in each group is highlighted in **bold**.

ing cost can be proportional to the distance squared. Second, more policies can be developed to handle dependencies of the existence of the sensed edge with other edges. Another direction is attempting to find the truly optimal policy, but we believe that this will prove intractable. Third, other possible combined costs (e.g., fuel costs, energy consumption, computation cost etc.) might be considered. Fourth, we might consider solving other graph problems (not necessarily navigating) under this general framework. Finally, the last direction is to generalize this to the multi-agent case where a team of agents need to cooperate when solving such problems. Here, other costs such as communication costs might be added to the total cost.

Acknowledgments

This research was supported by the Israel Science Foundation (ISF) under grant number 728/06 to Ariel Felner and by Deutsche Telekom Labs in Ben-Gurion University.

References

Bar-Noy, A., and Schieber, B. 1991. The canadian traveller problem. In *SODA '91: Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms*, 261–270. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

Burgard, W.; Moors, M.; Stachniss, C.; and Schneider, F. 2005. Coordinated multi-robot exploration. *IEEE Transactions on Robotics*.

Felner, A.; Stern, R.; Ben-Yair, A.; Kraus, S.; and Netanyahu, N. 2004. PhA*: Finding the shortest path with A* in unknown physical environments. *Journal of Artificial Intelligence Research* 21:631–679.

Hansen, E. A. 2007. Indefinite-horizon POMDPs with action-based termination. In *AAAI*, 1237–1242.

Heckerman, D.; Horvitz, E.; and Middleton, B. 1993. An approximate nonmyopic computation for value of information. *IEEE Trans. Pattern Anal. Mach. Intell.* 15(3):292–298.

Koenig, S., and Likhachev, M. 2002a. D* lite. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)*, 476–483.

Koenig, S., and Likhachev, M. 2002b. Incremental A*. In *Advances in Neural Information Processing Systems 14 (NIPS)*. MIT Press, Cambridge, MA.

Koenig, S.; Smirnov, Y.; and Tovey, C. 2003. Performance bounds for planning in unknown terrain. *Artificial Intelligence Journal* 147(1-2):253–279.

Koenig, S. 2001. Agent-centered search. *Artificial Intelligence Magazine* 22(4):109–131.

Korf, R. E. 1990. Real-time heuristic search. *Artificial Intelligence* 42(3):189–211.

Okabe, A.; Boots, B.; and Sugihara, K. 1992. *Spatial Tessellations, Concepts, and Applications of Voronoi Diagrams*. UK: Wiley, Chichester.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: An anytime algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1025 – 1032.

Puterman, M. 1994. *Markov Decision Processes*. New York: Wiley.

Shani, G.; Brafman, R. I.; and Shimony, S. E. 2006. Prioritizing point-based POMDP solvers. In Fürnkranz, J.; Scheffer, T.; and Spiliopoulou, M., eds., *ECML*, volume 4212 of *Lecture Notes in Computer Science*, 389–400. Springer.

Stentz, A. 1994. Optimal and efficient path planning for partially-known environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 3310–3317.

Szepesvri, C. 2004. Shortest path discovery problems: A framework, algorithms and experimental results. In *Proceedings of AAAI-2004*, 550–555.