Visibility-Preserving Movement for Teams — Algorithms for Walking Minimal Paths.

by

Joel Fenwick

B.CompSc(hons), B.Math

School of ICT, Griffith University

Submitted in fulfilment of the requirements of the degree of Doctor of Philosophy.

December 2007

Abstract

There is a large body of research into finding short (if not shortest) paths in a wide variety of environments including graphs, polygons and the two dimensional plane with polygonal obstacles, to name just a few. Much of this research has been focussed on moving a single (possibly complex) entity through the environment under a variety of cost metrics.

In recent years, relatively inexpensive robots have become more accessible, ranging from basic kits to complete units such as the Sony Aibo™. At the same time, virtual environments have been increasing in sophistication. For the past decade or so, there has been research interest (for example RoboCup) in robots or agents (in both real and virtual environments) acting in teams rather than in isolation. In most cases, team members will need to communicate in some way in order to complete their tasks. While a variety of communication mechanisms exist, an easy to understand abstraction for communication is line-of-sight. If two entities can see each other they can communicate. For example, infra-red or gesture based communication.

In this thesis, we examine a question which combines teams of agents, visibility and short paths. Consider a number of agents (moving entities) which need to move from one location (a start point) to another (a target point), while keeping each other in sight. For example, two people wish to travel from their places of work, to their homes, while keeping an eye on each other for safety reasons. We wish to minimise the distance the agents travel, but preserve visibility. We abstract this situation to have the agents moving inside a simple polygon. A problem instance in this domain consists of a simple polygon as well as start and target points for each of the agents. A solution to an instance consists of a path for each of the agents and a schedule for moving along those paths such that visibility is preserved. An *ideal* solution is one which uses the shortest paths between each start and target point. The start sightline S is the line segment joining the start points, while the target sightline T is the line joining the target points. When agents move at the same constant speed, we will show that ideal solutions always exist if S and T do not cross. If S and T do cross, then we show how to determine if an ideal solution exists. For cases where an ideal solution exists, we show how to produce it.

We term an instance with two agents represented by points, moving inside a polygon in $\mathbb{R}^2$, an MVPP(Mutually Visible Path Problem). We provide algorithms to produce schedules in time linear in the number of vertices of the polygon and the size of the schedule. In cases where an ideal solution does not exist, we show how to produce a solution with length arbitrarily close to that of the ideal solution. We also show that a small modification is enough to allow all instances to have ideal solutions. Specifically, allowing the agents to choose two movement speeds is sufficient to allow ideal solutions for the few pathological cases where visibility is normally too constrained to allow a solution.

An MVPP-n is an instance containing n agents where the start points for the agents are collinear (the same applies to the target points). We consider two forms of visibility constraint: the *complete constraint* and the *connected constraint*. Under these conditions we show that the majority of instances admit either an ideal solution or one which approximates the ideal. We show how to produce solutions in these cases. We do not know if all instances which do not admit an ideal solution will admit an approximation instead. However, we describe properties which instances which do not admit approximations must have if they are to exist.

We also discuss n-agent instances where the start points are not collinear. Here we require that the hulls of the start and target points can be separated by non-crossing chords. For the complete visibility constraint, we show how to compute ideal solutions in all cases. Under the connected constraint, we show how to produce ideal solutions, provided a constraint is met on the visibility graph formed by the start (and target) points.

Finally, we examine problem instances where agents are not represented by points. We consider agents represented by circles in the plane and give sufficient conditions for ideal solutions to exist. We also discuss a discrete form of MVPP, the DMVPP. A DMVPP is a two agent MVPP in a discrete polygon (a polygon composed of cells rather than points). We give efficient algorithms for computing ideal solutions in such cases (where they exist).

This work has not previously been submitted for a degree or diploma in any university. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the thesis itself.

Joel Fenwick

Thanks to my supervisors Vlad Estivill-Castro and René Hexel for their guidance and their patience during the long period of time when this thesis was "nearly done".[1]

Thanks to my colleagues Nathan Lovell, Artak Amirbekyan and Stuart Seymon (especially to Nathan and Artak for letting me dabble in their research).

Thanks to Allen and Liz for large quantities of "Settlers of Catan".

Thanks to Jono, Craig and Patrick for providing an environment where I could think about something else.

Thanks to Val for the interesting conversations.

Thanks to my family (especially Julie) for their support.

Thanks to God, for keeping me (relatively) sane.

[1]This research was funded in part under ARC (Australian Research Council) Discovery Research Grant. Project DP 0209818.

Contents

Chapter 1

Introduction

Questions about how to reduce the cost of journeys have a long history, from Euler and the bridges of Königsberg to the still troublesome travelling salesman problem. While computation cost is prominent in much of computer science, in many problems involving navigation, the distance travelled is a far more pressing concern than the processing required to guide the motion. In some cases this is because the processing time is not significant compared to the time required to carry out the motion. In other cases it may be that power required to drive motors is greater than that required to run the CPU.

A first step for problems (navigation and otherwise) is to attempt them for a single entity (or processor). For example, for many years, Dijkstra's algorithm has been the method of choice for computing shortest paths in graph based environments [Ede87, O'R87, CLR90, Lav06]. Many tasks can be solved using a single entity; however, there are benefits in using teams to attack a problem. Redundancy may allow the team to continue to operate in the event of the failure of an individual (although probably at reduced capacity). Entities may be able to monitor each other to detect problems or threats in the environment. In some cases, a problem may be solvable using multiple simpler, cheaper units instead of a single complex one. While the most powerful computing hardware may be very expensive, low to medium power units can be reasonably light and so made (physically) mobile. With more entities, the productivity lost in switching between different sub-tasks can be reduced. Improvements could come from things as simple as being able to operate in different locations simultaneously.

While humans could solve some of these problems with relative ease (at least for small instances), the operating environment may be inhospitable or inaccessible for them. For ex-

ample, surveying the ocean floor [YJBB07] or inspecting fire sites [JQW07]. Even if humans can enter the environment, it may still be desirable for agents[1] to act without significant guidance. This also applies to virtual environments and simulations which have increased in sophistication in recent years.

