Mutually visible agents in a discrete environment

Joel Fenwick **V. Estivill-Castro**

Institute for Integrated and Intelligent Systems
Griffith University, Australia
Email: j.fenwick@student.griffith.edu.au v.estivill-castro@griffith.edu.au

Abstract

As computer controlled entities are set to move and explore more complex environments they need to be able to perform navigation tasks, like finding minimal cost routes. Much work has been done on this problem with a single entity in a continuous environment. However these entities may be in teams and their actions may be constrained by the need to consider the actions of other entities. We consider the case of two entities wishing to reach their destinations while travelling the minimum distance and remaining in sight of each other. A version of this problem in continuous space has been addressed previously; however, the problem of minimum length paths does not only exist in continuous space. Robots may have restricted orientations in movement. Also, domains such as circuit design and computer games require discrete movements and restricted orientation. The restricted orientation and the fact that (even in genus zero environments) minimal length paths may not be unique present some challenges. This paper investigates the problem of two entities in a discrete setting, moving to preserve visibility and provides algorithms for computing schedules that minimise the total distance travelled.

1 Introduction

There has been significant interest in problems relating to visibility and/or shortest paths in polygonal environments due to the large number of applications in areas like computer graphics, robot navigation, geographical information systems and circuit layout. From the computational geometry perspective, these are presented as path planing problems (LaValle 2006) or guarding a region problems (O'Rourke 1987), and together with graph algorithms, the literature has seen many advances (Sack & Urrutia 2000). However, most of these advances concentrate on finding one shortest path within continuous (vectorial) representations, perhaps with some constraints, and possibly with restricted discrete orientations. Little work has been focused on cases with two or more entities where movement is to be determined under some visibility constraint. Less work falls into the intersection of finding several paths when the environment is discrete or enforces some sort of restricted set of orientations for movement (Fink & Wood 2003, Flocchini, Prencipe, Santoro & Widmayer 2005, Schuierer & Wood 1999, Widmayer, Wu, Schlag & Wong 1986).

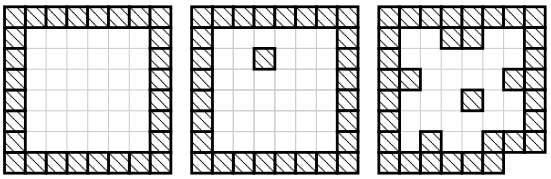

Figure 1: Inputs with the same sized bounding box.

We consider the case of two entities in a polygonal environment wishing to reach their destinations while travelling the minimum distance and remaining in sight of each other. A version of this problem in continuous space has been addressed (Fenwick & Estivill-Castro 2005); however, the problem of minimum length paths does not only exist in continuous space. Robots may have restricted orientations in movement. Also, domains such as circuit design and computer games require discrete movements and restricted orientation. Other work (Ickling & Klein 1992) in the continuous case has discussed two guards maintain mutual visibility while walking on respective sides of a polygon's boundary.

We note that, in order to formally discuss the complexity of the discrete problems, and therefore of algorithms for solving them, some consideration must be given to the specification of input. Perhaps because less research has been performed on discrete environments, there is no widely accepted format for the input as for the continuous case (where a simple polygon is provided as a sequence of n points in the plane and commonly accepted as having input size n (de Berg, van Kreveld, Overmars & Schwarzkopf 2000, Preparata & M.I. 1985, O'Rourke 1994)). In particular, the inputs we consider here could be expressed in discrete form (either as a list of border cells or a description of the state of all cells) or in continuous form as sequences of border points and a scale factor. The different possible forms to describe the input lead to different ideas about the size of the problem instance. Figure 1 illustrates the difference. When measured in terms of vectors, the input sizes are 4 points, 8 points and 26 points. If the centres of the cells are used instead, we have 4 points, 5 points, approximately 21 (depending on the precise encoding scheme). If the input is supplied as the state (interior cell vs exterior cell) of each cell in a rectangular region, then all the instances have the same size input size. They all have similar size if a list of those forbidden (exterior) cells that constitute the boundary is supplied. For the most part of this paper we will accept the input as a grid with cells labelled as belonging to the interior or (exclusive) to the exterior. This is a common representation in computer games (Davis 2000). We accept that the input could be translated to other formats in polynomial time,

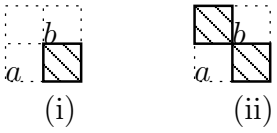

Figure 2: Navigable corners: (i) Cell b is reachable from a in one 8-connected move or two 4-connected moves. (ii) Cell b is not reachable from a in two 4-connected moves and is deemed not to be reachable in a single 8-connected move either. However, a and b may still be connected if there is a longer 4-connected path around the obstacles from a to b.

but not necessarily linear time. While our convention allows us to test a cell as belonging or not to the polygon in constant time, we will attempt to express our algorithms in terms of such a test, and to describe the complexity as a function of the complexity of this test. This would allow for other input formats.

This paper considers the discrete version of a problem where two agents must travel on minimal length paths while maintaining visibility. We provide the precise definitions and statement of the problem in Section 2. We take the opportunity to illustrate there that identifying if shortest paths form a solution is significantly different from the continuous case. Section 3 provides an algorithm for discrete polygons with or without holes. This algorithm is based on transforming the problem to finding connectivity in a graph of states. Therefore, the complexity is construction of the graph plus breadth first search (or Dijkstra's algorithm) on such a graph. In Section 4 we show an avenue to reduce this cost by using a smaller state graph based on the Euclidean shortest path. However, this restricts the algorithm to polygons without holes. We also characterise a subset of problem instances that always have a solution. This characterisation allows us to build decompositions of the general problem such that, if a simple reduction applies, it can be solved optimally.

2 Problem Description

Definition 1 *A* cell *is a square region of $\Re^2$ with unit sides. A point belongs to the cell if it is inside or on the border of the region. A cell is identified with its centre point.*

Definition 2 *A* discrete polygon *consists of a 4-connected set of cells in a rectangular grid surrounded by border cells. Unlike a "vectorially described" polygon, these border cells are not considered to be part of the polygon.*

Note that this 4-connected constraint is imposed on 8-connected problems as well. Figure 2 illustrates this. Cells are either interior cells or boundary cells.

Definition 3 *Two interior cells are* visible *if the line segment joining the centres of the cells does not intersect the interior of any boundary cells. Such a segment is called a* sightline.

Note that sightlines are permitted to touch a corner or the side of boundary cells. In the pattern recognition literature, visibility has been defined for discrete environments (a grid of pixels) in terms of an 8-connected path that is the rasterization a straight line segment joining the cells (Coeurjolly, Miguet & Tougne 2004, and references). We will not use this notion of visibility for this paper; however, we point out that there are efficient algorithms to test if two cells are mutually visible when the environment is provided as cells labelled as interior vs exterior (Coeurjolly et al. 2004).

