Elmar Langetepe

Online Motion Planning

MA INF 1314
For the online variant we can restrict the algorithm to explore the polygon up to depth \(d\). As before we ignore all cuts where the reflex vertex has distance > \(d\). The \(\sqrt{2}\)-approximation against the SWR up to depth \(d\) remains valid. Therefore for the application of Theorem 3.24 we conclude \(\beta = 1\) and \(C_\beta = 1\) and attain a \(8\sqrt{2}\)-approximation of the search ratio and the optimal search path.

**Corollary 4.8** The optimal search path in a simple, rectilinear can be approximated within a factor of 8 in the offline case and within a factor of \(8\sqrt{2}\) in the online case.

### 4.3 General simple polygons

As in the previous section we first concentrate on the offline computation of a SWR in a simple polygon. As already shown in Figure 4.4 the sub-polygons \(P_i\) of the essential cuts will be visited in the order along the boundary. More generally we extract the following general computation task, which finally ends in Theorem 4.1. A touring-a-sequence-of-polygons gives a generalization of the SWR computation.

**Definition 4.9**

(i) In the simple **Touring Polygon Problem (TPP)** version a sequence of simple, convex and disjoint polygons \(P_1,P_2,\ldots,P_k\) with \(n\) edges in total is given. Furthermore, a start point \(s\) and a target point \(t\) is fixed. We are searching for the shortest path that starts in \(s\), visits the polygons \(P_i\) in the order given by the index \(i\) and ends in \(t\).

(ii) In the general version of the TPP, the path between two successive polygons \(P_i\) und \(P_{i+1}\) \((i = 0,\ldots,k); P_0 := s; P_{k+1} := t\) can be forced to run in a so-called **fence-polygon** \(F_i\). The fence \(F_i\) is a simple polygon that contain \(P_i\) and \(P_{i+1}\). Additionally, the polygons might overlap, i.e., the intersection of \(P_i\) and \(P_j\) need not be empty. In the presence of a fence for polygons \(P_i\) and \(P_{i+1}\), it is allowed that only the boundary parts of \(P_i\) and \(P_{i+1}\) that do not belong to the boundary of the fence form a convex chain. We call this part the **facade** of \(P_i\) or \(P_{i+1}\), respectively. More precisely facade\((P_i) := \partial P_i \setminus \partial F_{i-1}\).

![Figure 4.9: The worst-case detour in a triangle is \(\sqrt{2}\).](image)

The interpretation of the TPP is as follows: It can happen that for \(j < i\) a polygon Polygon \(P_i\) has been visited by chance before polygon \(P_j\) is visited, the first visit will be ignored, the polygon \(P_i\) has to be visited again. More precisely the visit of \(P_i\) is valid, if the polygons \(P_1,\ldots,P_{i-1}\) have been visited

![Figure 4.10: An example for the simple version of the Touring Polygon Problem.](image)
in this order before, Figure 4.10 shows an example for the simple TPP configuration and Figure 4.11 examplifies the general case. The dashed part of the boundary of $P_4$ is the facade of $P_4$. Note, that $P_5$ was visited before $P_4$ is entered, we “register” the visit of $P_3$ after $P_4$ was visited.

**Theorem 4.10** (Dror, Efrat, Lubiw, Mitchell, 2003)

For the general TPP with $k$ polygons, $k + 1$ fences and $n$ edges in total for all polygons and fences there is an algorithm that computes a query structure for the TPP in $O(k^2 n \log n)$ time. The query structure has a complexity of $O(k n)$. For a fixed start point $s$ and for any query target point $t$ the shortest TPP path can be computed in $O(k \log n)$ time where $m$ denotes the number of segments of the shortest TPP path.

[DELM03]

Now let us come back to our initial SWR problem. We now sketch the proof of Theorem 4.1.

