INFORMATION AND COMPUTATION 106, 234–252 (1993)

Searching in the Plane

RICARDO A. BAEZA-YATES*

Department of Computer Science, Universidad de Chile, Casilla 2777, Santiago, Chile

JOSEPH C. CULBERSON†

Department of Computing Science, University of Alberta, Edmonton, Alberta T6G 2H1, Canada

AND

GREGORY J. E. RAWLINS

Department of Computer Science, Indiana University, 101 Lindley Hall, Bloomington, Indiana 47405

In this paper we initiate a new area of study dealing with the best way to search a possibly unbounded region for an object. The model for our search algorithms is that we must pay costs proportional to the distance of the next probe position relative to our current position. This model is meant to give a realistic cost measure for a robot moving in the plane. We also examine the effect of decreasing the amount of *a priori* information given to search problems. Problems of this type are very simple analogues of non-trivial problems on searching an unbounded region, processing digitized images, and robot navigation. We show that for some simple search problems, knowing the general direction of the goal is much more informative than knowing the distance to the goal. © 1993 Academic Press, Inc.

1. INTRODUCTION

The problems considered in this paper were suggested by general problems in graph searching [15], finding optimal paths [3, 17], boundary detection in digital images [7], and robotic navigation. When searching a graph or maze we usually assume that we have some representation of the maze or graph. However, in the real world, we may not have a complete representation, as is reasonable, for example, if we wish to have a robot

* Also supported by the Institute for Computer Research at the University of Waterloo, Ontario, Canada.

† Supported by Natural Sciences and Engineering Research Council Grant A-8053.

explore an unknown building or if we are playing a computer maze game. In cases of incomplete information in potentially unbounded domains how can we best search the domain? Further, some non-geometric searching problems can be phrased as geometric search problems in unbounded domains. For example, consider the problem of searching sequentially for a record which is known to be on one of m large tapes given that we have only one tape drive and that we must rewind the current tape before searching any other. How can we minimize the time needed to find the record assuming that the tapes are so large that it is impractical to search any one tape completely before searching any other tape? (This problem is solved in Section 2.)

Bentley and Yao [4] constructed an almost optimal algorithm to find an integer chosen from an unbounded set of positive integers. The problems we consider in this paper differ from theirs in that we have to pay costs proportional to the distance of a probe whereas they assume random access to any location. Karp *et al.* [15] consider "wandering RAMs" with bounded memory searching binary trees. For them the number of node visits was the cost measure; this problem is closer in spirit to the class of problems we consider here.

All problems considered in this paper are of the following form: we are searching for an object in some space under the restriction that for each new "probe" we must pay costs proportional to the distance of the probe position relative to our current probe position and we wish to minimize this cost. This is meant to model the cost in real terms of a robot (or human) searching for an object when the mobile searcher must move about to find the object.

To make this concrete, suppose that we are at the origin in the plane and we are searching for a line. Suppose that the line is distance n steps (we use steps as our metric) away from the origin.

- Given a normal to the line we can find the line in n steps.
- Given the line's distance and slope we can find the line in $3n$ steps.
- Given the line's distance and that the line is horizontal or vertical we can find the line in $3\sqrt{2}n \approx 4.24n$ steps [1].
- Given the line's distance we can find the line in $(1+\sqrt{3}+7\pi/6)\,n \approx 6.39n$ steps [13].
- Given the line's slope we can find the line in $9n$ steps.
- Given that the line is horizontal or vertical we can find the line in $13.02n$ steps [1].
- Given nothing at all we can find the line in $13.81n$ steps [1].

Except for the last two, all of the above results are provably optimal up to lower order terms. These results are representative of the gradation in cost as information about the target decreases.

Searching for a line of arbitrary slope a known distance away in the plane was posed by Bellman [3] and was solved by Isbell [13]; Melzak [16] has claimed a solution, but this solution is incorrect, giving a bound of 6.459... instead of 6.397... . The first two results are trivial; we present the fifth result and some generalizations in the next section, and summarize the remaining results in Section 4.

In this paper we begin with search problems in the plane. We distinguish between the cases in which the robot knows the distance (measured in steps) to the object and the cases in which it does not know the distance. In the first case, our bounds are functions of the known distance n; in the second case, our bounds are ratios of the distance walked divided by the (unknown) distance to the object, n. In all cases we assume that the robot starts at the origin; that the robot can only recognize the object when directly upon it; and that the object is an integer number of steps away from the robot (none of these restrictions lose generality). Finally, in all but one problem, we are only concerned with the worst case.

2. Searching for a Point on a Line

Suppose that the robot needs to find some distinguished point on a line. Assume that the point is n steps away along the line. If the robot knows that the point is to its left (or to its right), whether or not it knows the actual distance to the point, then it can optimally find the point in n steps. If the robot knows that the point is n steps away but not whether the point is to its left or right, then it is easy to show that the obvious algorithm is also optimal: Go left for n steps, then turn and go right for $2n$ steps.

2.1. *Point Arbitrarily Far Away*

Suppose that the robot does not know how far away the point is. What is the minimum number of steps it must make to find the point as a function of n? For this first, and simplest, problem we spend some time developing the basic ideas and manipulations since they are used several times.