Definition 4 *A* discrete path *in a discrete polygon is a (either 4 or 8)-connected sequence of interior cells beginning with the "start cell" and ending with the "target cell". A path is* minimal *if it has minimal length, where length is the smallest number of cells visited from the start to the target cell less one (or the smallest number of jumps).*

Note that we can compute the minimal continuous path in a discrete polygon, since such a polygon can be treated as a continuous simple polygon. The minimal *continuous path* in a polygon is a polygonal line (Guibas & Hershberger 1987). Also, in discrete environments, *discrete geodesic shortest paths* have been defined (Coeurjolly et al. 2004, and references) so that these paths are 8-connected curves that are segmented into *Discrete Straight Segments* (those 8-connected paths that are rasterization of a straight line between two cells). For the purposes of this paper, we do not use such notion of discrete path, as here we intend to reflect the motion restriction, rather than to emulate a continuous domain with a grid.

We are now in a position to formulate the problem.

Definition 5 *A discrete mutually visible path problem (*DMVPP*) consists of a discrete polygon P and four distinguished interior cells $\vec{s}_1$, $\vec{s}_2$, $\vec{t}_1$, $\vec{t}_2$. The line $\vec{s}_1$, $\vec{s}_2$ and the line $\vec{t}_1$, $\vec{t}_2$ are sightlines. A solution to a* DMVPP *consists of* discrete *paths μ_1 (from $\vec{s}_1$ to $\vec{t}_1$) and μ_2 (from $\vec{s}_2$ to $\vec{t}_2$) and a schedule (movements for each agent at each step) such that agents a_1, a_2 move from $\vec{s}_i$ to $\vec{t}_i$ without loosing visibility.*

Movement between cells is considered instantaneous with uniform cost. Visibility is only evaluated with respect to cell centres. In cases where the type of movement matters we will use DMVPP^4 or DMVPP^8 to disambiguate the discrete paths allowed.

We will use the following additional notation. Discrete paths will be denoted M_i or μ_i while continuous paths will be denoted π_i or Π_i (the second only being used for paths of minimal Euclidean length).

2.1 Distinctive Cases

One distinctive aspect from the continuous Euclidean setting is that the metrics in use here admit multiple paths of minimal length. In the continuous Euclidean case, it is always possible to find a schedule and paths of as short as within any $\epsilon > 0$ of the sum of the minimal path lengths (Fenwick & Estivill-Castro 2005). Moreover, if the start sightline and the target sightline do not cross and a solution exists, the minimal Euclidean paths always admit a schedule (Fenwick & Estivill-Castro 2005). However, we show here that there does not always exist a schedule for any pair of minimal paths in a DMVPP. Figure 3 illustrates this. If the points $\vec{t}_1$, $\vec{t}_2$ coincide with $\vec{s}_1$, the minimum path length is 6. The figure has highlighted a minimal path which preserves visibility. However there are paths of length 6 which pass through cell y which is not visible to $\vec{s}_1$.

Perhaps it is more reasonable to consider metrics that charge slightly more for diagonal steps. This discourages unnecessary zigzagging in minimal paths. While the set of minimal paths becomes more restricted, such metrics would not however, deal with the issue of minimal length paths which must contain diagonals but in which the position of those diagonals is not constrained (Lovell & Fenwick 2004). Again, Figure 3 shows that minimal paths trough y fail to maintain visibility while equivalent minimal paths with later diagonal steps do succeed and all paths would have the same cost even if diagonal steps are more costly than vertical steps.

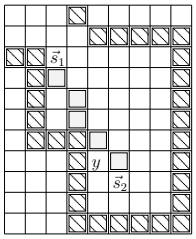

Figure 3: In the discrete, case minimal paths are not always part of a solution. The dark shaded cells are border cells while the light shaded cells show a valid shortest path between $\vec{s}_2$ and $\vec{t}_2$=$\vec{t}_1$=$\vec{s}_1$.

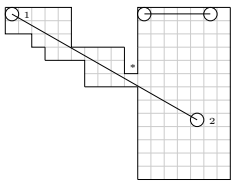

Figure 4: This DMVPP[8] has a solution, but no solution which uses minimal length paths (a_2 moves to the gap marked *, allowing a_1 to come out).

Figure 4 illustrates more challenges of the discrete case. This DMVPP admits a schedule and paths, but no path of minimum length has a schedule for a solution. Moreover, Figure 5 illustrates a DMVPP which has no solution at all. It is not surprising that the DMVPP is much harder than the continuous one, because finding Euclidean shortest paths in vectorially described polygon of genus zero can be performed efficiently (Guibas & Hershberger 1987) as a result of their uniqueness. Discrete 4-connected paths or 8-connected paths are not unique even in genus zero. If we consider polygons with holes, even the continuous case is far more challenging, and efficient solutions seem only possible when the path's homotopy type is given (Bespamyatnikh 2003, Efrat, Kobourov & Lubiw 2002, Hershberger & Snoeyink 1994).

A simple alternative to compute shortest paths in our discrete polygons is breadth-first search using the topological information resulting from the graph that has as nodes the interior cells and edges are present if two interior cells are adjacent (in 4-connectivity, the cells hold a North-South or East-West relationship, while 8-connectivity admits diagonals). Later, we will discuss using geometric information; however, we now present our first algorithm that solves a DMVPP in polynomial time by providing an optimal solution or declaring no solution is possible.

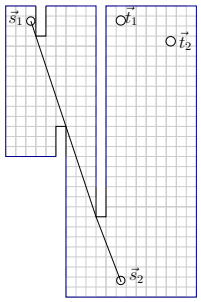

Figure 5: An unfeasible DMVPP. Any movement from the start cells results in visibility loss.

3 An Algorithm for Arbitrary Genus

Definition 6 *The* region $R_n(c_s, c_t)$ *is the union of all discrete paths from* c_s *to* c_t *with length* n *or less. The shortest path region is* $R(c_s, c_t) = R_n(c_s, c_t)$ *where* n *is minimal and* $R_n \neq \emptyset$.

Definition 7 *A* minimal successor *of a cell* c *in some shortest path region is a cell connected to* c *which is the successor of* c *on some minimal path to the target.*

A solution to a DMVPP can be found for any genus by exploring a "feasible states graph". If the exploration fails, then one learns that this instance of the DMVPP can not be satisfied (as in Figure 5). This is remarkable, since in the continuous case, algorithms are known for genus zero only (ie only for simple vectorial polygons). The "feasible states graph" FSG is constructed as follows. Each node of FSG is made of a pair (c_1, c_2) of interior cells of the polygon that are mutually visible (see Definition 3). A node represents a possible state in the travelling of the agents. An edge between two states (two nodes) (c_1, c_2) and (c'_1, c'_2) is in FSG if c'_1 is a successor of c_1 and c'_2 is successor of c_2. That is, the agents can move from the state (c_1, c_2) to the state (c'_1, c'_2). Also, c'_2 does not need to be different form c_2, so one agent may stay stationary for this step in the schedule.