Introducing additional entities does present challenges, however. Tasks need to be decomposed and assigned to team members. Arriving at an efficient decomposition may be non-trivial. For example, Zlot and Stentz [ZS06] discuss using auctions to distribute tasks among robots in a team. In that work, different robots might have different "opinions" about the cost of different parts of the overall task. The presence of a greater number of entities may make diagnosing problems harder. For example Nair et al. [NTMR04] consider teamwork in agent systems and the need to understand the behaviour of the team as a whole rather than just individual members. Coordination may be required between team members to ensure they do not interfere with each other. As an example, consider the extra care which must be taken with multi-threaded software to ensure that critical operations are threadsafe and that deadlocks cannot occur. A centralised approach with a single entity making the decisions may seem to solve this problem, but this may be inefficient or impossible in some circumstances. For one thing, this adds single point of failure (and possibly a communications bottleneck) in the controller. Even centralised planning and coordination may need to reconcile different data about the world coming from different entities.

Often entities will need to be able to communicate. Different communication methods may have significantly different properties such as maximum range, effect of reflections and transparency of certain materials. However, an easy-to-grasp abstraction for communication is line of sight. "In the absence of sophisticated satellite receivers or high power devices, a common constraint for successful communication is maintaining 'line-of-sight' between agents." [RD01] That is, if two agents can see each other, then they can communicate. If entities are able to echo messages (act as repeaters), then communication can be extended to areas where it could not otherwise reach. Moving to preserve visibility can also have other benefits. For example, navigation could be improved by using other team members as navigation landmarks [RDM01].

[1] The term *agent* is widely used in the literature (for example in agent theory [Woo02] and artificial intelligence [Nil98]). In this thesis we will use the term agent to refer to an entity which is capable of moving in its environment. We do not assume other properties sometimes associated with agents, such as independent goals.

In this thesis, we consider a problem domain which combines short paths for teams with a visibility requirement. Suppose there are two people who wish to walk from their places of work to their homes. To ensure their safety, they wish to coordinate their movements so that they can always see each other. One way to accomplish this, would be for them to meet at work, travel together, then separate when one of them gets home. However, there may be shorter routes which also meet the visibility requirement.

A common abstraction in navigation problems is to consider movement inside a polygon. We will discuss polygons in continuous space, that is, the boundary of the polygon is composed of line segments and all locations are points in $\mathbb{R}^2$. We call a problem which involves two points moving inside a polygon, a *mutually visible path problem* or MVPP.

Solving these problems requires finding paths to travel along, as well as a sequence of movements (a schedule) along those paths which preserves visibility. While there is almost always at least one set of paths which will work, they may be much longer than the shortest paths from source to target. In the example above, two people may have short journeys from work to home. If those places are a long distance away from those of the other person, then having the people meet and move on the same path will impose a large extra cost. It is not clear at first glance, how short the paths can be and still allow visibility to be preserved.

The task of finding shortest paths in two dimensions is reasonably well known. So, rather than trying to compute shortest paths for groups of agents directly, we will use the shortest path for each agent individually as the basis for a solution. In the continuous setting, we will assume that these shortest paths are provided to the algorithms we will describe. This thesis will provide an efficient test to determine whether or not the shortest paths admit a schedule. For cases where shortest paths are not usable, we describe a class of paths which both admit schedules and approximate the shortest paths (in the two agent case). We give an efficient algorithm for computing these replacement paths and we show how to produce schedules for shortest paths or their replacements.

We place very strong restrictions on the movement and vision of the agents involved. Our agents can only move at a single fixed speed or not at all. They cannot retrace their steps (that is, they cannot move to the same point on their paths more than once). Agents are considered to be visible through even the narrowest gap. We show that solutions exist (for the majority of instances) even under these very strict constraints. So the results and schedules produced under these conditions can also be applied when the conditions are relaxed, for

example, if backtracking or arbitrarily varying speeds are permitted. We will show that arbitrary speeds are not required and that a small number of permitted speeds is sufficient to make all two-agent instances solvable.

In many navigation problems, increasing the number of entities or dimensions involved makes them significantly more complex or expensive to solve. For example, each extra agent added to a system multiplies the number of possible states. If two agents form a team with 100 states and a third agent with 10 states is added, the number of states for the team could rise to 1000. We note that there are other difficulties associated specifically with increasing the number of dimensions. For example, many geometric algorithms in polygons in 2-D divide the polygon into triangles as a first step. This operation does not generalise to three dimensions. That is, it is not always possible to divide a polyhedron into tetrahedrons [O'R87]. Since this thesis discusses the plane, the issues involved in higher dimensions do not concern us. This thesis identifies several classes of instances in our problem domain which are not dramatically impacted by adding additional agents.

Although we specifically talk about line-of-sight as the minimum needed for communication, other interpretations of the sightline could be used. For example, when two agents move along paths, the sightline sweeps out the area between the paths. The agents may be required to clear an area in an evacuation or security scenario, in which case the sightline could be a sensor beam to detect if people have attempted to move back into the already cleared area. The same technique could be used to check for unexpected obstructions in the environment. In work on building maps by other researchers, a robot detects obstacles indirectly by noting that it can no longer see its partner [RDM01].

It is also not necessarily the case that all members of a team are identical or have the same capabilities. Security personnel could move through an environment with others agents moving so as to remain between them for safety. Alternatively, some components may be too expensive or require too much power, to allow all members to use them. For surveys or detailed examination of an area, a navigator could be positioned on each end of a line while entities with other sensors (for example, ground-penetrating radar or metal detectors) could be positioned between them.

With more than two agents, more complex spatial relationships between agents can occur. When moving into a dangerous or complex environment, it may be desirable to preserve a path to the exit. This could be to leave the area in the event of danger or merely for the purposes

of communication. Team members could be used as mobile landmarks (or communication repeaters). In those cases, line-of-sight provides a path to the exit. We show how to plan movements from an exit in an environment, to specified locations where line-of-sight must be preserved.