Any algorithm to solve this problem can be described as a function, f, where $f(i)$ is the number of steps it makes to the left (or right) before the ith turn and where the odd terms are the number of steps to the left and the even terms are the number of steps to the right as measured from the origin. That is, starting at the origin, the robot walks $f(1)$ steps to the left, turns and returns to the origin, then walks $f(2)$ steps to the right, etc. Observe that if the robot is to find the point, then f must be such that

$$f(i) \geqslant f(i-2)+1 \qquad \forall i \geqslant 1, \text{ where } f(-1)=f((0)=0.$$

Linear Spiral Search. Execute cycles of steps where the function determining the number of steps to walk before the ith turn starting from the origin is

$$f(i)=2^i \qquad \forall i \geqslant 1.$$

(We will see later why this algorithm is called linear spiral search.) The total distance walked is $2\sum_{i=1}^{\lfloor \log n \rfloor + 1} 2^i + n$, which is no more than $9n$ steps. It is straightforward to show that this bound of $9n$ steps is achieved by an infinite class of algorithms.

THEOREM 2.1. *Linear Spiral Search is optimal up to lower order terms.*

Proof. Let the point be found after the $(i+1)$th turn and before the $(i+2)$th turn for some i. That is, let i be such that $f(i)+1 \leqslant n < f(i+2)$. The worst case ratio of the total distance walked divided by the distance to the point is then

$$\max_{i \geqslant 1}\left(\frac{2\sum_{j=1}^{i+1} f(j)+f(i)+1}{f(i)+1}\right)=1+2\max_{i \geqslant 1}\left(\frac{\sum_{j=1}^{i+1} f(j)}{f(i)+1}\right). \tag{1}$$

Since we already know that a $9n$ algorithm is possible suppose that f is such that

$$\frac{\sum_{j=1}^{i+1} f(j)}{f(i)+1} \leqslant c \qquad \forall i \geqslant 1, \tag{2}$$

where c is a constant. A lower bound on c yields a lower bound of $(2c+1)n$ steps for the problem (from Eq. (1)). We now show that c must be at least 4, from which it follows that any $9n$ algorithm is optimal up to lower order terms.

First, from Inequality (2) it follows that

$$c > 1+f(i+1)/(f(i)+1).$$

Since f is strictly monotone increasing for even or odd i we can choose a sufficiently large i so that $\sum_{j=1}^{i} f(j) - c > f(i) + 1$. For a fixed and sufficiently large i

$$\begin{aligned} c[f(i+k)+1] &\geqslant \sum_{j=1}^{i+k+1} f(j) \\ cf(i+k) &\geqslant \sum_{j=1}^{i+k+1} f(j) - c \\ &\geqslant \sum_{j=1}^{i} f(j) - c + \sum_{j=1}^{k+1} f(i+j) \end{aligned}$$

and thus c must also satisfy the following infinite set of inequalities:

$$f(i+k) > \frac{f(i) + 1 + \sum_{j=1}^{k+1} f(i+j) - f(i+k)}{c-1} \qquad \forall k \geqslant 1.$$

Together with the previous inequality on c this system of inequalities may be solved inductively for each k by deleting the $f(i+k)$ term on the right hand side, substituting the derived bound on $f(i+k)$ into the inequality for $f(i+k-1)$ and using that bound on $f(i+k-1)$, and so on.

As an example, here are the first two steps of this bounding process. For ease of description we change variables to the normalized function $h(j) = f(j)/(f(i)+1)$. For $k = 0$ we have that

$$c > 1 + h(i+1) > 1$$

Therefore, $c > 1$. This implies a lower bound of $(2c+1)n = 3n$.

For $k = 1$ we have that

$$h(i+1) > \frac{1 + h(i+2)}{c-1} > \frac{1}{c-1}.$$

Therefore,

$$c > 1 + h(i+1) > 1 + \frac{1}{c-1}.$$

This implies that

$$c^2 - 2c > 0.$$

Therefore, $c > 2$ (implying a lower bound of $5n$). And so on inductively.

In general, c must be such that the following polynomials are all positive:

$$g(k) = c^{k-1} \sum_{j=0}^{\infty} \binom{k-j}{j} (-1/c)^j.$$

The minimal value of c for which each of these polynomials is greater than zero bounds c from below.

These polynomials obey the recurrence

$$g(k) = cg(k-1) - cg(k-2)$$

which has characteristic equation

$$\lambda^2 - c\lambda + c = 0.$$

This equation has roots

$$\frac{c \pm \sqrt{c^2 - 4c}}{2}.$$

If the roots are distinct then

$$g(k) = \frac{1}{c\sqrt{c^2-4c}}\left(\left(\frac{c+\sqrt{c^2-4c}}{2}\right)^{k+1} - \left(\frac{c-\sqrt{c^2-4c}}{2}\right)^{k+1}\right)$$

and this function is positive for all $k > 0$ if and only if $c > 4$.

Alternately, if the roots are equal $(c = 4)$ then

$$g(k) = (k+1)\,2^{k+2}$$

and this function is positive for all $k > 0$.

Therefore any algorithm to find a point on a line an unknown distance, n, away must take at least $9n$ steps. ∎

Note that this is a lower bound on the *constant multiple* of the distance walked to the actual distance to the point. In fact there exist algorithms which take no more than $9n - \Theta(\lg n)^i$ steps for any i.