Clearly that the starting sightline is a state, and thus a node in FSG; and so is the target sightline. Therefore, any algorithm that finds if these two nodes are connected in FSG determines a feasible solution or declares that the instance is infeasible.

A subgraph of FSG is the graph SMP of states on minimal paths (this graph has the same type of nodes as FSG but an edge between two states (two nodes) (c_1, c_2) and (c'_1, c'_2) is in SMP if c'_1 is a minimal successor of c_1 and c'_2 is minimal successor of c_2). A path from the starting sightline to the target sightline in SMP ensures that it is possible to solve the DMVPP using a minimal path on each agent. This results in an optimal solution of total minimal length. Since Figure 4 shows that failure to find a path in SMP just means that the solution may not be possible with minimal discrete paths, we use FSG with weights to compute a solution of minimal length. The edges of FSG are assigned a weight of 2 if both cells are different in the move represented by the edge while the edge will have a weight of 1 if only one cell moved. Then, a minimal path in the weighted FSG has length equal to the sum of the lengths of the paths of the agents. Our algorithm applied to Figure 4 with FSG weighted will find an optimal solution (although we know it is not made of shortest paths; the mutual visibility constraint forces agents to take detours).

The algorithms for exploring FSG or SMP can be breadth first search, A*, or Dijkstra's algorithm for the shortest path between a source and a target in a weighted graph. The challenge is to construct FSG and/or SMP. To find all nodes we need a means to determine if two cells are visible. If the polygon is converted to a vectorial description or provided as a vectorial polygon, determining the visibility of two points in its interior can be performed efficiently in simple polygons (Guibas, Hershberger, Leven, Sharir & Tarjan 1987). However, if we are given a data structure that rapidly returns if a cell is in the interior or the exterior (for example, in constant time), the visibility between two cells can be determined by testing if any of the cells that intersect the sightline is exterior (and the genus would not impact the complexity).

To find the edges of FSG we only need to test that this node exists (the cells are internal and mutually visible). For a node (c_1, c_2), there are at most 80 possible adjacent nodes in 8-connected (and 24

in 4-connected) corresponding to the 9 possibilities for an agent made of 8 moves and one for remaining steady. In the worst case, building FBS has complexity $O(vis(n)n^2)$ time where $vis(n)$ is the complexity of testing visibility between two cells and n is the number of interior cells (there are $O(n^2)$ nodes and the number of edges is linear in the number of nodes because the degree is bounded by a constant).

The construction of the SMP graph requires the computation of the regions $R(c_{s_1}, c_{t_1})$ and $R(c_{s_2}, c_{t_2})$ (alternatively, using a region builder algorithm, the computation of discrete minimal paths μ_1 (from $\vec{s}_1$ to $\vec{t}_1$) and μ_2 (from $\vec{s}_2$ to $\vec{t}_2$)). Useful data structures are the "successor maps" from cells in $R(c_{s_i}, c_{t_i})$. In these maps, each cell is linked to the set of minimal successors. If "flooding" by breadth-first search to compute μ_1 and μ_2 on a grid representation is used, then $R(c_{s_i}, c_{t_i})$ and the successor maps can be calculated at the same time. The nodes in the graph can be constructed beforehand using knowledge of $R(c_{s_i}, c_{t_i})$ or during exploration starting with (c_{s_1}, c_{s_2}).

While in the worst case the construction of SMP is also $O(vis(n)n^2)$ time, it typically requires $O(vis(n)|R(c_{s_1}, c_{t_1})||R(c_{s_2}, c_{t_2})|)$ time since a hash table implementation of the state map requires $\Omega(|R(c_{s_1}, c_{t_1})||R(c_{s_2}, c_{t_2})|)$ space. It also improves the complexity of the next phase, because now a search for the start sightline to the target sightline is linear in the size of the graph; that is $O(|R(c_{s_1}, c_{t_1})||R(c_{s_2}, c_{t_2})|)$. So the overall complexity is $O(vis(n)|R(c_{s_1}, c_{t_1})||R(c_{s_2}, c_{t_2})|)$ time.

In the next section we attempt to reduce the cost of calculating the regions $R_i(c_{s_i}, C_{t_i})$ for polygons with no holes.

4 Improved efficiency for genus zero cases

In order to reduce the cost for the algorithm in Section 3, we explore conditions that allow us to reduce the cost of finding the shortest path regions. We now develop algorithms for situations where some additional information is known and the polygon has genus zero. First, we study the case where we are provided with a discrete minimal path between the endpoints. We show how the rest of the region $R(c_{s_i}, c_{t_i})$ can be computed in Section 4.1. If such a path is not known, but Π (the Euclidean shortest path between the endpoints) is known, then we show in Section 4.3 that rasterizing Π will give more inexpensive ways to find a discrete minimal path between the endpoints, and thus, by the results to be presented in Section 4.1 an overall faster algorithm.

4.1 Region Builder

If one minimal discrete path is known, the shortest path region $R(c_s, c_t)$ can be calculated efficiently. The worst case complexity of $O(n)$ time, for the number n of internal cells in the polygon, is now $O(|R(c_s, c_t)|)$ time. Again, we assume that there is a constant time test to determine if a particular cell is an external cell or not. The following lemma shows that if we know a cell is in a discrete minimal path and we know one minimal successor, then the options for other minimal successors are limited.

Lemma 1 *Let P be a genus zero 8 or 4-connected discrete polygon. If a minimal discrete path travels in direction α at cell c to cell c^+, then no other minimal discrete path with the same endpoints travels from c in a direction outside $[\alpha - 90°, \alpha + 90°]$.*

Proof: First the 4-connected case. Without loss of generality suppose that $\alpha = 0$ (rotate P if necessary).

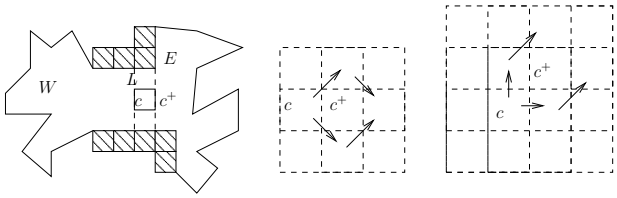

Figure 6: For a cell c and its successor c^+. (a) If c^+ in a minimal 4-connected path is East; then the target must be in the region E. (b) In 8-connected, if c^+ is East, then only either North-East or South-East can fork other minimal paths (and most be explored, for example the target could be two cells East of c). (c) In 8-connected, if c^+ is North East, then only moves North or East can fork other minimal paths (and most be explored, for example the target could be on North and one North-East move).