For example Qian et al. [JQW07] discuss a robot called "Ferret" which is designed to scout fire areas. Significantly, when Ferret loses radio contact with its controllers, it retreats to a known safe location. In this type of situation (even if not for this particular robot) extending communication range with mobile communication nodes would be desirable.

In navigation problems in general, it is not always possible to know the environment perfectly and plan ahead. There are many situations where the details of the environment emerge only as the agent moves. For some tasks, there are algorithms which cope with such unknown data. However, in order to evaluate the efficiency of these algorithms, an idea of the best possible solution is required. That is, a baseline to compare algorithms against. With the exception of a small class of pathological cases, this thesis proves a lower bound on the distance travelled in the best possible solution. In a large class of instances, this bound is tight and we show how to produce schedules which achieve the lower bound. In other cases, we show how to produce schedules which approximate the lower bound.

The first part of the thesis deals with points in continuous space. However, navigation problems and environments are not always modelled in this way. In the second part, we consider alternative models for our agents. Firstly, when modelling the movement of a robot, its physical dimensions may need to be considered (the piano movers problem [SS83] is just one example of this). With this in mind, we consider circular agents. This simplifies matters since we do not need to model the precise shape of the agent, just a bounding circle.

Secondly we discuss discrete environments. Discrete representations appear in motion planning and robotics research and may be employed for a number of reasons. It may be computationally inefficient to represent all the detail of the environment. Alternatively, the agent's sensors may only allow it to determine its position to be somewhere within a small area. In other cases integer coordinates may be all that is available. This applies particularly to games and other virtual environments where grid representations are common.

In summary, this work can be viewed in a number of ways:

1. Solving a velocity planning problem[2]. Since we want an ideal solution where one exists, in many cases we produce schedules for moving along known paths, in this case the shortest paths between the endpoints.

2. Solving a problem in constrained path planning. We are finding paths which preserve the visibility constraint. We try the shortest paths first but if that fails, then we need to produce new paths which approximate the ideal solution.

3. Providing an offline solution for use in future competitive analysis. As stated above, we give a lower bound on the lengths of paths under very strict conditions.

1.1 Overview

The rest of the thesis is laid out as follows. Chapter 2 discusses some related work. Questions about navigation and/or movement appear in a variety of disciplines including robotics, motion planning and agent theory. However, the closest work to this thesis, lies in the field of computational geometry. We briefly review the progress in computational geometry regarding navigation with some visibility requirement in place. While there is work about placing stationary guards or the movement of individual guards, there is not a lot of work combining visibility, minimal movement and multiple agents (the two guards problem [IK92] and work on moving ladders [LS85] are examples).

To the best of our knowledge, maintaining visibility while minimising some cost based on distance travelled, was first examined by Mitchell and Wynters [MW90a, MW90b] who allowed agents to move under an arbitrary (continuous) motion function. While their algorithms implicitly put some restrictions on movement (for example they do not use backtracking), this thesis shows that arbitrary speeds are not truly required for all cases with two agents. In fact, we show that with constant common speed, all non-crossing cases are solvable under our tighter movement restrictions. Mitchell and Wynters also raised the question of using more than two agents. While the class of instances for n-agents is complex, we address two subclasses which form a natural generalisation of the two agent problem.

[2]See Kant and Zucker [KZ86] or Chapter 2 for a definition.

Part I

Part I discusses agents represented by points moving in a continuous polygon.

Chapter 3 describes MVPPs. That is, moving two point-agents from their start points to their target points through a polygon with no holes. We label the line segment joining the start points S and the line segment joining the target points T. A schedule is a sequence of movements for the agents along particular paths. We refer to a schedule which uses shortest paths as an *ideal solution* for that instance. We show how to efficiently determine whether or not an instance will admit an ideal solution. This is based on two main ideas: whether or not S and T cross, and an important concept in this thesis called *trivial visibility*. For those instances which do have an ideal solution we show how to produce a schedule in $O(\|\Pi_1\| + \|\Pi_2\| + \mathcal{O})$ time (where $\|\Pi_1\|$ and $\|\Pi_2\|$ are the number of segments in the paths and $\mathcal{O}$ is the size of the schedule). For those instances which do not have an ideal solution we show how to produce replacement paths which approximate the shortest paths as closely as required. These paths can be constructed in time linear in the number of segments in the path being approximated.

This chapter shows a general approach which is applied in later chapters. Instances are classified as *crossing* or *non-crossing*, depending on whether S and T cross. The task of schedule production is broken into three parts: the Starting task, the Connecting task and the Finishing task. It turns out that most of the complexity occurs in starting and finishing a schedule.

Beginning in Chapter 4, we extend MVPP instances to allow n agents instead of 2. The general n agent problem (start and target points in arbitrary position) is too complex for a single research project. However, we present a natural generalisation of the two agent case which forms a subset of the general problem. The start points for all agents be collinear (a similar restriction applies to the target points). We call this new type of problem an MVPP-n.

Having more than two agents raises the question of what sort of visibility constraint should be used. In this thesis, we discuss two constraints. Either every agent must be able to see every other agent directly (the *complete visibility constraint*) or the graph formed by the agents' visibility must be connected (the *connected visibility constraint*).

Following the approach from Chapter 3, MVPP-n instances are divided into crossing and non-crossing instances. Chapter 4 considers non-crossing instances. We show that all

such instances admit ideal solutions and how to produce those solutions. This requires the construction of bounding paths, that is, paths which surround all the other paths. We show how to construct these paths in either $O(n \log n + (\mathcal{I} + \mathcal{L}) \log(\mathcal{I} + \mathcal{L}))$ or $O(n^2 + n\mathcal{L})$ time (where n is the number of agents, $\mathcal{I}$ is the number of intersections between paths of different agents and $\mathcal{L}$ is the total number of path segments for all agents).