2.2. *The Average Case for a Point a Bounded Distance Away*

Suppose that the robot knows that the point is within n steps and that it is distributed uniformly about the interval of length $2n$ centered on the origin. Then the naive algorithm is also the best average case algorithm, with an average distance of $3n/2$. However, if the robot knows that the point is likely to be near the origin, then it might want to turn after a smaller number of steps, since if it goes very far from the origin the probability of finding the point further on is much less than that if finding it near the origin on the other side. We show below that the optimal average case algorithm for most distributions, including bounded domains, has an *infinite* number of turning points! Intuitively, this happens because there is always a point at which it is better to turn back and look at ranges on the other side of the origin where the probability of finding the point is greater.

For clarity in the following proof, we normalize over the range $[-1, 1]$.

THEOREM 2.2. *Let $f(x)$ be a density function over the range $[-1, 1]$ with right and left tail distributions $F_r(t)$, $F_\ell(t)$. Suppose we have points $-t_1$ and t_2 such that*

$$\frac{F_r(t_2)}{F_\ell(-t_1)} < \frac{1-t_2}{1+t_2} \qquad \text{where} \quad 0 < t_1 < t_2 < 1.$$

If the last turn was at the point $-t_1$, then the average distance travelled if we turn at the point t_2 is less than the average distance travelled if we do not turn at the point t_2.

Proof. Suppose we are at the point t_2 proceeding right, and the last turn was at the point $-t_1$. That is, we have not found the point yet. Let d be the remaining distance we travel to find the point p. Thus d depends on the search strategy we use, as well as the probability distribution of p. The expected distance traveled if we turn at t_2 is

$$E_T[d] = E_T[d \mid p > t_2]\, F_r[t_2] + E_T[d \mid p < -t_1]\, F_\ell(-t_1).$$

Note that the only possibilities under the stated assumptions are $-1 \leqslant p < -t_1$ and $t_2 < p \leqslant 1$. But $E_T[d \mid p < -t_1] = t_1 + t_2 + E[-t_1 - p \mid p < -t_1]$ and $E_T[d \mid p > t_2] = 2(1 + t_2) + E[p - t_2 \mid p > t_2]$ (assuming that we only turn at -1 on failing to find the point to the left of $-t_1$). Thus

$$\begin{aligned} E_T[d] = {} & (t_1 + t_2 + E[-t_1 - p \mid p < -t_1]) \\ & \times F_\ell(-t_1) + (2 + 2t_2 + E[p - t_2 \mid p > t_2])\, F_r(t_2). \end{aligned}$$

Alternatively, if we do not turn then by a similar breakdown

$$\begin{aligned} E_N[d] = {} & (2 + t_1 - t_2 + E[-t_1 - p \mid p < -t_1]) \\ & \times F_\ell(-t_1) + E[p - t_2 \mid p > t_2]\, F_r(t_2). \end{aligned}$$

It is better to turn if $E_t[d] < E_N[d]$; that is, when

$$\begin{aligned} & (t_1 + t_2 + E[-t_1 - p \mid p < -t_1])\, F_\ell(-t_1) \\ & \quad + (2t_2 + 2 + E[p - t_2 \mid p > t_2])\, F_r(t_2) \\ & \quad < (2 + t_1 - t_2 + E[-t_1 - p \mid p < -t_1])\, F_\ell(-t_1) \\ & \quad + E[p - t_2 \mid p > t_2]\, F_r(t_2), \end{aligned}$$

from which the result follows by simple manipulation. ∎

COROLLARY 2.1. *If for each $0 < t_1 < 1$, there exists a t_2 such that the conditions of the previous theorem are satisfied, and the density function $f(x)$ is symmetric with respect to the origin, then the optimal average case search algorithm has infinitely many turning points.*

Proof. If there exist such points t_1 and t_2, then there also exists a point $1 > t_3 > t_1$ such that

$$\frac{F_l(t_3)}{F_r(t_2)} < \frac{1-t_3}{1+t_3}$$

because $F_l(t) = F_r(t)$. Repeating this argument we obtain an infinite number of turns. ∎

A similar result may be proved for any distribution depending on the characteristics of the tail distributions. Suppose, for example, that the point has a triangular distribution given by

$$p(x) = \left|\frac{n-x}{n^2}\right| \qquad \forall -n \leqslant x \leqslant n.$$

In this case, the search space is finitely bounded, yet the theorem can easily be seen to apply.

2.3. *Searching for a Point in m Concurrent Rays*

Suppose that the robot is at the meeting point of m rays and that the robot has to find a point at distance n away on some one of the rays with the restriction that the robot can only travel along a ray (for example, the rays may represent rails or corridors). If the robot knows the distance to the point then it has an optimal $(2m-1)n$ algorithm as can be shown by a straightforward argument.

Suppose that the robot does not know the distance to the point. If $m = 1$ then the robot finds the point in n steps. If $m = 2$ then we have the equivalent of searching for a point on a line (Section 2.1) and so the robot finds the point in $9n$ steps.