Cell c^+ is East of c. Let L be the set of cells (including c) from the vertical chord through c (these are visible to c). The polygon P is now partitioned into three parts L, the "East" part which includes c^+ (denoted E) and the "West" part (denoted W). Refer to Figure 6. We know that $E \cap W = \emptyset$ since the polygon has genus zero, so there is no path from W to E which does not pass through L.

No minimal path through c and c^+ can include any cell in L after c. If it did, then a move North would produce a shorter path. Thus, the target point must be in E. Note that this does not automatically exclude paths which travel North or South from c, since such paths could be postponing the single East move out of L in favour of vertical movement.

Any path which travels West from c will pay a cost of at least 2 additional horizontal moves to get through L. So, such a path cannot be minimal. This establishes the 4-connected case.

In the 8-connected case where $\alpha \in$ {North, South, East, West} the reasoning is the similar but L is defined differently. We let L be all cells with centres on the North-East diagonal and on the South-East diagonal extending from c to the polygon boundary. Cells in L are visible cells. The region E is defined as those cells 4-connected to c^+, while W is those cells not in $L \cup E$. The target cannot lie in L, since a diagonal sequence would be a shorter path than any path involving c^+. Similarly, the target cannot be in W since the cell where any path entered W could be reached in fewer steps by using diagonal steps and a vertical movement (or by not retracing steps). So in this case, the lemma is stronger, namely any direction not within 45° of α cannot produce a minimal path.

For the other 8-connected cases, we have $\alpha \notin$ {North, South, East, West}. We suppose $\alpha =$North-East and rotate the polygon otherwise. Let L be the set of cells with centres on a line through c with gradient -1 and visible to c. The region E is again the cells 4-connected to c^+ while $W = \overline{(L \cup E)}$. As before, if there is a minimal path through c,c^+, the target cell must be in E. If the target where in W, there would be a cell where a path moving West, South or South-West entered W. But, then this path would be shortened with fewer steps using movements in L.

Figure 6 shows a situation where this "large" range is required. □

We are now in a position to describe how to obtain $R(c_s, c_t)$ in time proportional to its size using as input a minimal discrete path between c_s and c_t. The algorithm labels internal cells. When cells are labelled as *used* they belong to $R(c_s, c_t)$ while they are labelled as *bad* they are in fact in the boundary of

$R(c_s, c_t)$. Initially, a stack S is loaded with the known minimal path, placing c_t first and working backwards so c_s ends up at the top of the stack. The initialisation also labels these cells as used and marks them with the distance to the target (so the target cell gets distance 0). All used cells c (with the exception of the target) have at least one used cell c^+ that they recognise as their successor.

Repeatedly, the top cell c in S is extracted.

1. **If c is already marked as used**, we use Lemma 1, to find neighbours of c that are possible forks of a new path from c to the target. These neighbours are pushed back into S; those that represent an angle of larger magnitude with respect to the movement c to c^+ are pushed first and the one with smallest angle in magnitude to the orientation c to c^+ is pushed last. These cells c^+ will not be labelled used nor bad; however, they will be marked with c as the used cell responsible for including them in S.

2. **If the top cell c in S is not a used cell** when extracted from S, then c points to the used cell r that caused its insertion. We use another stack T and insert in T the cell c and all cells in the path starting at c and following the boundary of used cells that is traced from r to the target. The distance values are attached to these cells in T enumerating from the value in r. The cells in the path in T are then classified as used in the region or marked as bad. This analysis proceeds by extracting the top cell c' in T.

   (a) **If c' has a neighbour cell c_n already used and with distance one less than** c', (so that c' would reach c_n and become part of a shortest path) then c' is marked as used with its corresponding value and pushed into the stack S; also, in this case, all of the cells in T are now part of a minimal path, so they are all extracted and as they come out of T, they are inserted in S as used, with the corresponding distance label and successor pointer. Note that in this case, T will be empty and the analysis of the path in it is finished.

   (b) **If c' has no neighbour cell already used so that c' can reach it and become part of a new minimal shortest path**, then c' is marked as bad and the analysis continues with the next cell in T.

4.2 Complexity and Correctness

Lemma 2 *Let P be a polygon In a 4- or 8-connected discrete polygon of genus zero, given μ a minimal discrete path between c_s and c_t, the algorithm labels a cell as used with its distance to the target if and only if it is in $R(c_s, c_t)$.*

Proof: We show that all cells labelled as used are in $R(c_s, c_t)$ and that no cells in $R(c_s, c_t)$ are missed.

The first part follows because the algorithm labels all cells in μ as used. Or alternatively, it labels all cells in a shortest path from c_s to some cell c, a shortest path from c to c' and a shortest path from c' to c_t. The path c_s, c is shortest because it is so built in S, the path c to c' is shortest because is so built and stored in T. The path c' to c_t is shortest because we have the shortest distances to c_t recorded. The concatenation of these 3 paths must be a shortest path from c_s to c_t since it has the same length as μ.

The second part is challenging. Since the polygon has genus zero, there can be no holes in $R(c_s, c_t)$. We proceed by contradiction. Assume there are cells in $R(c_s, c_t)$ which are not marked as used. Then, at least one of them must be a successor to a marked cell because these cells are in minimal paths from c_s to c_t and c_s is marked. We just pick the cell $c \in R(c_s, c_t)$ closest to c_s that was not marked. In order for c not to marked as used, it must either have not been checked at all or have been marked as bad. Lemma 1 tells us that all possible valid successors for all checked cells are considered. Thus, it must have been marked as bad. Cell c is only marked bad if the path which follows the border of the known region was extended as far as possible and failed to connect anywhere after and including c. The known region would need to double back so it was on both sides of c. This contradicts the statement that μ is minimal. □

The complexity analysis is as follows. Each cell is introduced into T at most once and into S at most once. All the work is proportional to the push and pop operations in these two stacks. This gives a runtime complexity proportional to the number of cells marked as used and those marked as bad. Since a bad cell is always a neighbour of a used cell, the bad cells are inside the boundary of $R(c_s, c_t)$. Thus, the total work of the algorithm is proportional to $R(c_s, c_t)$. However, for each cell we not only push it or extract it; we also query if its interior or exterior, and we need a data structure per cell to record if it is been labelled (an in that case, the label is used or bad), to record its successor and its distance to the target. We assume that we have a constant time function IS_INTERIOR$(c) \rightarrow$ *Boolean* as part of the given input and we will not charge this algorithm for reading the input and building this function (we may want to run many instances of a DMVPP on the same polygon).