**Proof.** Let us assume that $P$ and a start point $s$ on the boundary is given. We construct a TPP input $(P_1, \ldots, P_k, F_1, \ldots, F_k, s, t)$ as follows. Let $c_i$ be the $i$-th essential cut of $P$ along the boundary of $P$ and $P_i$ the corresponding sub-polygon. We set $P_i := P_{c_i}$. Any fence will be the polygon $P$ itself, which is $F_i := P$. The facades of any $F_i$ is given by the cut $c_i$. Finally, we set $t := s$ for returning to the start. The SWR is the shortest path that starts at $s$ visits the polygons $P_i$ in the given order inside the polygon $P$ and ends at $s$. The cuts $c_i$ build convex facades for the possibly non-convex polygons $P_i$. This gives exactly the task in the corresponding TPP. The complexity of the facades is in $O(1)$ and the complexity of the fence is in $O(n)$. We can have $\Omega(n) = k$ many polygons $P_i$. The running time is in $O(n^3 \log n)$.
Finally, we consider the online exploration of general simple polygons. The greedy approach for rectilinear polygons explored the cuts of the reflex vertices in the order (of the vertices!) along the boundary. Let us assume that in a general polygon we get some more information and all cuts are given. If we explore the cuts in the order of the corresponding reflex vertices and construct the shortest (optimal) path for this visiting order, the corresponding path can be arbitrarily large in comparison to the SWR of the polygon. Figure 4.12 shows an example where the greedy approach with additional information does not succeed w.r.t. a constant competitive approximation.

Figure 4.12 also shows that it makes sense to bundle the reflex vertices and subdivide them into cuts that will be detected if the agent moves to the left and cuts that will be detected, if the agent moves to the right. This is what the corresponding SWR does in principle. We would like to formalize this idea by categorizing the reflex vertices correspondingly.

**Definition 4.11** Let $P$ be a simple polygon and $s$ be a start point at the boundary of $P$. The Shortest Path Tree, $\text{SPT}(P,s)$, contains the shortest paths inside $P$ that runs from $s$ to the all vertices of $P$. The SPT is the smallest set of segments that contains all the paths. W.r.t. the SPT a reflex vertex $v$ of $P$ is denoted as a left vertex Ecke, if the $\text{SPT}(P,s)$ makes a counter clockwise turn at $v$ and right vertex, if the $\text{SPT}(P,s)$ makes a clockwise turn at $v$; see Figure 4.13. The interpretation is that w.r.t. the path from $s$, $v$ lies to the left or $v$ lies to the right of the preceding vertex.

Different from the rectilinear case we will not approach the reflex vertices orthogonally, we make use of circular arcs. Consider Figure 4.14. The agent is located at $s$ and detects the reflex vertex $v$. The angle $\alpha$ for the cut is unknown because vertex $v$ blocks the corresponding edge. Assume the agent moves directly toward $v$. An adversary will choose a very large angle $\alpha$ — as in Figure 4.14(i) — such that an
arbitrary short path orthogonal to the cut is sufficient. In this sense the direct path to the vertex is not competitive.

Therefore we explore the vertex (or its cut) by a half-circle starting at \( s \) around the midpoint of \( sv \) and radius \(|sv|/2\). For approaching the cut this gives a competitive ratio of at most \( \frac{\pi}{2} \). For the above looking-around-a-corner problem the exploration by the half-circle is not the overall best strategy, as will be shown in the next section. In comparison to the optimal corner strategy the half-circle strategy can be easily analysed and has nice properties against the shortest path.

The half-circle exploration is not the overall best strategy for looking around a corner. A refined analysis shows the following result:

**Theorem 4.12** (Icking, Klein, Ma, 1993)
The problem of looking around a corner can be solved within an optimal competitive ratio of \( \approx 1.212 \). [IKM94]

We first formally show the competitive ratio of the half-circle strategy for detecting the cut and also give a simple lower bound.

**Theorem 4.13** The unknown cut of a reflex vertex in a simple polygon can be detected by a half-circle strategy within a competitive ratio of \( \frac{\pi}{2} \) against the shortest path to the cut. It can be shown that there is no online strategy that explores any corner (visit the cut) within a ratio less than \( \frac{2}{\sqrt{3}} \).

**Proof.** We consider the normalized version of the problem from Figure 4.15. For the offline optimal solution either the vertex \( O \) will be visited directly or the cut will be approached orthogonally, the cut is not know which is indicated by the unknown angle \( \varphi \). For \( \varphi \in [0, \pi/2] \) the orthogonal distance \( \sin \varphi \) gives the optimal solution. For \( \varphi \in [\pi/2, \pi] \) the shortest path to \( O \) of length 1 is optimal.