It is straightforward to show that the robot need only visit the rays cyclically since there is no advantage to favouring one over another. No other order can improve the worst case. Let the rays be numbered in order of visits (assuming a cyclic visiting pattern) 1, 2, ..., m, where $m \geqslant 2$. Let $f(i)$ be the distance moved counting from the origin before the ith turn. In order to guarantee finding the point, f must be such that

$$f(i) \geqslant f(i-m) + 1 \qquad \forall i \geqslant 1 \text{ where } f(-j) = 0 \ \forall 0 \leqslant j \leqslant m-1$$

Generalized Linear Spiral Search. Execute cycles of steps where the function determining the number of steps to walk before the ith turn starting from the origin is

$$f(i) = \left(\frac{m}{m-1}\right)^i \qquad \forall i \geqslant 1.$$

The worst case ratio is then

$$1 + 2\frac{m^m}{(m-1)^{m-1}} \qquad \text{(for large } m\text{)}.$$

Thus to search two concurrent rays (equivalently, a line) we use increasing powers of 2, to search three concurrent rays we use increasing powers of $\frac{3}{2}$, and so on. Note that if the rays are ordered uniformly in the plane then the turning points of generalized linear spiral search describe the intersection points of a logarithmic spiral (hence its name). Finally, recall that we assume the point to be an integral number of steps away, so the last turn may be a fraction of a step in excess.

THEOREM 2.3. *Generalized Linear Spiral Search is optimal up to lower order terms.*

Proof. Let the point be found after the $(i+m-1)$th turn and before the $(i+m)$th turn. The worst case ratio is

$$1 + 2\max_{i \geqslant 1}\left(\sum_{j=1}^{i+m-1} f(j)/(f(i)+1)\right).$$

Let c be a constant such that

$$\sum_{j=1}^{i+m-1} f(j)/(f(i)+1) \leqslant c \qquad \forall i \geqslant 1.$$

As in the lower bound analysis of the straight line search problem it is possible to construct an infinite sequence of functions of c each of which must be positive. The positivity of each function places bounds on how small c can be.

The functions are

$$g(k) = c^{k-1}\sum_{j=0}^{\infty}\binom{k+m-2-(m-1)j}{j}(-1/c)^j.$$

These functions obey the recurrence

$$g(k) = cg(k-1) - c^{m-1}g(k-m).$$

This recurrence has characteristic equation

$$\lambda^m - c\lambda^{m-1} + c^{m-1} = 0.$$

This equation has a positive real double root at $c = m^m/(m-1)^{m-1}$, namely $\lambda = c(m-1)/m$. All other roots are negative or imaginary. ▮

3. Searching for a Point in a Lattice

Suppose that a robot has to find a point in a rectangular grid in the plane and that the robot can move left, right, up, or down in one step. We can think of this as an exploration of a simple rectangular maze with no barriers. Call the points of the lattice which lie n steps from the origin the *reference diamond*. This is of course a circle under the $\mathscr{L}_1$ metric. We say a point is *within* a reference diamond n if it is at distance k, $0 \leqslant k \leqslant n$. We use compass bearings (north N, east E, south S, and west W) to describe algorithms.

If the robot knows that the point is exactly n steps away, then it moves directly to the northernmost point at distance n, then follows a zigzag path that visits all the nodes at distance n in sequence. The total number of steps is $9n-2$, which is optimal.

Suppose that the robot does not know how far away the point is. As a function of the unknown distance n, the worst case behavior of any algorithm is evoked by an adversary that places the point at the last point visited at distance n by the algorithm. Counting the number of points in the reference diamond, we get a trivial lower bound of $2n^2 + 2n$ steps for any algorithm.

Suppose, without loss of generality, that any algorithm always begins by going north. The simple spiral, which fills in a square centered on the origin, requires $4n^2 + 3n$ steps to visit the last square at distance n, directly west of the origin. (It visits all the points in a $2n+1$ by $2n+1$ square, except those directly north of the point at distance n west of the origin.) This is nearly twice the lower bound, because the reference diamond encloses about half of this square.

We can modify the spiral to yield a path that more closely follows the boundary of a reference diamond on each cycle. However, one problem we inevitably encounter is that we must visit points either further or closer to the origin between every pair of points at distance n. That is, each cycle of the spiral must visit nodes of two (at least) adjacent reference diamond boundaries. This fact is used to establish the following improved lower bound.

THEOREM 3.1. *Any algorithm which can find a point at some unknown finite distance n in the lattice requires at least $2n^2+4n+1$ steps.*

Proof. Consider any algorithm $\mathscr{A}$. Let $A(n)$ be the number of steps required by $\mathscr{A}$ to visit all points at distance n. We define $f_+(n)$ and $f_b(n)$ respectively to be the number of poins on the $(n+1)$th reference diamond and the number of points beyond the $(n+1)$th reference diamond visited by $\mathscr{A}$ before the last visit to a point in reference diamond n.

We note that to visit m points, the algorithm must take at least $m-1$ steps. Since there are $2(n-1)^2+2(n-1)+1$ points within reference diamond $n-1$, we have

$$A(n-1)\geqslant 2(n-1)^2+2(n-1)+f_+(n-1)+f_b(n-1)$$

where the inequality may occur if some point is visited more than once. We consider the two cases where $f_+(n-1)\geqslant 2n-1$ and $f_+(n-1)\leqslant 2n-2$. If $f_+(n-1)\geqslant 2n-1$, then

$$A(n-1)\geqslant 2(n-1)^2+4(n-1)+1$$

and this meets our claim for distance $n-1$.