However, we call the "properties storage structure" the data structure for maintaining used/bad labels, successor pointers and distance values for a cell indexed by its integer coordinates in the gird. Under this conditions, the analysis reveals an overall complexity of $O(I + Ak)$ time, where

- I is the cost to initialise the property storage structure,
- A is the number of cells in $R(c_s, c_t)$, and
- k is the complexity of property lookups and border checks for cells.

Since we can find the length $|\mu|$ of the given minimal path in time less than $R(c_s, c_t)$; the identifiers for cells can be normalised to a pair of integers in the range $[0, |\mu|]$. The property storage structure can then be a hash table for which modulus functions would be a "good hash function" (Gonnet & Baeza-Yates 1991). With such a good hash function, a table of size at least A would suffice to retrieve and update properties in $o(1)$ time.

Thus, the overall complexity is $O(A)$ time and space; this is a good result as the complexity is linear in the size of the output. Of course, in the worst case, the complexity can be intimidating because the region can have quadratic size in $|\mu|$. But, for most interesting polygons, this algorithm behaves in complexity much smaller than the number of interior cells of the polygon.

4.3 Covering an Euclidean shortest path

In order to construct the region in the previous section a minimal discrete path is required. In this section we show that, if the Euclidean shortest path Π between cell centres is known, then it can be rasterized to produce such a path. We do this by proving that Π is "covered" by $R(c_{s_i}, c_{t_i})$. While the end result applies to genus zero an intermediate lemma is more general.

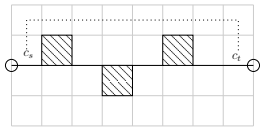

Figure 7: If a 4-connected discrete polygon is not genus zero, then minimum length discrete paths may not cover a continuous line segment. The horizontal segment is between the corners of two cells. Minimum length discrete paths (dotted lines) avoid obstacles, rather than covering the Euclidean shortest path.

Definition 8 *A set of cells C in a discrete polygon* covers *a continuous path π, if $\forall p \in \pi$ implies $\exists$ cell $c \in C$ such that $p \in c$.*

Definition 9 *The* discrete length *of a continuous path π, between two cells c_s, c_t in a discrete polygon, is the minimum value k such that $R_k(c_s, c_t)$ covers π.*

We now prove that for genus zero discrete polygons, the length of minimal discrete paths is equal the discrete length of the minimal continuous path(Π). We do this by proving that Π is covered by $R(c_s, c_t)$.

Lemma 3 examines individual line segments in Π and shows they are covered then Lemma 4 deals with Π as a whole.

The following lemma establishes the desired result for a single straight line segment. The Euclidean shortest path is made of a polygonal line where the vertices are vertices of the polygon (Guibas & Hershberger 1987) and the line segments are straight. Because the path is from c_s to c_t, the vertices of the Euclidean shortest paths would be corners of cells, or the two centres of the cells c_s and c_t.

Lemma 3 *Let P be a discrete polygon which is either 8-connected or (4-connected and genus zero) and Π a Euclidean shortest path between cells c_s and c_t. For each line segment (p_1, p_2) in Π, $\exists$ cells c_1, c_2 such that $p_1 \in c_1, p_2 \in c_2$ and $\forall$ point $p \in (p_1, p_2)$ there is a cell $c \in R(c_1, c_2)$ with $p \in c$.*

Note that this lemma establishes the base case for when Π has only one line segment. We employ a standard rasterization algorithm but state it explicitly for clarity.

If Π has more than one line segment, up to three cells could include a segment endpoint (a vertex of Π). We will accept any of these cells for the lemma.

Proof: For simplicity, we assume throughout this proof that the slope of the segment lies between $0°$ and $90°$ inclusive, other slopes follow by symmetry.

We first establish the 4-connected case. If both endpoints of the segment are cell centres, a minimum length discrete path can be constructed as follows. Beginning with the start cell c_s, add the current cell to the discrete path. If the segment leaves the cell through either the North edge or the East edge, then add as the next cell the neighbour cell the line enters. Make this new cell the current cell of the discrete path.

If the line segment leaves through the corner, then at least one of the cells to the North or East must be free of obstacles (add either free cell), then add the the North-East cell to the discrete path and make it the current cell. Continue until the current cell includes the target point.

This path has minimum length since it has length equal to the Manhattan distance between its endpoints. It also covers the segment.

Two cases remain to establish the 4-connected case. The next case is when at least one of the endpoints of the line segment is a cell corner rather than a cell centre. The path production algorithm can be modified in the following way. Suppose the start point p lies on a cell corner. We only require a slightly different initialisation. If the vertex p is

- in the North-East corner of c_1, we add c_1 to the discrete path and either one North or one East cell, followed by the North-East cell.
- the South-East corner of c_1, we make c_1 the first cell of the discrete path and next in the discrete path is the East cell,
- the South-West p corner of c_1, we start the discrete path with c_1, and
- the North-West corner in c_1, we start with c_1 and the North cell.

After this initialisation we continue with the previous algorithm.

Adding extra cells for a non-centred target works in a similar way. Note that the choice of c_1 is not restricted here.

The last case remaining is a horizontal or vertical line segment with endpoints on cell corners (and hence the entire of Π must fall on cell boundaries). In such a case, obstacles can touch the segment. The path can be constructed by adding cells along the line until an obstacle (or the target) forces a change of side. Such a path will cover the segment. It is of minimal length because the only way it could fail to be so is if there was a shorter path around an obstacle which forced a change in side. This will not happen provided the polygon is genus zero. In fact, any line segment which is not vertical, nor horizontal, would be covered by the method above by an optimal discrete 4-connected path, even if the polygon had genus different from zero. But, horizontal or vertical line segments may not be covered by $R(c_s, c_t)$ is the polygon has holes (see Figure 7).

Now, the 8-connected case is slightly more difficult. We will construct a discrete path for Π in the 8-connected case in a similar manner as for the 4-connected case, with two differences. First, we will break each cell into 4 quadrants with a horizontal dividing line through the centre of the cell and a vertical dividing line also through the centre of the call. These quadrants are named North-West, North-East, South-East and South-West in clockwise order. The second difference is that we will construct a discrete shortest path from the endpoints of the line segment; however, not every point in the line segment will be covered by this discrete path. We will need some further discussion to show that the straight line segment is covered by $R(c_s, c_t)$ by slightly modifying the path built by the method into another shortest path that covers regions not covered by the path before.

The algorithm, however, works again by incrementally building the discrete path guided by the cells intersected by the straight line segment. Recall that we assume the line segment has slope in $[0°, 90°)$ and other cases are handled by rotations and symmetry.

Let c_s be the initial current cell. When analysing the current cell, if the segment leaves the cell via

- the East boundary and in the East cell the segment intersects more than one quadrant of the East cell, move East,
- the North boundary and the straight line intersects more than one quadrant of the North cell, add the North cell s current,
- in all other cases, add the North-East cell as current.