Crossing instances are more complex and Chapters 5, 6 and 7 discuss this case in detail. Here we provide constructive proofs for the existence of solutions. We show the conditions required for ideal solutions to exist and how to produce them. Where ideal solutions do not exist, we give the conditions required for solutions which approximate the ideal to exist (and if they do, how to produce those solutions). We identify those shapes which make instances difficult to solve.

In Chapter 8 we discuss two extensions for MVPPs. First we show that allowing two speed ratios for the agents is sufficient to allow ideal solutions to all two agent instances. Secondly we consider a class of n-agent instances where the start (and target) points are not collinear. Provided the convex hulls of the start and target points meet certain restrictions, all complete constraint instances admit ideal solutions.

In contrast to the preceding material, for these type of instances, the connected visibility constraint is more complex than the complete constraint. For the connected visibility constraint we need to limit the instances we consider but once this is done, the remaining instances all admit ideal solutions.

Part II

Part II covers non-point representations of agents.

Chapter 9 considers continuous instances where the agents are modelled as circles instead of points. We show how to produce ideal solutions under certain conditions. This case, as well as the discrete case in Chapter 10, is significant because there are instances which have no solution (ideal or otherwise).

Chapter 10 considers a discrete form of MVPP (DMVPP). In this type of instance, the environment and paths within it are composed of cells. This leads to some significant differences from the continuous setting of Part I. Shortest paths are no longer guaranteed to be unique. In some instances, some shortest paths admit schedules and others do not. Some instances only have schedules for longer paths and some instances do not have solutions at all.

For those instances which admit ideal solutions, we show how to find them using the following steps. First, the region of cells containing all shortest paths between the start and target cells can be found in $O(\|\mu\|)$ time (for genus zero polygons) or $O\left(\min(\Lambda, \|\mu\|^2) + I(\|R\|)\right)$ time (for polygons with holes), where $\|\mu\|$ is the length of the shortest paths, Λ is the number of cells making up the polygon, $\|R\|$ is the size of the region and $I(N)$ is the time required to initialise a set structure and add N elements to it. Once the regions (R_1, R_2) for the agents are found, an ideal solution (if one exists) can be computed in the time required to perform $O(\|R_1\| \cdot \|R_2\|)$ visibility tests.

A summary of the results of this thesis can be found in Chapter 11.

A summary of notation for this thesis is given in Appendix A.

1.2 Publications arising from this research

- Material in Chapter 3 and Chapter 9 appeared in "Optimal paths for mutually visible agents" — Joel Fenwick and V. Estivill-Castro in "Algorithms and Computation, 16th International Symposium ISAAC", Lecture notes in Computer Science 3827, Springer pp 869-881 Editors X. Deng and D.-Z. Du

- Material in Chapter 10 appeared in "Mutually visible agents in a discrete environment" — Joel Fenwick and V. Estivill-Castro in the proceedings of the Australasian Computer Science Conference (ACSC2007), pp 141-150 Conferences in Research and Practice in Information Technology (CRPIT) Volume 62. Editor Gillian Dobbie.

Chapter 2

Related Work

Discussions involving entities moving in constrained environments appear in computational geometry; motion planning and robotics; and multi-agent systems. Researchers in multi-agent systems have been primarily concerned with protocols for negotiating to decide who should perform tasks (and what those tasks should be). The agents may have differing knowledge and ideas about the cost and value of particular actions. Auctions are a commonly used mechanism for assigning tasks to agents under such conditions (see for example Nair et al. [NTMR04], Zlot and Stentz [ZS06] and Wooldridge [Woo02]). Since our entities will be assumed not to have individual knowledge or independent decision making abilities, we focus on computational geometry and motion planning. That being said, we use the term *agents* rather than *robots* to describe our entities in order to avoid possible misconceptions about physical limitations being placed on the entities. For our purposes, an agent is an entity which can move in an environment. In Part I of this thesis, agents will be modelled as points moving in polygons.

Where we talk about vision (specific definitions will be given where required), we will assume that vision is omnidirectional and unlimited in range. This is a fairly common assumption although there is work using more limited forms (for example Lee, Shin and Chwa [LSC99] also Eppstein, Goodrich and Sitchinava [EGS07]).

We will discuss art gallery problems in Section 2.1 as an example of problems where vision is important. Section 2.2 discusses finding paths in continuous environments and graphs. In Section 2.3 we give some navigation tasks which involve both movement and vision. We also explain competitive analysis, which is a means to evaluate algorithms which operate with incomplete information. This thesis is concerned with moving multiple agents, so Section 2.4

looks at work moving multiple robots (or robots composed of more than one component). We discuss discrete environments in Section 2.5. Section 2.6 discusses two papers related to separating groups of points. Section 2.8 discusses work by Mitchell and Wynters which is closely related to the material in this thesis.

2.1 Art Gallery Problems

The original art gallery problem consists of positioning guards (with 360° vision) on the vertices of a polygon such that all points in the polygon are visible to at least one guard [O'R87]. The question which comes from this is: what is the smallest number of guards which is sufficient to guard all simple polygons with n vertices? O'Rourke [O'R87] cites Chvátal[Chv75] as proving that $\lfloor \frac{n}{3} \rfloor$ stationary guards is the maximum ever required for a simple n-vertex polygon. A shorter proof of Chvátal's result was given by Fisk[Fis78]. Moving guards are also considered and algorithms are given for guards "patrolling" along a single line segment. As long as a point is visible from some point on the segment it is considered to be guarded. Allowing guards to move reduces the number required to guard general polygons to $\lfloor \frac{n}{4} \rfloor$ provided the patrol segments can be chosen to travel in the interior of the polygon.

A watchman route is a path which sees every point in the polygon at least once. Carlsson, Jonsson and Nilsson [CJN99] gave an $O(n^6)$ algorithm for finding a minimum "floating" watchman route. A floating route is one which is not required to pass through any particular point. "Fixed" routes which must pass through a given point are more expensive.

A recent variation on art gallery type problems is the "sculpture garden problem" proposed by Eppstein, Goodrich, and Sitchinava [EGS07]. There, each guard point can see all points in an angular wedge. This vision is not blocked by walls in the environment and the goal is not to see all of the polygon. Instead, the goal is produce an expression to test if a point lies inside the polygon, based on which guards can see the point.