On the other hand, suppose $f_+(n-1)\leqslant 2n-2$. After the last visit to a point at distance $n-1$, there remain $4n-f_+(n-1)$ points unvisited in the nth reference diamond. Observing that it is impossible to visit any two points of a reference diamond without visiting at least one point between them, we see that visiting these remaining points requires at least $2(4n-f_+(n-1))-1$ steps. Thus,

$$\begin{aligned}A(n)&\geqslant A(n-1)+2(4n-f_+(n-1))-1\\&\geqslant 2(n-1)^2+2(n-1)+f_+(n-1)\\&\quad+f_b(n-1)+8n-2f_+(n-1)-1\\&\geqslant 2n^2+6n-1-f_+(n-1)\\&\geqslant 2n^2+4n+1. \quad \blacksquare\end{aligned}$$

An examination of this proof leads to several observations which lead to interesting consequences. The first observation is that we achieved the lower bound by a balancing act which was optimized by choosing $f_+(n)\approx 2n$. This implies that to get close to the lower bound, an algorithm should attempt to visit about half of the points at distance $n+1$ between visits to those at n, and between the remaining points at n we should visit points at distance $n-1$. Clearly, we should attempt to do the intermediate visits to the closer points first. These concepts lead to the algorithm illustrated in Fig. 1a.

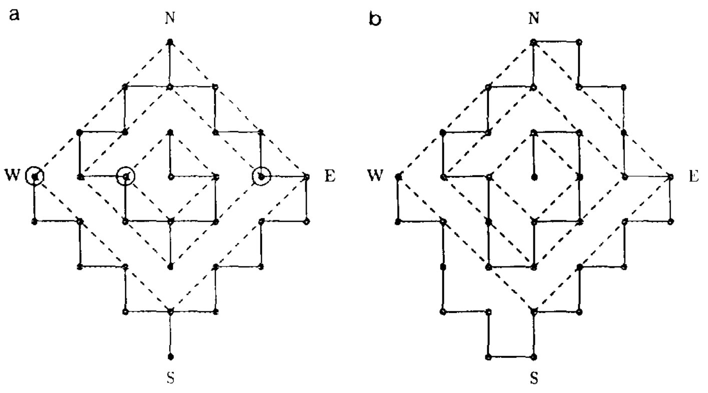

FIG. 1. Balanced algorithms for better worst case behavior: (a) balanced algorithm, NSESWSNWN; (b) flipped balanced algorithm, NESSWSWNN.

The sequence of directions is indicated for the visits to the points at distance 1. Note that alternate points on the line of points with horizontal coordinate zero are visited twice, including the origin. The idea is that between visits to points at distance one in the northern hemisphere, we visit points at distance zero, while in the southern hemisphere, we visit points at distance two. Similarly, between visits to points at distance three, in the northern hemisphere we visit points at distance two, while in the southern hemisphere we visit points at distance four.

In Fig. 1a, the circled points indicate the last points visited at the distance indicated by the dashed lines. Note that odd distances are completed to the west of the origin, while even distances are completed to the east, another indication of the balancing used in this algorithm. To continue the algorithm from any circled point, move out to the next level, and visit the points on the appropriate two sides of the reference diamond to the next circled point. To complete the nth distance set takes $4n+3$ steps, and thus by induction the algorithm requires $2n^2+5n+2$ steps before visiting the last point at distance n.

The second observation about the proof of Theorem 3.1 is that from the treatment of $f_b(n)$ it seems obvious that for the worst case analysis we may choose $f_b(n)=0$. In the previous algorithm, we only alternate between adjacent levels. Nevertheless, revisiting points seems wasteful. Extension of the pattern in Fig. 1b yields a $2n^2+5n+2$ step algorithm which does not repeat points. The sequence of steps for visiting the points at distance one is indicated in the caption. This algorithm can be derived from the previous algorithm by "flipping" the paths which repeat points so that the intermediate visit is to a point at the next distance out, and then avoiding this point on the next cycle. In the event that the goal is located at one of these "flipped out" points, this algorithm is clearly superior to the previous algorithm, but the worst case function of n is the same.

Third, we observe that the proof does not make use of induction. That is, for points at distance less than $n-1$ in the proof, only the trivial lower bound is used. So far our lower bound and our best algorithm differ by approximately n. The question then arises, can we increase the lower bound by using the lower bound for points at distance less than $n-1$ inductively?

Interestingly, if we are given *the parity of the point's distance* then we can improve the search by n steps! That is, we can get a pair of algorithms which make $2n^2+4n+n \bmod 2$ steps by exploring the appropriate pair of levels at each step. Either of these algorithms visits *all* points within any distance n. These algorithms illustrate that this inductive approach is not likely to improve the lower bound.