Repeat the analysis of the current cell until the target is reached.

As discussed in (Lovell & Fenwick 2004) the minimum length for a rasterized line segment in 8-connected path is the maximum of the horizontal difference and the vertical difference. Since this method produces a path which is minimal by this measure, it is a minimal 8-connected path.

However, the discrete path produced covers the straight line segment only when such straight line has slope $0°, 45°$ or $90°$. To deal with this, we first assume, without loss of generality, that the segment has slope larger than $0°$ and smaller than $45°$. In this case, the path produced by the method described above will consist only of East and North-East movements and it sometimes moves North-East leaving some part of the straight segment below the path.

Consider the first North-East move which has some part of the segment below it. Following it will be a sequence (possibly empty) of North-East moves terminated by an East move. Since the slope of the segment is less than $45°$, a section of the discrete path composed of North-East moves will diverge from the straight line segment until an East move is encountered. This means that all the moves in the sequence leave uncovered parts of the segment below and hence all such cells are free of obstacles. Transposing the first and last moves in the sequence will produce a new path with sequence shifted to the right. This new path covers the points in the segment below the previous path.

If an East movement is followed by a North-East movement, then there may be some part of the segment above the path. In that case, the two movements can be transposed (and the process repeated to handle any other such cases). So the union of all paths produced this way will cover the segment (Note that none of these transformations increases the length of the path).

The same issues arises in the 8-connected case as well. Namely, the line segment may be horizontal or vertical along the grid that defines the cells. Similarly, some careful initialisation may be necessary. For cell c_1, if the start point is

- South-East, we add the current cell and move East,
- South-West add the current cell,
- North-East add the current cell and move North-east,
- North-West move North.

Once this initial move has been made applied, the method proceeds as before. Similar operations apply for the target cell c_2. These paths are still of minimum length.

Interestingly, running along cell boundaries does not create the problem in 8-connected as it does in 4-connected (and hence the lemma is true without the genus zero constraint). Because each (navigable) corner must have the three surrounding cells free, it is always possible to switch sides without extra cost, using a diagonal motion. Such a path is of minimal length and covers the segment. □

The previous lemma tells us about individual straight line segments, now we need to address the whole Euclidean shortest path and show that we cannot do better by taking a different path. Moreover, the lemma provides a linear algorithm in the length of the discrete path.

Lemma 4 *Consider a genus zero discrete polygon P, cells c_s, c_t and the Euclidean shortest path Π from c_s to c_t. In such a case Π is covered by $R(c_s, c_t)$.*

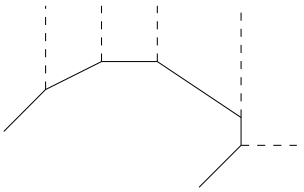

Figure 8: Decomposition of a path as described in Lemma 4.

Proof: Consider constructing the region $R(c_i, c_{i+1})$ for each straight line segment c_i, c_{i+1} of Π as described in Lemma 3. Note that we must be able to match the end-cells around the vertices because 3 cells around an (navigable) corner must be free (Figure 2) and Lemma 3 does not depend on our choice of endpoint. The union of these regions must be a subset of $R_k(c_s, c_t)$ which covers Π for some k. It remains to show that k is minimal.

Since the polygon is genus zero all paths between c_s and c_t must pass the same obstacles. Decompose the polygon as follows, for each segment in Π draw a vertical chord through the endpoint of each segment (these are vertices of the vectorial description of the polygon). If the segment was vertical draw a horizontal chord. See Figure 8.

Now let μ be a discrete path constructed by joining the covers of straight line segments as described in Lemma 3. First we address the 4-connected case. The Euclidean path Π cannot reverse its direction (with respect to the x-axis without a vertical or a horizontal segment). If such segments occur, travelling anywhere other than along Π in that section means the discrete path has paid extra to "go wide" and must pay that cost again to get back in line with Π. So no gains can be made in such sections. For the other sections, take the union of covers for adjacent segments. In such segments, μ will be monotonic with respect to horizontal and vertical directions and have length equal to the Manhattan distance between the endpoint. Hence μ has minimal length.

Now the 8-connected case. In sections for segments of Π with slopes between $\pm 45°$ inclusive, the minimum length of any path through that section is determined by the number of horizontal moves; so μ is minimal. Similarly to 4-connected no savings can be made on a vertical segment (although other paths may be able to match the optimal length). Further, any previously made gains will be lost in such sections.

Provided a sequence of segments is convex, there is no gain to be made by moving away from μ. Non-convex sections arise when there is a segment where the corners which terminate the segment are on "opposite sides". Any alternate path would need to enter the same point on the section boundary as μ or risk passing on extra cost.

So there are no paths shorter than μ from c_s to c_t, hence k is minimal and Π is covered by $R_k(c_s, c_t) = R(c_s, c_t)$. □ Note again, that we obtain a linear algorithm for finding a discrete path of minimal length guided by the minimum Euclidean path.

5 A Family of Solvable Instances

While we have shown that there are instances of an DMVPP that are not solvable, we show here a large, non-trivial family of 4-connected DMVPPs that admits a solution using minimal paths. The characterisation is the following theorem.

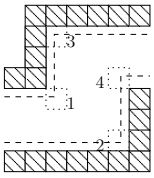

Figure 9: Cells 1, 2, 3 and 4 are examples of complete corners for a 4-connected DMVPP. In this case, cells 2 and 3 will not be processed by our algorithm, since they will be encountered while processing cell 1 and cell 4 respectively.

Theorem 1 *Consider a 4-connected* DMVPP *with non-crossing sightlines where the start cells are aligned (that is, they share either an x-coordinate or a y-coordinate) and the target cells are also aligned. In this case, a schedule exists for any pair of minimal discrete paths between their respective endpoints.*

In order to prove this result, we will make use of the following lemma.

Lemma 5 *A minimal path in a 4-connected genus zero discrete polygon cannot contain more than one contiguous sequence of cells from any horizontal or vertical chord of cells of the polygon.*

Proof: By contradiction. Suppose such a path and chord exist and cells c and d are in the chord and do not belong to the same contiguous sequence of cells in the chord. Since this is 4-connected, moving in the chord from c to d creates and even shorter path. This contradicts that c and d belong to a shortest path. □

We also will need the following definition.

Definition 10 *A complete corner on μ_i is a corner cell c which is aligned both horizontally and vertically with points on the other path. (See Figure 9)*

Note that endpoints are also considered corners (but are not necessarily complete).