We compare the optimal solutions to the half-circle strategy for any \( \varphi \). Until the half-circle finally hits \( O \) at angle \( \varphi = \pi/2 \) (and therefore for all \( \varphi \in [\pi/2, \pi] \)), the half-circle strategy has arc length \( \varphi \) for any \( \varphi \in [0, \pi/2] \). For all possible cuts with angle \( \varphi \in [\pi/2, \pi] \) we attain a ratio \( H(\varphi) = \frac{\varphi}{\sin \varphi} \). The first derivatives gives \( H'(\varphi) = \frac{\sin \varphi \cos \varphi - \varphi \cos \varphi}{\sin^2 \varphi} \) and by simple analysis we have \( H'(\varphi) > 0 \) for \( \varphi \in (0, \pi/2) \). Therefore in both cases the ratio \( \frac{\pi}{2} \) is the worst-case.

![Figure 4.15: The optimal path to the unknown cut either is given by the direct path to \( O \) of length 1 for \( \varphi \in [\pi/2, \pi] \) or is given by an orthogonal path of length \( \sin \varphi \) for \( \varphi \in [0, \pi/2] \). For the half-circle strategy the worst-case ratio is attained at \( \varphi = \pi/2 \) with a ratio of \( \pi/2 \).](image)
For the lower bound we consider Figure 4.16. We provide a bit more information for the online strategy. Either $\phi$ is exactly $\frac{\pi}{6}$ or $\phi = \frac{\pi}{2}$. In the first case the optimal path has length $\sin \frac{\pi}{6}$ and in the second case the optimal path has length 1.

![Figure 4.16: The lower bound construction gives a ratio of $\frac{2\sqrt{3}}{3}$.](image)

Any strategy will visit the $\pi/6$-cut somewhere (may be also at the end at point O). Therefore we consider the isosceles triangle with ground length $OW$ and two angles of size $\pi/6$. This means the one segment if the triangle runs in parallel with the $\pi/6$. Consider the vertex $X$ of the triangle on the $\pi/6$-cut. Either an online strategy visits the $\pi/6$-cut to the left or to the right of $X$. Both cases might include that exactly $X$ is visited.

In the first case (visit to the left of $X$), the adversary present the $\pi/2$-cut as the true cut and the agent now moves toward O. The optimal path has length 1, whereas the strategy runs at least $2 \cdot \frac{1}{2 \cos \frac{\pi}{6}} = 2 \cdot \frac{\sqrt{3}}{3} = \frac{2\sqrt{3}}{3}$. In the latter case the (visit to the right of $X$), the adversary present the $\pi/6$-cut as the true cut, the ratio is at least $\frac{1}{\sin \frac{\pi}{6}} = \frac{2}{\sqrt{3}}$. In both situations the same worst-case ratio is attained.

We will now sketch the ideas for the competitive only exploration of a general polygon by a recursive subdivision of the reflex vertices in groups of left and right vertices and by a consequent successive exploration of the groups by half-circles.

By Algorithm 4.3 we explore a single right vertex. The strategy manages two lists of vertices. The TargetList contains right vertices that have been detected (but not explored) ordered in ccw-order along the boundary. Right vertices, that will be detected by ExploreRightVertex and that do not lie behind left vertices of the SPT, will be inserted into TargetList during the execution of ExploreRightVertex. It might happen that the goal vertex Target changes during the execution. In this sense ExploreRightVertex does not only explore a single right vertex, the target changes. The exploration is restricted to a set of right vertices that subsequently lie along the boundary such that no left reflex vertex occurs in between. The goal is to explore all vertices of the sequence. We consider the exploration as shown in Figure 4.17 and exemplify the usage of Algorithm 4.3.

The agent starts in $s$. We initialize BasePoint by $s$ and TargetList contains only $r_1$. The target $r_1$ is visible. Back is also $s$. We follow the half-circle arc$(s, r_1)$ until the next right vertex $r_2$ is detected at $e_1$ (the first event). Since $r_2$ does not lie behind a left vertex and lies in ccw-order behind $r_1$ we insert $r_2$ into the target list TargetList. In ccw-order $r_2$ lies in front of $r_1$ and $r_2$ is the first element of the target list. Therefore there is an update of Target and the agent now moves along the half-circle arc$(s, r_2)$. At
Algorithm 4.3 Exploration of a right vertex.