We illustrate the initial steps of these algorithms in Fig. 2. The algorithm for odd n illustrated in Fig. 2a optimally visits all points at distance 1 in 7 steps using the indicated sequence, terminating at the circled point at distance one. Note that it re-visits the origin three times. The last point visited at distance n for odd n is at the western tip of the reference diamond. After visiting this point, the algorithm proceeds WNE... visiting the points at distance $n+2$ and $n+1$ until reaching the western most tip of the $(n+2)$th diamond. Completion of the cycle which visits the points at distance n for odd n requires $8n$ steps. Letting $n=2k+1$, the number of steps required to visit all within distance n is $7+\sum_{i=1}^{k} 8(2i+1)$. This simplifies to $2n^2+4n+1$ when n is odd. For even n, the algorithm visits the last unvisited point at distance n just three steps before completing the

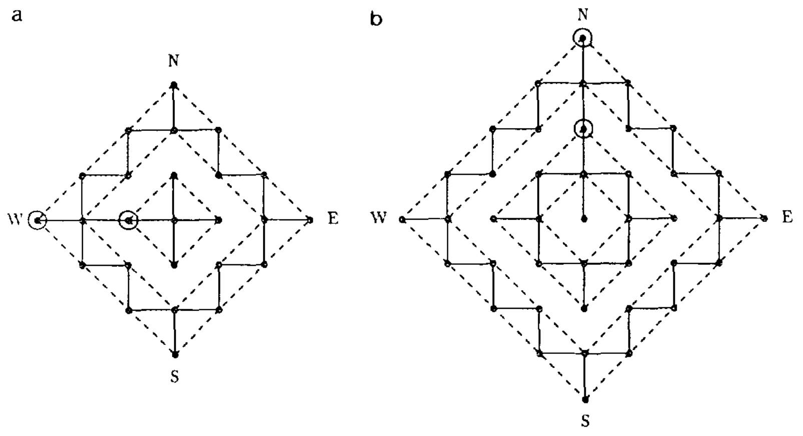

FIG. 2. Odd and even spiral algorithms: (a) the odd algorithm, NSEWSNW; (b) the even algorithm, NESEWSWSNWNWENEN.

cycle for reference diamond $n+1$. Using a similar argument, the algorithm takes $2n^2+8n+4$ steps to visit the last point at distance n.

The algorithm for even n is illustrated in Fig. 2b. The first cycle is given, which visits all points within distance 2 in 16 steps. The cycles end at the north point of the reference diamond for n even. The cycles start NESE... and follow around between the $(n-1)$th and nth reference diamonds, for even n. In general, this algorithm visits all points within n, for n, even, in $2n^2+4n$ steps. However, if the goal is the last point visited at an odd distance n, then this algorithm takes $2n^2+8n+3$ steps.

Flipped versions of the odd and even algorithms are illustrated in Fig. 3. In addition to avoiding repetitions, the worst case for the non-optimized parity is improved in each algorithm. For the odd algorithm, the last even point visited is at the south tip of the reference diamond, and for the even algorithm the last odd point is at the western tip. In each case, the worst case is reduced by $2n$ steps to $2n^2+6n+4$ for the odd algorithm and to $2n^2+6n+3$ for the even one.

For both algorithms, if the adversary is free to place the point at a distance of opposite parity to the parity that the algorithm was designed for, then the bound is worse than modified spiral search. The $2n^2+5n+2$ step algorithms can be seen as a trade off between the even and odd algorithms, making some of the wasted steps at even levels useful at the odd levels and vice versa.

Intuitively, the cost of turning the corners from one side of the reference diamond to the next combined with the need to make all the visits from the nth level to the closer intermediate points before any visits to the outer intermediate points, seems to prevent us from achieving the lower bound. We seem to require one extra step at each reference diamond to make these turns, and this apparently accounts for the difference of n between our upper and lower bounds. However, the gap remains open.

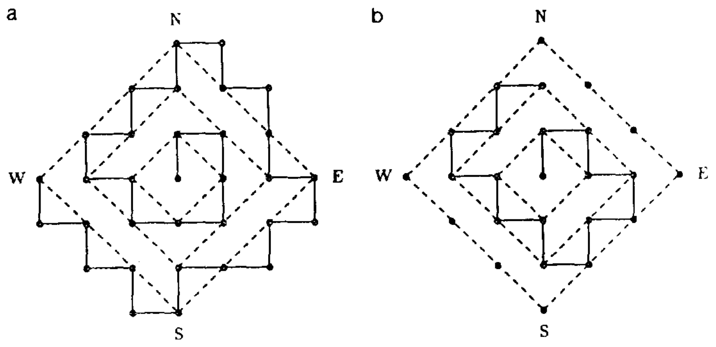

FIG. 3. Odd and even spiral algorithms: (a) the flipped odd algorithm, NESSWWN; (b) the flipped even algorithm, NESESWSWNWNWNENE.

4. Searching for a Line in the Plane

Suppose that the robot has to find a line in the plane. There are four natural scenarios: the robot either knows or does not know the distance to the line and it either does or does not have any information about the line's slope. We consider only the least information version of the problem; the other cases have been solved elsewhere. The first case, when the slope is known, is trivial. When the distance but not the slope is known the worst case search distance is $(1+\sqrt{3}+7\pi/6)\,n \approx 6.397n$ steps. An algorithm to optimize the average distance walked in this case is discussed in [11], and a variation where instead of searching for a line we are searching for a circle is discussed in [12]. For further results see [1, 3, 13, 17, 14, 10].

Suppose that the robot does not know the line's distance or slope. It seems clear that the optimal search path must be similar with respect to rotations and dilations (that is, the curve must have *spiral similarity*). The only known curve with these properties is the logarithmic spiral.