We now proceed with the proof of Theorem 1. Note that what we need to do is construct a schedule. The proof proceeds by showing an algorithm that builds the schedule in steps. One of these steps is how to navigate a corner. If neither of the agents are at a corner, the algorithm indicates which the agent advances from a vertically aligned position along the optimal discrete paths μ_1 and μ_2. That is, without loss of generality, we assume the starting sightline is vertical; that is, the cells c_{s_1} and c_{s_2} are vertically aligned (and the horizontal case will follow by symmetry).

The algorithm shows how to move at least one agent, while preserving the invariant that the agents are aligned. For example, if the next step (in the minimal discrete path) of either agent is vertical, then the schedule includes (ie adopts) the move of those agents which have a vertical step. This preserves vertical alignment.

The algorithm will only face the following possibilities, and will chose the action that applies first. That is, the algorithm takes the following as a *decision list* (an option will only be taken if those above it do not apply).

1. One of the agents is at a complete corner.
2. One of the agents' next move is vertical.
3. Both the agents' next moves are horizontal in the same direction.

In fact, the entire proof is based on showing that it is impossible to reach the following situation.

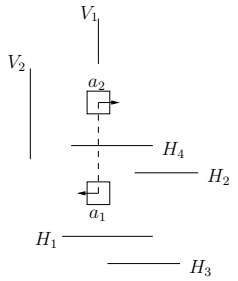

Figure 10: Possible positions for the target sightline T if the agents move in opposite directions.

Definition 11 *A scenario where the agents are vertically aligned and both of the agents' next step in a minimal discrete path is horizontal, but in different directions (ie, one West and one East) is called a non-schedule point.*

Lemma 6 *If the current situation of two agents that have travelled along minimal paths maintaining visibility is that they are aligned vertically, and they are not at a corner, then a non-schedule point is impossible.*

Proof: First we show that a non-schedule point (Definition 11) cannot actually occur. Without loss of generality assume that a_1 and a_2 are vertically aligned. Figure 10 shows possible positions for the target sightline T.

- First suppose T is vertical. In this case, we have two further possibilities
  1. In the first, the position (for the target sightline) is to be vertically aligned with the current positions of a_1, a_2 (illustrated as V_1 in Figure 10). This is not possible. If T were visible from a_2, then the non-vertical moves by the agents cannot possibly be optimal. If T were not visible, then the agents would need to navigate a corner to reach it, and this case would reduce to the next (which we illustrated as V_2).
  2. The target vertical sightline T is to one side of the agents (illustrated as V_2. We argue this is also impossible, because one of the agents (in the figure it would be a_2) moves away from T. In order to reach T, this agent must cross the vertical chord through the agents' initial positions. However, since the polygon is genus zero, the crossing cell would be visible to a_2. Using Lemma 5, we reach the contradiction that the discrete path for a_2 is optimal.
- Now we analyse the horizontally aligned possibilities for T. This also has several cases.
  1. Consider first when T is as illustrated by H_1 (above or below both agents with an endpoint on either side of the chord through a_1, a_2). One of the agents would need to end at the East end of H_1. It cannot be a_1 by the same reasoning which ruled out V_2. Agent a_2 is no good either, since in order to reach H_1 there would need for some later cell on μ_2 which is horizontally aligned with a_1 forming a complete corner. This which is a contradiction. This eliminates H_1 and also H_3 (H_3 is the same as H_1 but without endpoints on both sides of the line through a_1, a_2).

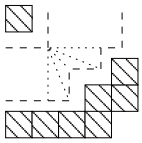

Figure 11: Sightlines in a complete corner.

2. The case illustrated as H_2 is a target sightline whose y coordinate is between a_1, a_2. This is ruled out by the same reasoning as V_2.
3. However, ruling out a position like H_4 (a target that is crossing the sightline between a_1, a_2) takes a little more work. Notice that we cannot have a_1 heading for the East end since this would make μ_1 non-minimal. Further, a_1 and a_2 cannot be at their starting positions, since we do not allow the initial and target sightlines to cross.

   So where were the agents before now? Being further apart vertically but still on opposite sides of H_4 would not solve the problem of the initial sightlines crossing. If they started on the same side of H_4, then the chord through H_4 and Lemma 5 contradicts the minimality of its path.

   What is the orientation of S (the start sightline for the problem)? A horizontal S would have required at least one of the agents (say a_i) to be level horizontally with and visible to T in the past. But this means that μ_i cannot be optimal (Lemma 5).

   For a vertical S one of three problems occurs.

   (a) S is vertically aligned with and visible to a_2. In such a case one of the agents will intersect the chord through H_4 more than once.

   (b) Whichever side of the chord through a_1, a_2 the sightline S is on, one of the agents is moving back in that horizontal direction. If an obstacle did not force this reversal of direction, then at least one of the paths cannot be minimal. If it was forced by an obstacle, then both agents would have been on the same side of the chord through H_4 reducing to item 3a

□

We now complete the algorithm. Here is the decision list and its actions.

1. When one of the agents is at a complete corner and the two agents are vertically aligned, use symmetry or rename the agents so a_1 is on the corner. Navigate the corner from vertical alignment to horizontal alignment as follows. Agent a_1 waits while a_2 moves until it is next horizontally aligned with a_1. We use Figure 11 to illustrate that visibility is preserved during these moves. No obstacles can be inside the region bounded by μ_2 and the sightlines when the agents are aligned, so the only way for any of the sightlines to be blocked is if μ_2 reversed direction and moved behind an obstacle. Lemma 5 excludes such a possibility.

   Both agents are never at a complete corner simultaneously (like that in Figure 12) because then at least one of the paths cannot be minimal. Since the polygon has genus zero, there is no reason to turn away from the direct path.

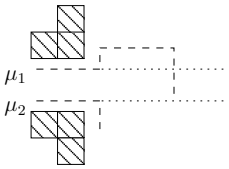

Figure 12: Why a schedule never needs to complete corners starting at the same time. In order to produce a complete corner μ_1 must move north. In order to align with μ_2's corner it must travel South since the polygon is genus zero this makes μ_1 non-minimal.

2. When one of the agents' next move is vertical the schedule adopts the relevant agents' vertical move.
3. If both agents' moves are horizontal in the same direction, then the schedule adopts both moves.

For space reasons we do not show here that it is impossible for alignment to change from vertical to horizontal in a single move unless the agents are occupying the same cell are some point and then how we deal with agents occupying the same cell.

It is remarkable that DMVPPs with aligned sightlines can be solved so efficiently (compared to our previous algorithms), since we only need to find minimal discrete paths for the agents and then the schedule can essentially be found in linear time. The next subsection shows that this also has implications for the generic instances.

5.1 Impact on All Instances

One approach to deal with problems where the endpoints are not aligned is to decompose the problem into (up to three) sections. For example, find the first cell on μ_2 which is aligned with c_{s1}. Denote this cell s_2'. Hence if a schedule exists to move a_2 from c_{s2} to s_2' (and possibly another schedule to handle the target cells similarly), we have a complete schedule since Theorem 1 guarantees us a schedule if the endpoints are aligned.