ExploreRightVertex( TargetList, ToDoList ):

2. Target := first vertex of TargetList.
3. while Target is no longer visible do
   4. Move along shortest path from BasePoint toward Target.
4. end while
5. Back := last polygon vertex reached along the shortest path from BasePoint to the current position.
6. while Target is not fully explored do
   7. Move along halfcircle arc(Back, Target) in cw-order.
   8. Update TargetList, ToDoList, Target, Back during the task.
7. end while

// Special situations during the half-circle move:
if the boundary of P blocks the move then
  Follow the boundary until the half-circle can be continued.
end if
if Target will get out of sight by a vertex then
  Move toward Target to the vertex, that blocks the sight.
end if

the second event e2 the visibility to the current target gets blocked by \( \ell_1 \), which is the second special situation. The agent moves toward the target \( r_2 \) to \( \ell_1 \). At \( \ell_1 \) we update \( Back := \ell_1 \), since we reached a polygon vertex. Now we move along the arc arc(\( \ell_1, r_2 \)). Close behind \( \ell_1 \) the BasePoint \( s \) is no longer visible and still \( Back := \ell_1 \) remains true. At event \( e_3 \) the \( s \) gets visible again, we are no longer at a vertex and we set \( Back := s \). Note that the BasePoint \( s \) will be \( s \) all the time. At event \( e_4 \) the current Back point \( s \) vanishes again, we set \( Back := r_1 \) and run along the half-circle arc(\( r_1, r_2 \)). At event \( e_5 \) the next right reflex vertex \( r_3 \) in ccw-order is detected, inserted into TargetList and gets the new Target. Therefore we move along the half-circle arc(\( \ell_1, r_3 \)) until at \( e_6 \) the \( r_1 \) (Back) gets out of sight and we set \( Back := r_2 \) and continue with arc(\( r_2, r_3 \)). This movement is blocked between \( e_7 \) and \( e_8 \) from the boundary where the first special situation is used and the agent follows the boundary until picking up arc(\( r_2, r_3 \)) again. Finally, \( r_3 \) is fully explored. The vertex \( r_3 \) defines an essential cut that dominates the cuts of \( r_1 \) and \( r_2 \).

Since the current target Target is explored, the procedure Algorithm 4.3 ends. It might be the case that TargetList still contains non-explored reflex vertices. In general the procedure Algorithm 4.3 is part the procedure ExploreRightGroup that explores such a group of reflex vertices successively by Algorithm 4.3 with corresponding BasePoints.

Algorithm 4.4 Exploration of a group of right vertices.

ExploreRightGroup( TargetList, ToDoList ):

1. StagePoint := current position of the agent.
2. ToDoList := \( \emptyset \)
3. while TargetList is non-empty do
   4. ExploreRightVertex( TargetList,ToDoList ).
   5. For the current cut, move along the point on the cut that has the shortest distance (in \( P \)) to the StagePoint, update TargetList and ToDoList.
5. end while
6. Move along the shortest path (in \( P \)) back to StagePoint.

We exemplify ExploreRightGroup (Algorithm 4.4) and its interplay with ExploreRightVertex by Figure 4.18. Beginning at \( s \) as the current stage point and with the first single target \( r_1 \) in the target list
we start $\textit{ExploreRightVertex}$. Analogously, to the above description at event $e_1$ the vertex $r_2$ is detected and $\text{arc}(s, r_2)$ is started. $r_2$ is the new $\text{Target}$. Since the vertices $r_6$ and $r_3$ which are detected during the movement of $\text{arc}(s, r_2)$ up to $e_2$ do not lie behind (in ccw-order) $r_2$ they will not become new targets in the procedure $\textit{ExploreRightVertex}$. In the current target list $\text{TargetList}$ $r_6$ and $r_3$ lie behind $r_2$. In $e_2$ the current target vertex $r_2$ is fully explored and $\textit{ExploreRightVertex}$ ends here. Fortunately, w.r.t. the current $\text{Back}$ point $s$ the segment $se_2$ is orthogonal to the cut of $r_2$, the agent is located at the point on the cut with the smallest distance back to the $\text{Back}$ point.