The robot executes a logarithmic spiral $r = k^{\theta}$, where $k = 1.250...$ (this value is a numerical approximation for the best logarithmic spiral). If the line is n steps away this algorithm takes approximately $13.81n + O(\ln n)$ steps. (Note that this is only an upper bound since we have assumed that the search path is a logarithmic spiral.)

If we restrict the line to be horizontal or vertical (that is, two orthogonal directions), we can prove a $12.74n$ lower bound assuming a logarithmic spiral, and a 13.02 upper bound, using similar techniques as before (see [1]). This lower bound also applies to the general case.

5. Further Problems

We conclude by stating, and giving some partial results for, three general planar search problems.

5.1. *Line of Restricted Slope a Bounded Distance Away*

Suppose that the robot knows the line's distance away and that the line's slope belongs to the finite set $\{\theta_1, \theta_2, ..., \theta_k\}$.

Given the distance n to the line, the possible set of slopes describes a polygon that circumscribes a circle of radius n. It is not difficult to formulate the optimal algorithm as a function minimization problem where the function has between k and $2k$ variables. If we restrict the polygon to be a regular j-gon then Table 5.1 summarizes the results for $j \leqslant 6$. These results are all optimal as may be proved by direct (although tedious) algebraic manipulation of the appropriate path minimization problems.

The minimum length for general j is unknown. In the above results the

TABLE 5.1

Results for Regular j-gons for Small j

j	Path length
3	$4n$
4	$3\sqrt{2}n \approx 4.24n$
5	$(4\sin^2(\pi/5) + (1 + 2\cos(\pi/10))/\cos(\pi/5))\,n \approx 4.97n$
6	$(4 + 2/\sqrt{3})\,n \approx 5.15n$

optimal path is a collection of diagonals of the regular j-gon. All of the above minimal paths stay within the boundary of the j-gon. However, it is possible to show that there exists a j for which the minimal algorithm must *leave* the boundary of the regular j-gon.

5.2. *Searching for a Point on the Boundary of a Region*

Suppose that we have a polygonal line (a non-self-intersecting continuous curve made up of straight line segments) bisecting the plane into two halfplanes. The robot's task is to find a point known to be somewhere on the polygonal line. The difference between this problem and the simpler one of searching for a point on a line is that the robot can at times shorten its path by moving off of the polygonal line.

For example, suppose the robot must search the boundary of a region bounded by two concurrent rays (assuming the robot is at the concurrent point initially) where the robot is not constrained to stay on the rays. There is a simple algorithm we call "bow-tie search"—walk along one ray for some number of steps, walk in the plane to the second ray, walk along the second ray for some number of steps, then return to the last point of departure on the first ray and repeat. As the angle between the rays is reduced to zero, the problem reduces to searching for a point on a ray. As the angle is increased to 180°, the problem reduces to searching for a point on a line. If the angle between the rays is 90°, the best bow-tie algorithm is given by $f(i) = k^i$, where $k = 1.849\ldots$. The worst case of this algorithm takes $7.422\ldots\, n$ steps.

In general, let $c = \cos(\theta)$, where θ is the smaller angle between the rays. The optimal k (for bow-tie search) is

$$2 - \frac{\left(\begin{aligned}4 + (c-1)^{1/3}\,&((c^2 + 14c + 17 - 4(c+3)\sqrt{2(c+1)})^{1/3}\\ &+ (c^2 + 14c + 17 + 4(c+3)\sqrt{2(c+1)})^{1/3})\end{aligned}\right)}{c+3}.$$

The worst case ratio is then

$$k + 1 + \frac{k\sqrt{k^2 - 2kc + 1}}{k - 1}.$$

We do not know if "bow-tie search" is optimal.

6. Open Problems and Conclusions

For each of the problems discussed in this paper there are three different criteria we could try to optimize:

- Minimize the maximum distance walked.
- Minimize the average distance walked.
- Maximize the probability that the object is found given that the robot can only walk x steps.

As we saw in the problem of searching for a point on a line, the best average case algorithm can be different from the best worst case algorithm. This answers a question of Oglivy [17] on whether the average case and worst case are always the same. What are the best search strategies for the problems considered in this paper with respect to the second and third criteria? What are the best search strategies using each of the above criteria assuming that we have k communicating (or non-communicating) robots instead of only one?[1] How can we modify our search strategies if there are obstacles in the plane? Finally, we pose the same search problems in higher dimensions.

To our knowledge search problems in which we have only partial information as to the location of the searched for object have not been previously studied as a class. We think that they are deserving of comprehensive study as simple optimality arguments (in particular variants of convexity and symmetry properties) are often applicable. Further, and more importantly, these problems are (very simple) models of searching in the real world. It is very often the case that we do not know many of the parameters that are usually taken for granted when designing search algorithms.

The results presented in this paper, summarized in Table 5.2, suggest that the relative information of knowing the general direction of a goal is much higher than knowing just the distance to the goal (in hindsight this result is intuitively obvious). Of course these are very simple problems and results from the more comprehensive problems may be more enlightening.

[1] Recently some results for this problem have been presented in Baeza-Yates and Schott, Parallel searching in the plane, *in* "XII Int. Conf. of the Chilean Computer Society, Santiago, Chile, Oct. 1992," pp. 269–279.