2.2 Path Finding

In robotics and motion planning, the focus is often on finding any path through a complex (and possibly changing environment). Two examples of this (among many) are Varadhan et al. [VKSM06] and Kant and Zucker [KZ86].

Varadhan et al. [VKSM06] give an algorithm for moving a single robot among static obstacles (that is, an environment with genus[1] greater than zero). This method is interesting because it builds a roadmap by decomposing the region into stars. It then finds "connectors" (points on the common boundary between stars) and links them to "guards" in the kernel of each star. Sampling is used to determine the star decomposition. The motion planning itself is done with a cell decomposition. Note that the authors admit that there is a weakness in the algorithm if two obstacles touch at a point, since this prevents star decomposition.

Kant and Zucker [KZ86] address the problem of moving a robot through an environment containing moving (as well as static) obstacles. They do this by decomposing the overall problem (which they term the "trajectory planning problem") into two tasks. First, a path is found which avoids static obstacles. For this task (the "path planning problem") time is not considered. The second task (the "velocity planning problem") involves calculating a sequence of velocities for moving along the chosen path. Intuitively, if a collision with a moving obstacle would occur, one robot slows down until the other passes.

That work also makes use of point-agents. The authors note that this decomposition may not produce a solution even if one exists. In particular, problems occur when obstacles are not moving independently of the robot. This does not apply to the tasks discussed in this thesis (even though we do not consider collision) because we are discussing objects whose motion is constrained relative to other objects. Specifically, they cannot move out of sight of each other.

In this thesis however, we are interested in shortest paths. A process for computing Euclidean shortest paths efficiently is described in the "Handbook of Discrete and Computational Geometry" [GO97]. First the polygon is triangulated (this triangulation will be linear in the number of polygon verticies). Next, a sequence of triangles containing the minimum path called a "sleeve" is produced. Finally the shortest path is found within the sleeve. Apart from the initial triangulation phase this process requires linear[2] time. The question is how to perform the triangulation. The handbook says that for "practical purposes" use an "$O(n \log n)$ deterministic algorithm ... or a slightly faster randomised algorithm". For "theoretical purposes" they cite a linear time algorithm by Chazelle [Cha91]. Being able to

[1] Following the convention in the computational geometry literature (for example O'Rourke [O'R87]), genus is the number of obstacles in the interior of the environment.

[2] A linear time algorithm is one with $O(n)$ time complexity. For an explanation of $O(\cdot)$ notation, see Cormen, Leiserson and Rivest [CLR90].

triangulate a simple polygon in linear time is a significant result. However, LaValle [Lav06] goes as far as to describe Chazelle's algorithm as "...so complicated that it is doubtful whether anyone will ever implement it".

In their work of 2001, Amato, Goodrich and Ramos [AGR01] give an expected linear time for triangulation using a randomised algorithm to decompose the polygon into trapezoids. They cite other papers to give a linear time transformation into triangles. Interestingly (as in the cited criticisms Chazelle's algorithm has received) while the theoretical time may be linear the authors note that their algorithm "is not likely to be of practical value."

Although the majority of this thesis deals with continuous settings, if a triangulation of the environment is known, then graph algorithms (such as Dijkstra's algorithm) provide are an alternative way to produce paths.

Fredman and Tarjan [FT87] introduced the *Fibonacci heap* data structure which requires $O(\log n)$ amortised time for delete operations and $O(1)$ amortised time for other operations. This allowed them to perform Dijkstra's algorithm in $O(V \log V + E)$ time (where E is the number of edges and V is the number of vertices).

In 1990 Ahuja, Mehlhorn et al. [AMOT90] proposed new data structures to perform Dijkstra's algorithm in time $O(E + V\sqrt{\log C})$ where C is an upper bound on the weight of edges.

Barbehenn [Bar98] presents an alternative view of shortest paths. Instead of vertices being points in the environment, they represent whole regions with graph edges linking vertices for adjoining regions. Weights are associated with the vertices rather than the edges. That is, the agent is charged for the regions it travels through rather than the specific path. In this case, performing Dijkstra's algorithm with a binary heap requires $O(E+V \log V)$ time. Barbehenn argues that although this matches the bound using a Fibonacci heap, it is an improvement because binary heaps are easier to implement. It should be noted however, that Fredman and Tarjan's bound is for edge-weighted (not vertex-weighted) graphs.

The importance of Dijkstra's algorithm can be summed by by Thorup's 1999 paper [Tho99] which states that "Since 1959, all theoretical developments in SSSP [single source shortest paths] for general directed and undirected graphs have been based on Dijkstras algorithm, visiting the vertices in order of increasing distance from s.". They then go on to propose an alternative algorithm with linear performance for positive integer weights.

Graph exploration is also used to find minimal paths in the discrete setting in Chapter 10. In that case, the graphs are constructed so that any paths found will be minimal, so breadth first search suffices for the purpose of finding minimal paths.

In Chapter 9 we consider agents represented by circles. This imposes extra limitations on paths used by such agents. Polishchuck and Mitchell [PM07] discuss "thick" paths (that is, paths which a unit disk could move along). They give an algorithm for finding optimal non-intersecting[3] thick paths for k agents. They cite another work[4] which says that the general case (allowing start and target points to be positioned arbitrarily) is NP-hard. This is interesting because it applies even to polygons with no obstacles. This complexity is avoided in their work by requiring all path endpoints to be on or "close to" the boundary of the polygon. This is an important difference to the assumptions of this thesis. We do not place restrictions on the positions of our endpoints. Since we are not dealing with the complexity of actually computing the shortest paths, this does not impose such a burden on us.

There are many other variations on this theme. For example restricting angles of rotation or constructing cost functions piecemeal.

Backer and Kirkpatrick [BK07] discuss finding paths with bounded curvature through a polygonal environment. However, that work focuses on finding a path if one exists at all and notes from a reference that finding minimum length paths is NP-hard. It should be noted that this is not in the context of genus zero polygons.