Now in $\textit{ExploreRightGroup}$ the procedure $\textit{ExploreRightVertex}$ is called up again with $r_3, r_6$ in the target list. This exploration ends at $e_3$. The vertices $r_4$ and $r_5$ are detected (and inserted in the target list) during the walk along $\text{arc}(s, r_3)$. Note that $r_6$ is deleted during an update of the list. $r_6$ was exposed. The vertices $r_4$ and $r_5$ do not lie behind $r_3$ and therefore first $r_2$ is fully explored and $\textit{ExploreRightVertex}$ ends again.

In between the $\text{Back}$ point has changed to $r_1$. Now w.r.t. the cut of $r_3$ at $e_3$ the agent is not located at a point on the cut of $r_3$ that has the shortest distance to the stage point $s$ (and also to the current $\text{Back}$ point). Therefore we move to such a point $e_4$ along the cut of $r_3$. This movement is part of the $\textit{ExploreRightGroup}$ procedure. Now $\textit{ExploreRightVertex}$ is applied with target $r_4$ and current back point...
so that the arc \( r_1 \) is used until this procedure end at \( e_5 \).

The current back point has changed to \( r_6 \) and the ExploreRightGroup forces the agent to slip along the cut of \( r_4 \) to move to the point closest to \( r_6 \) and \( s \). The TargetList is updated in between and also \( r_5 \) is deleted out of TargetList. Now the TargetList is empty and in our case we return to the StagePoint which is \( s \) in this case.

The procedure ExploreRightGroup is used in the frame procedure Algorithm 4.5. This procedure builds up groups of left and right vertices which are explored in an alternating way. The usage of ExploreRightGroup goes into the depth in the sense that there is a list of stage points (StagePoint) (back points on the shortest path back to \( s \)) stored in the ToDoList that will be used as starting points for the procedure ExploreRightGroup. Analogous procedures for the exploration of left vertices and groups of left reflex vertices will be used.

The main procedure starts with the exploration of a right group from the start and returns to the start. After that all known left vertices are ordered along the boundary and the same group procedure is called for the left vertices from the start. Then the recursion starts by moving to the stage points and recall the procedures from there.

**Algorithm 4.5** Exploration of simple polygons.

**ExploreRightGroupRec** (TargetList):

ExploreRightGroup(TargetList, ToDoList).

for all Vertex \( v \) in ToDoList do

Move along the shortest path to \( v \).

NewTargetList := all detected left vertices,
               which are successor of \( v \) in the SPT.

ExploreLeftGroupRec(NewTargetList).
end for

**ExplorePolygon** (\( P, s \)):

TargetList := right vertices visible from \( s \), sorted in cw-order along the boundary of \( P \).

ExploreRightGroup(TargetList, ToDoList)

TargetList := detected left vertices, lying behind (in the SPT) the vertices of ToDoList.

Additionally add all from \( s \) visible left vertices to TargetList.

Sort TargetList in ccw-order.

ExploreLeftGroupRec(TargetList).

**Theorem 4.14** (Hoffmann, Icking, Klein, Kriegel, 1998)

The strategy PolyExplore explores an unknown simple polygon within a competitive ratio of 26.5 against the SWR.

The ratio of 26.5 might appear to be huge, in fact it is an improvement of the ratios 133 (Hoffmann et al. [HIKK97]) or 2016 (Deng et al. [DKP91]) previously known. Indeed, the ratio is merely a result of the analysis. The best known lower bound for the strategy was given by an example where the ratio is roughly 5. The conjecture is that the ratio of the strategy is indeed close to 5, whereas a full proof can only be given for 26.5.

The online and the offline strategies given above can be easily restricted to a depth \( d \). As mentioned before it suffices to ignore all reflex vertices with distance \( > d \). This means that the approximation factors of 26.5 and 1 remain valid for the depth-restricted case. Note that the SWR for depth \( d \) might leave \( P(d) \); see Figure 4.19.

For the online case we can make use of \( \beta = 1 \) and \( C_\beta = 26.5 \) for the exploration of \( P(d) \), in the offline case we have \( \tilde{\beta} = 1 \) and \( \tilde{C}_\beta = 1 \). Application of Theorem 3.24 gives the following result:
Corollary 4.15  The optimal search path and the optimal search ratio for general simple polygons can be approximated offline within a ratio of 8 and online within a ratio of 212.