Lemma 7 *Consider a* DMVPP *where the start cells are not aligned with agents a_i and a_j. Further, suppose $\exists s_i' \in \mu_i$ such that s_i' is (vertically or horizontally) aligned with c_{sj} and s_i' is the earliest such point. In such a case, the subproblem of moving a_i from c_{si} to s_i' along some minimal path and a_j stationary has one of two outcomes: there is a schedule which uses a minimal path from c_{si} to s_i' or there is no schedule at all for any path.*

Proof: If a schedule exists for the some minimal path from c_{si} to s_i', then we are done. We prove the alternative case where no minimal path will do by contradiction. Suppose that no minimal path exists from c_{si} to s_i' which has a schedule but there exists a longer path that does have a schedule. See Figure 13. Assume that there is no minimal solution for which a schedule exists. Let c be a cell with maximal distance along some minimal path such that there is a schedule to move a_i from c_{si} to c without loosing visibility of a_j and no further movement along any minimal path is possible.

Without loss of generality assume that S, the cell to the south of c is a minimal successor. Note that visibility of this cell must be blocked. Now consider the possible positions of c_{sj}. It cannot be directly south of c since a blockage of S implies that c is not visible either. It cannot be directly East of c since then

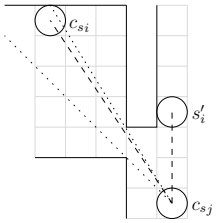

Figure 13: Part of a DMVPP before the start cells are aligned. Dashed lines are sightlines, dotted lines show the visible area between the two obstacles.

S cannot be a minimal successor (this also excludes North). What about somewhere between South and East? Since visibility of S is obstructed and the polygon is genus zero, no path can travel through or below S and retain visibility. If the cell E East of c is a border cell, then there can be no possible path. Alternatively E must be a minimal successor, which must be blocked (else c is not maximal distance along). The same reasoning as for S prevents a path moving past E, so there is no possible path. So we have our contradiction. The other cases follow by symmetry. □

6 Final Remarks

We have produced very effective algorithms for the discrete version of two agents travelling inside a polygon of arbitrary genus and maintaining visibility while minimising the distance travelled. It is not hard to see that our algorithm generalises for even more agents. If k agents are travelling, the FSG graph would be made of k-tuples indicating the current cell of each agent. The SMP graph would have nodes if the cells have the desired visibility constraint (for example, a spanning tree of visibility or the complete visibility between the agents).

Also, we have produced algorithms to rapidly construct this graph from the heuristic provided by the Euclidean shortest path in a vectorial polygon. In the case of genus zero, we have shown that the heuristic actually produces the necessary minimal discrete paths. We also have found that aligned starting and ending positions allows very efficient solutions in the 4-connected case. Since the current solution for the continuous case are not as comprehensive, we will explore how effective a discrete solution is as an heuristic for a continuous case if one were to rasterize a vectorially described polygon.

References

Bespamyatnikh, S. (2003), 'Computing homotopic shortest paths in the plane', *Journal of Algorithms* **49**(2), 284–303.

Coeurjolly, D., Miguet, S. & Tougne, L. (2004), '2D and 3D visibility in discrete geometry: an application to discrete geodesic paths', *Pattern Recognition Letters* **25**(5), 561–570.

Davis, I. (2000), Warp speed: Path planning for star trek: Armada, *in* 'AAAI Spring Symposium on AI and Interactive Entertainment.', AAAI Press, Menlo Park, CA.
URL: *www.qrg.cs.northwestern.edu/aigames.org/2000papers.html*

de Berg, M., van Kreveld, M., Overmars, M. & Schwarzkopf, O. (2000), *Computational Geometry Algorithms and Applications*, second edn, Springer Verlag, Berlin.

Efrat, S., Kobourov, S. & Lubiw, A. (2002), Computing homotopic shortest paths efficiently, *in* 'Proceedings of the 10th Annual European Symposium on Algorithms', pp. 411–423.

Fenwick, J. & Estivill-Castro, V. (2005), Optimal paths for mutually visible agents, *in* X. Deng & D.-Z. Du, eds, 'Algorithms and Computation, 16th International Symposium ISAAC', Lecture Notes in Computer Science 3827 Springer, Sanya, Hainan, China, pp. 869–881.

Fink, E. & Wood, D. (2003), 'Planar strong visibility', *International Journal of Computational Geometry and Applications* **13**(2), 173–187.

Flocchini, P., Prencipe, G., Santoro, N. & Widmayer, P. (2005), 'Gathering of asynchronous robots with limited visibility', *Theoretical Computer Science* **337**(1-3), 147–168.

Gonnet, G. & Baeza-Yates, R. (1991), *Handbook of Algorithms and Data Structures, 2nd edition.*, Addison-Wesley Publishing Co., Don Mills, Ontario.

Guibas, L., Hershberger, J., Leven, D., Sharir, M. & Tarjan, R. (1987), 'Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons', *Algorithmica* **2**, 209–233.

Guibas, L. J. & Hershberger, J. (1987), Optimal shortest path queries in a simple polygon, *in* 'SCG '87: Proceedings of the third annual symposium on Computational geometry', ACM Press, New York, NY, USA, pp. 50–63.

Hershberger, J. & Snoeyink, J. (1994), 'Computing minimum length paths of a given homotopy class', *Computational Geometry: Theory and Applications* **4**(2), 63–97.

Ickling, C. & Klein, R. (1992), 'The two guards problem', *International Journal of Computational Geometry and Applications* **2**(3), 257–285.

LaValle, S. (2006), *Planning Algorithms*, Cambridge University Press, UK.

Lovell, N. & Fenwick, J. (2004), Linear time construction of vectorial object boundaries, *in* M. Hamza, ed., 'IASTED Signal and Image Processing Conference', ACTA Press, pp. 914–919.

O'Rourke, J. (1987), *Art Gallery Theorems and Algorithms*, Oxford University Press, New York.

O'Rourke, J. (1994), *Computational Geometry in C*, Cambridge University Press, UK.

Preparata, F. & M.I., S. (1985), *Computational Geometry An Introduction*, Texts and Monographs in Computer Science, Springer-Verlag, New York.

Sack, J. & Urrutia, J., eds (2000), *Handbook of Computational Geometry*, Elsevier.

Schuierer, S. & Wood, D. (1999), 'Multiple guard kernels of simple polygons', *Journal of Geometry* **66**, 161–168.

Widmayer, P., Wu, Y., Schlag, M. & Wong, C. (1986), 'On some union and intersection problems for polygons with fixed orientation', *Computing* **36**, 183–197.