Cheng et al. [CNVW07], and Daescu et al. [DMN+06], and others discuss subdividing environments and then calculating low cost paths where different regions have different weights or even different distance functions for each region. Daescu et al. give an algorithm which approximates a path with k links with a path which is close in length and has at most $2k-1$ links. Our results are not directly applicable to such settings because their shortest paths are not guaranteed to have properties that we depend on. For example, shortest paths may cross an unobstructed line segment multiple times.

[3] Polishchuck and Mitchell used the word "crossing" but the definition of crossing used in this thesis is different.

[4] "Geometric Wire Routing". Tech-report #332 (1998) by Oliver Bastert and Sandor P. Fekete from University of Cologne. The university website says it "has not been published"

2.3 Navigation Tasks and Competitive Analysis

Single or multi-agent navigation tasks can take many forms. Here we highlight some which are of particular interest.

- Target finding: An agent starts at a designated position and must move to a target somewhere in the environment. The location of the target is unknown to the agent.
- Exploration: In a continuous setting this means to move so as to see all of the environment. Often, this is for the purpose of building a map. In a discrete setting, exploration can mean visiting every location in the environment.
- Map validation: Planning a path so that every point in the (known) environment can be seen from somewhere on the path. This is to confirm the environment matches the map the agent has.
- Evasion: Moving agents to find a moving target somewhere in the environment.

We will discuss some examples of offline and online algorithms followed by competitive analysis.

2.3.1 Offline Algorithms

An offline algorithm is one where all relevant information about the problem is available to the algorithm at the beginning. For example sorting algorithms, where the data to be sorted fits into main memory.

Icking and Klein [IK92] described the "two guards problem". The boundary of a polygon is broken into two chains between two distinguished points on the boundary. In this case, the points are designated s and g. The goal here is for agents to walk from s along the boundary chains (one agent per chain) to g such that the agents are always visible to each other. Agents are allowed to backtrack, but the goal of the "general walk" problem is to minimise the overall distance travelled by both agents.

The existence of a solution can be tested in $O(n \log n)$ time (where n is the number of vertices in the polygon) and a solution can be produced in $O(n \log n)$ time plus time linear in the size of the solution. Icking and Klein also discuss variations such as not allowing backtracking ("straight walks") and having the agents start at s and g and moving to the opposite endpoint ("counter walks").

Tseng, Heffernan and Lee [THL98] answer a question raised by Icking and Klein. They determine where start and target points can be placed in order to admit a straight walk.

While both the two guards problem and MVPPs involve walking polygonal paths, MVPPs differ in a number of ways. First, some of the complexity in two guards comes from non-convex shapes which produce deadlocks. Such forms do not occur in the paths which we use. Secondly, in MVPPs, start and target points for agents can be separated in space rather than always being the same. Finally, we do not permit backtracking along paths, whereas the two guards problem, does permit it for certain types of walks.

2.3.2 Online Algorithms

An online algorithm is one where decisions must be made in the absence of complete information about the problem instance (see for example "Handbook of Discrete and Computational Geometry" [GO97]). As decisions are made, more information is revealed about the instance. For example, as an explorer moves, their view of their environment changes and new features may become apparent. This information would not become available until the agent actually moved.

Target Finding

Target-finding problems involve positioning an agent at a start location and requiring the agent to travel to a target point. In the online case this must be done without *a priori* knowledge of the map. One example of this type of problem is given by Cohen [Coh96]. A group of "cartographer" robots worked to enable a single "navigator" robot to travel to a target. The cartographers moved randomly and had no ability to sense their absolute position. They did however, have the ability to sense the target. The solution given was to have the cartographers stop moving when they could see either the target or another robot which was a known distance from the target. Given enough cartographers, they will form a directed graph leading to the target. The navigator then follows this graph to the target. Streets (discussed on page 31) are another example of a target finding problem.

Evasion

Efrat et al. [EGHP+00] discuss the "evasion" problem. A group of moving guards must find a moving target in a polygonal environment. The only assumption made about the movement

of the target is that it is continuous. The solution given in that paper is to form the guards into a chain which is anchored to border of the polygon on either end. Each guard can see its neighbours, so the target cannot slip between guards.

Lee, Shin and Chwa [LSC99] discuss a version where the map is known but the adversary's movement function is not known. In this particular paper, there is a single searcher and a single target. The polygonal environment has a "door" thorough which the pursuer enters and through which the target could escape (or be forced, depending on the problem type). Other cases considered here are "buildings" which are environments consisting of simple polygons joined by "corridors". They show necessary conditions for a room to be searchable — searchable means that there is a path which will spot the target regardless of its movement function (subject to it being continuous of course). They also discuss restricted vision rather than the usual omnidirectional vision sense. In that case, vision consists of two "rays emanating from his [the searcher's] position, where the direction of the rays can be varied continuously with bounded angular rotation speed".

Isler, Kannan and Khanna [IKK04] consider the evasion problem on a graph where the agents (hunters and rabbits) are visible to each other only when they are at adjacent nodes. They give randomised strategies for two searchers.

Pellier and Fiorino [PF05] propose algorithms for handling polygonal environments with genus greater than zero. Agents are assigned one of a number of roles. Explorers are used to investigate unknown areas of the environment. Guards are held stationary to divide the environment into disjoint components. Idle agents are waiting for a new task. Stuck agents are unable to move on (without risking the target slipping past them) until other agents are available to help.

Murrieta-Cid et al. [MCTS+07] consider a variant with one pursuer and one evader. Here the goal is for the pursuer to keep the evader in sight. In that work the pursuer has bounded sensor range and is required to keep the evader within that range (that is between upper and lower bounds). Conversely the goal of the evader is to move out of sight. They provide necessary and sufficient conditions for the evader to achieve this.

Gerkey, Thrun and Gordon [GTG06] introduce a pursuer with a restricted field of view. They state that calculating the minimum number of these searchers is NP-Hard. They also discuss converting the continuous problem into a discrete one using cell decomposition. Note that these cells are not uniform.