Figure 4.19: In this case SWR(d) leave the part P(d). PolyExplore keeps inside P(d).

4.4 Polygons with holes

In the previous section competitive strategies for the exploration of simple polygons were presented. We would like to show that in a scene with polygonal obstacles such results cannot be obtained. We consider non-simple polygons which means that the polygon has holes (or obstacles) inside. These holes are non intersecting and they are given as simple polygons itself.

The task of exploring a polygon with holes is much more complicated. At the first place the computation of the SWR is NP-hard. There is a simple reduction of the TSP problem by placing small obstacles around the corresponding point set. Furthermore, for simple polygons it can be shown that it suffices to explore the boundary. More precisely, if the boundary of a simple polygon P was seen along an exploration path, also any point inside P has been seen by the path. This is not true for polygons with holes as depicted in Figure 4.20. The path $\pi$ sees the boundary of all obstacles and the outer boundary, but there is still a portion of the polygons that is not explored.

Figure 4.20: A polygon with holes. The path detects the full boundary but not all points inside P have been seen.

We can show that there is no strategy that explores any polygon with holes within a constant competitive ratio against the shortest exploration path.

Theorem 4.16  (Albers, Kursawe, Schuierer, 1999)

Let $A$ be an arbitrary online strategy for an agent with a vision system for the exploration of a polygon $P$ with holes. Let $n$ denote the overall number of vertices of $P$. we have

$$|\pi_A| \geq \Omega(\sqrt{n}) \cdot |\pi_{opt}|.$$
Chapter 4  Exploration in polygons

Proof. We recursively construct a polygonal scene as shown in Figure 4.21. The starting scene consists of \( k + 1 \) thin rectangles of length \( W = 2k \) and arbitrarily small height, called spikes, and \( k \) rectangles of width 1 and height 1, the so-called bases. The construction has height roughly \( H_1 = k \). The agent starts at the lower left corner. Between a spike and a base there is an arbitrary thin corridor, so that the agent can move inside and have a look behind the base. Behind one of the bases the situation appears recursively, again with \( k \) spikes of length \( W_i = 2k - i \) and \( k \) bases of width 1. The overall height is \( H_i := \frac{1}{(2k)^{i-1}} \). The agent does not know whether the next sub-problem has the bases on the left or on the right side.

The construction will be repeated \( k \) times with values \( H_{i+1} = \frac{H_i}{2} \) and \( W_{i+1} = W_i - 1 \) for \( i = 2, \ldots, k - 1 \), starting with \( H_1 = k - 1 \) and \( W_1 = 2k \). This means that we have \( k \) sub-problems, each nested behind the base of a previous one (up to the starting problem). Altogether, we have \( k \times (2k + 1) \) rectangles and \( 4k \times (2k + 1) = n \) edges, with \( k \in \Omega(\sqrt{n}) \).

The strategy \( A \) has to see all points. In the first stage for finding the second block, the agent can either look behind the \( k \) bases from the left by moving distance \( 2k - 1 \) or moves to the right (distance \( 2k \)) and then upwards. For both cases the next block will be presented at the last visit. In the first case the next base rectangles are located to the left, in the latter case the next base rectangles are located to the right. So the same situation occurs again. This means that the agent has to move at least \( k \) times distance \( k \) which gives \( \Omega(k^2) \) in total. This means \( |\pi_A| \in \Omega(k^2) \).

The optimal offline strategy directly moves to the base where the next recursive sub-problem is nested. Then the sub-problem is explored optimally with path length \( 2H_i \). Finally, the agent has to move to the right upper corner and moves back along the left side to look behind all bases; see Figure 4.21.

We have

\[
|\pi_{\text{opt}}| = 2W_1 + 2 \sum_{i=1}^{k} H_i \\
= 2W_1 + 2H_1 \sum_{i=1}^{k} \frac{1}{(2k)^{i-1}} \\
= 4k + 2k \left( \frac{1}{(2k)^k} - 1 \right) \\
= 4k + 2k \left( \frac{2k \left( 1 - \left( \frac{1}{2k} \right)^k \right)}{2k - 1} \right) \\
\leq 8k.
\]

This gives a ratio of \( \Omega(k) = \Omega(\sqrt{n}) \) which gives the bound \( \Omega(\sqrt{n}) \). \( \square \)

Finally, by a simple trick we show that also the optimal search path cannot be approximated within a constant ration. The optimal search path for the above situation might decide to detect a point that has distance 1 from the start after \( \Omega(k) \) steps, therefore the search ratio might be \( k \).