TABLE 5.2

The Advantage of Knowing Where Things Are

	Knowledge		
Problem	Direction	Distance	Nothing
Point on line	n	$3n$	$9n$
Point on m-rays	n	$(2m-1)n$	$(1+2m^m/(m-1)^{m-1})n$
Point in lattice	n	$9n-2$	$\leqslant 2n^2+5n+2$, $\geqslant 2n^2+4n-1$
Point in lattice with parity	n	$\leqslant 2n^2+4n+n \bmod 2$	$\leqslant 2n^2+4n+n \bmod 2$
Orthogonal line in plane	n	$4.24\cdots n$	$\leqslant 13.02n$, $\geqslant 12.74n$
Line in plane	n	$6.39\cdots n$	$\leqslant 13.81n$, $\geqslant 12.74n$

ACKNOWLEDGMENTS

Jon Bentley and Andrew Yao (private communication) independently studied the problem of searching for a point on a line (unpublished).

We thank Jon Bentley, Gaston Gonnet, Ron Graham, Bob Reckhow, and Chee Yap for many useful discussions on these problems. All of the results presented here, plus some other problems, were first reported in [1]. An extended abstract of this paper was presented at the Scandinavian Workshop on Algorithm Theory, Sweden, 1988 [2]. Recently, a number of papers in the same vein (with related results) have appeared [18, 9, 6]. We thank Prabhakar Raghavan who pointed out the relation of the results cited in this paper to the area of amortized analysis of on-line algorithms [5]; see [8] for further references.

We gratefully acknowledge the painstaking attention to detail and the useful organizational suggestions made by the anonymous referees.

RECEIVED December 27, 1988; FINAL MANUSCRIPT RECEIVED December 20, 1991

REFERENCES

1. BAEZA-YATES, R. A., CULBERSON, J. C., AND RAWLINS, G. J. E. (1987), Searching with Uncertainty," Research Report CS-87-68, Department of Computer Science, University of Waterloo.
2. BAEZA-YATES, R. A., CULBERSON, J. C., AND RAWLINS, G. J. E. (1988), Searching with uncertainty, *in* "Proceedings SWAT 88, First Scandinavian Workshop on Algorithm Theory" (R. Karlsson and A. Lingas, Eds.), pp. 176–189, Lecture Notes in Computer Science, Vol. 318, Halmstad, Sweden.
3. BELLMAN, R. (1956), A minimization problem, *Bull. Amer. Math. Soc.* **62**, 270.
4. BENTLEY, J. L., AND YAO, A. C.-C. (1976), An almost optimal algorithm for unbounded searching, *Inform. Process. Lett.* **5**, 82–87.
5. BORODIN, A., LINIAL, N., AND SAKS, M. (1987), An optimal online algorithm for metrical task systems, *in* "Proceedings of the 19th Annual ACM Symposium on the Theory of Computing," pp. 373–382.

6. CARLSSON, S., AND KARLSSON, R. (1989), Sorting and selection by a robot, *in* "Proceedings of the IX International Conference of the Chilean Computer Science Society," Santiago.
7. CHANG, S.-K. (1974), A triangular scanning technique for locating boundary curves, *Computer. Graphics Image Process.* **3**, 313–317.
8. COPPERSMITH, D., DOYLE, P. RAGHAVAN, P., AND SNIR, M. (1990), "Random Walks on Weighted Graphs and Applications to On-Line Algorithms," IBM Research Report, IBM T. J. Watson Research Center, Yorktown Heights.
9. EADES, P., LIN, X., AND WORMALD, N. C. (1989), "Performance Guarantees for Motion Planning with Temporal Uncertainty," abstract presented at the First Canadian Conference on Computational Geometry, Montreal.
10. FABER, V., AND MYCIELSKI, J. (1986), The shortest curve that meets all the lines that meet a convex body, *Amer. Math. Monthly* **93**, 796–801.
11. GLUSS, B. (1961), An alternative solution to the "lost at sea" problem, *Naval Res. Logist. Quart.* **8**, 117–128.
12. GLUSS, B. (1961), The minimax path in a search for a circle in the plane, *Naval Res. Logist. Quart.* **8**, 357–360.
13. ISBELL, J. R. (1957), An optimal search pattern, *Naval Res. Logist. Quart.* **4**, 357–359.
14. JORIS, H. (1980), Le chasseur perdu dans la forêt, *Elem. Math.* **35**, 1–14. [In French]
15. KARP, R. M., SAKS, M. AND WIDGERSON, A. (1986), On a search problem related to branch-and-bound procedures, *in* "27th Annual Symposium on Foundations of Computer Science," pp. 19–28.
16. MELZAK, Z. A. (1973), "Companion to Concrete Mathematics: Mathematical Techniques and Various Applications," p. 153, Wiley, New York.
17. OGLIVY, C. S. (1962), "Tomorrow's Math: Unsolved Problems for the Amateur," pp. 23–25, Oxford Univ. Press, London/New York.
18. PAPADIMITRIOU, C., AND YANNAKAKIS, M. (1989), Searching without a map, *in* "ICALP'89," pp. 610–620, Lecture Notes in Computer Science, Vol. 372, Stresa, Italy.
19. RIVEST, R. L., MEYER, A. R., KLEITMAN, D. J., AND WINKLMANN, K. (1980), Coping with errors in binary search procedures, *J. Comput. System Sci.* **20**, 396–404.