SLAM

In some cases, a robot may not have direct knowledge of its own position. In these cases the exploration task can still be performed using techniques such as "simultaneous localisation and mapping" (SLAM). Estimates of the robot's postion are derived from the robot's sensors and the part of the map constructed so far. Recent work on SLAM includes that of Howard [How06] and Mourikis and Roumeliotis [MR06]. Howard [How06] says that (at that time) not much work had been done on SLAM using multiple robots. If the initial poses[5] of all the robots are known then an existing particle filter method "can be readily generalised to solve multi-robot SLAM". This is because if the start position and orientation of all the robots is known, then their sightings can be converted into a common coordinate system. The problem is more difficult if the conversion between systems is not known even when the robots can "hear" each other's sightings. Howard assumes sighting information is sent over a "reliable" wireless link. In this case when each pair of robots finally meet for the first time their current relative poses are determined and from then on, each robot can incorporate subsequent sightings from the other into its map. To deal with sightings before the meeting a virtual robot is created which moves backwards along the other robot's path replaying its sightings in reverse order. This assumes that the robots will encounter each other at some point. Arranging for robots to meet if they start in an unknown environment and out of sight of each other is termed the *robot rendezvous* problem by Roy and Dudek [RD01].

Mourikis and Roumeliotis [MR06] analyse the level of accuracy that can actually be achieved under Cooperative SLAM (C-SLAM). That is, SLAM using multiple robots.

2.3.3 Competitive Analysis

Since information revealed while an online algorithm is working may show earlier decisions to be suboptimal, online algorithms often can only produce approximations (of varying degrees of accuracy) to solutions produced by offline algorithms for the same problem.

Competitive analysis is a means to evaluate online algorithms by comparing them with offline algorithms. "In competitive analysis, the performance of an online algorithm is compared against an all-powerful adversary on a worst-case input." [KP00] The adversary modifies the information about the instance not already known to the online algorithm in order to ensure it performs badly relative to the offline algorithm. This means that the most inconvenient

[5] A robot's pose includes both its position and its orientation.

input is always the most likely. The ratio between the online and offline algorithms is termed the *competitive ratio* [KP00]. Algorithms where the ratio for all input instances is less than a constant c are said to be c-competitive.

Research in this area is ongoing. Koutsoupias and Padadimitriou [KP00] suggest that the assumptions of competitive analysis are too strict, that it is hard to differentiate between good and bad algorithms with the same worst case and the optimising for this worst case produces "unnatural" algorithms. They propose a variation ("comparative analysis") where some knowledge of the probability of different inputs is permitted to the online algorithm. Peserico [Pes04] lists some variations on what the adversary is permitted to do as well as disproving a conjecture (the lazy adversary conjecture) about how hard the adversary needs to work in order to ensure poor performance.

Areas in which competitive analysis is applied include paging (memory management) [KP00], and resource scheduling [DH04]. These first two are also addressed by the more general abstract "k-server problem" [Alb03]. Use in agent systems includes work on auctions, for example market clearing [BSZ06] where the algorithm must match buyers and sellers without knowing if better deals will be available later.

Competitive Algorithms in Navigation

Now we focus on navigation and motion planning problems. Bose and Morin [BM04] consider the problem of routing (path finding) in a graph representing a triangulation of a polygon and give a competitive algorithm. Bose, Brodnik, et al. [BBC+02] show that generalising the graph to a convex decomposition (where the routing algorithm has no memory) does not work for deterministic algorithms (although there is a randomised algorithm which works).

Blum and Chalasani [BC00] consider a variation where a point-agent is placed in an environment containing axis-aligned rectangular obstacles. In their work, the agent knows the position of both s and t (the start and target points) but not the locations of any obstacles. This work is different from the other cases discussed here in that vision is not used; the robot must collide with an obstacle in order to detect it. Also the aim is not only to reduce the distance travelled in any particular trip but to find better routes over repeated trips.

Hoffmann, Icking, Klein et al. [HIKK01] give a 26.5-competitive algorithm for exploring a simple polygon. In that case, exploration includes returning to the start point. Their

online algorithm operates by walking on circular paths to find "cuts" (similar to "extensions" defined later in this thesis) for known but not explored vertices.

Schuierer [Sch99] gives algorithms for searching more general simple polygons using trees embedded in the plane.

Another example of a target-finding problem operates in *streets* [Kle91]. Consider a simple polygon (P) with two distinguished points (s, g) on its boundary and two boundary chains L (from s to g moving anticlockwise) and R (from s to g moving clockwise). If every point on L is visible to some point on R (and vice versa), then P, s and g are said to form a street. Klein [Kle91] gives an online algorithm for finding the target g. The algorithm is $(1 + \frac{3}{4}\pi)$-competitive.

Generalised streets or G-streets, introduced by Datta and Icking [DI94] only require that points on the boundary of the polygon are visible to a point on some horizontal chord joining the two boundary chains. The set of G-streets contains all streets as well. Datta and Icking report a competitive ratio of 9 in the L_1 metric[6].

Datta, Hipke and Schuierer [DHS95] introduce an additional two varieties of streets. *HV-streets* generalise G-streets to allow boundary points to be visible from either a horizontal or a vertical chord between the boundary chains. Their second variety: *θ-generalised-streets*, like G-streets, requires that all boundary points be visible from some chord joining the bounding chains. However, such chords must be at angle θ from the horizontal. That paper gives competitive ratios of 14.5 and 19.97 respectively (these ratios are under the L_1 distance metric.

One of the ways in which streets differ from our setting is that the sources and targets for streets must lie on the polygon boundary, whereas in our case endpoints can be anywhere in the polygon.

Icking et al. [IKKL02] discuss the searching problem including a survey of results for streets. This work also considers the task of exploring (that is visiting every cell) of an unknown discrete environment. It gives a lower bound of 2 on the competitive ratio for doing this. We discuss discrete environments in Chapter 10.