To avoid this situation we shift the start \( k \) steps away from the block construction as shown in Figure 4.22. Now any non-visible point has distance at least \( k \). An optimal exploration path has length at
most 10k and gives a constant approximation of the search ratio (which is a constant). As shown above any online strategy will detect the last point distance at most 4k away after at least $\Omega(k^2)$ steps. Thus the search ratio is in $\Omega(k)$.

**Corollary 4.17**  For polygons with holes there is no strategy that approximates the optimal search path and the search ratio within by a constant factor.

![Diagram](image)

Figure 4.22: Shifting the start point away means that any invisible point has distance $\Theta(k)$, this gives a constant search ratio for the best offline exploration path.
Bibliography


# Index

<table>
<thead>
<tr>
<th>•</th>
<th>see disjoint union</th>
</tr>
</thead>
<tbody>
<tr>
<td>∪</td>
<td>disjoint union</td>
</tr>
<tr>
<td>1-Layer</td>
<td>14</td>
</tr>
<tr>
<td>1-Offset</td>
<td>14</td>
</tr>
<tr>
<td>2-Layer</td>
<td>14</td>
</tr>
<tr>
<td>2-Offset</td>
<td>14</td>
</tr>
<tr>
<td>lower bound</td>
<td>5</td>
</tr>
</tbody>
</table>

## A

| Abelson | 45 |
| accumulator strategy | 31 |
| adjacent | 8 |
| Albers | 30, 111 |
| Alpern | 63 |
| angular counter | 43 |
| approximation | 30 |
| Arkin | 30 |

## B

| Backtrace | 19 |
| backward analysis | 83 |
| Betke | 30 |
| Bug-Algorithms | 52 |

## C

| CAB | 88 |
| caves | 80 |
| cell | 8 |
| C_free-condition | 46 |
| C_half-condition | 47 |
| Chin | 97, 98, 100 |
| columns | 29 |
| competitive | 35, 37 |
| configuration space | 46 |
| constrained | 31 |
| Constraint graph-exploration | 31 |
| cow-path | 62 |
| current angular bisector | 88 |
| cut | 98 |

## D

| Deng | 101, 110 |
| DFS | 8, 11 |
| diagonally adjacent | 8, 27 |
| Dijkstra | 19 |
| diSessa | 45 |
| disjoint union | 15 |
| doubling | 92 |
| doubling heuristic | 62 |
| Dror | 97, 104 |
| Dudek | 40 |
| Duncan | 35, 37 |

## E

| Efrat | 97, 104 |
| error bound | 45 |
| Euclidean metric | 101 |

## F

| facade | 103 |
| Fekete | 30 |
| fence-polygon | 103 |
| Fleischer | 93 |
| functionals | 62 |
| funnel (polygon) | 82 |
| funnel polygons | 82 |
| funnel situation | 82 |

## G

| Gabriely | 27, 29 |
| Gal | 63 |
| Geometric search | 90 |
| goal set | 90 |
| Greedy | 101 |
| grid-environment | 8 |
| gridpolygon | 8, 30 |

## H

| Hit-Point | 52 |
| Hit-Points | 46 |
| Hoffmann | 110 |
Spanning-Tree-Covering .......................... 23
split-cell .............................................. 14
Stepanov ................................. 52, 53, 55, 58
street .................................................. 79
street polygon ................................. 79
sub-cells ............................................. 23
Sutherland ................................. 3
Szwarcfiter ................................. 8

T

Tarjan ................................. 5
tether strategy ................................. 31
tool ............................................. 23
touch sensor ................................. 8
Touring Polygon Problem .......... 103
triangulation ................................. 100
Trippen ................................. 93

U

unimodal ........................................ 63

V

vertex search ................................. 90
Vidyasagar ................................. 56
visibility polygon .......................... 61, 61
visible ........................................ 61

W

Wave propagation .......................... 19
weakly visible ................................. 79
Wilkes ................................. 40
work space ................................. 46

Y

y–monotone ................................. 97
Yannakakis ................................. 91, 93