A related task is referred to as the covering problem by Gabriely and Rimon [GR03]. In that case, all points in an area must be covered by a tool (for example sweeping a floor).

[6] Distances under the L_1 metric are the sum of the absolute values of the component distances. For example $|\Delta x| + |\Delta y|$ in two dimensions [Lav06].

The environment is discretised by dividing the region into cells and discarding any partially occupied cells (the size of the cells is derived from the size of the tool). The complexity of the task is measured in terms of the total number of cells.

2.4 Multiple Robots

In this section, we discuss work involving multiple robots (or complex robots with multiple components). We also discuss a major problem affecting such work, that is the dimension of the configuration space.

Pham and Parhi [PP03] discuss experiments (both in simulation and with real robots) using multiple robots to find targets. The robots manoeuvre in a cluttered environment and have to avoid each other. The focus of the work is on using Neural nets and Petri nets for navigation so competitive issues are not discussed.

Howard, Parker and Sukhatme [HPS06] discuss work as part of DARPA's Software for Distributed Robotics Program. Here the task is to cover a previously unexplored environment with sensors. There are two classes of robots. The first maps the environment. Once this is done, a second class of robots is moved to cover the environment using this map. Of particular interest here is the fact that the robots which provide the sensor coverage lack the sensors necessary to navigate and have to be guided into position by explorer robots. In order to do this the sensor robots form a chain following the robot in front of them with the leader being an "explorer" robot. This is an example of a connected vision constraint.

The RoboCup competition and symposium has addressed issues in robotics including team robotics yearly since 1997 [Rob]. The 2006 symposium [LSST06] had 143 paper submissions.

The multiforaging task has robots (single or multiple) collecting objects scattered around the environment and delivering the objects to designated locations. The robots do not know the map or the locations of the objects ahead of time. Balch [Bal99] discusses this task and analyses strategies for using multiple robots for the task.

Rekleitis, Dudek and Milios [RDM01] use two robots to build a map. One robot explores the environment while keeping the other stationary robot in sight. Holding one of the robots stationary means that it can be used as a reference point to compensate for odometry errors in the other robot. The robots switched roles from time to time. That work also employs

another interesting technique. Rather than relying on sensors to detect obstacles directly, they do so indirectly when one robot lost sight of the other.

Peng and Akella [PA05] consider the problem of moving a group of robots along specified paths. Like this thesis, the aim is to produce a set of movements such that the robots travel along these paths subject to some constraints with no backtracking permitted. In this case the constraints are that the robots must not collide with each other and the velocities of the robots are continuous. In our case, the constraints are based on vision and there is a set of permitted velocities. For the most part, this means we permit only one non-zero velocity, with acceleration considered to be instantaneous. Further, Peng and Akella [PA05] optimise for travel time rather than path length.

The problem of moving a group of robots under constraints is extremely similar to that of moving a single robot composed of multiple parts. For example moving a robot arm requires calculating positions for each joint in order to position the "hand". This must be done so that the different components which make up the arm remain the correct distance from each other and do not attempt to move through each other.

For example, two agents with motion constrained by vision could be modelled as a single agent with two parts joined by a telescopic rod. In this thesis, however, we ignore the problem of different agents occupying the same space.

Saha, Roughgarden et al. [SRLSA06] consider the planning problem of moving a robot arm on a tour of specified points such as occurs in welding, painting or inspection. As well as seeking paths which do not cause the different components of the robot to collide, they also consider the issue of the collection of joint angles to position the end at a particular point not being unique.

The *configuration space* or *C-space* is the set of legal configurations for entities in a problem instance. Alternatively, it can be viewed as the space of legal tuple values for the parameters of the system. For example, for moving a point through a maze, the C-space would be all x, y pairs not occupied by walls. For a robot arm, the C-space could be the space of joint values which do not violate the physical constraints for the robot. That is, the C-space is not necessarily equivalent to the location of entities since there may be multiple tuples which lead to a given location.

One issue with using C-spaces is their size (in particular their dimension). Each additional parameter in the system can potentially add a dimension to the C-space[7]. For example, the work by Gerkey, Thrun and Gordon [GTG06] on the evasion problem has some discussion about extending to multiple pursuers but it suffers from an explosion in the size of the configuration space.

One technique for alleviating this is to use "Probabilistic Road Maps"(PRM) [SL02] (early work on PRM includes Overmars [Ove92] and Kavarki and Latombe[KL94]). Under this method, the configuration space is sampled at random and each configuration is tested to determine if the configuration it represents is non-colliding. Such safe points are termed *landmarks*. The roadmap (graph) is then constructed by connecting landmarks with non-colliding paths (where possible). Since collision testing is expensive, the authors attempt to improve on the method by delaying testing particular links until a path between goals is found to run through them. They also use a two tree approach, searching the graph from both source and destination. As Varadhan et al. [VKSM06] note, PRM brings another slant on the issue of completeness[8]. PRM is termed *probabilistically* complete because if a solution exists it will be found eventually.

Other recent work in PRM includes that of Hsu, Latombe and Kurniawati [HLK06] who analyse why PRM is successful and consider different sampling strategies. One of the arguments given in favour of PRM is that modelling the free space, that is the part of the environment in which the robot can move, could be very complex. However, provided the environment has certain visibility properties (for a definition of *expansive* sets see the paper) then PRM will do well. The cases where PRM has trouble are narrow passages but the authors claim that in practice these formations are unlikely to occur by accident. This is similar to this thesis in that solutions are comparatively easily attained provided the visibility is not pathological.

Aronov et al. [AdBvdS+99] take another approach to reducing the dimension of the configuration space. If the total number of degrees of freedom over the group of robots (and hence the dimension) is d in an environment with n obstacles, they reduce the dimension to $d-1$ for a pair of robots and $d-2$ for a group of three robots (although there is an additional assumption for the three-robot case). This is accomplished by having the robots remain in

[7] An in-depth discussion of configuration spaces can be found in LaValle [Lav06].

[8] Completeness refers to whether an algorithm will find a solution if it exists.

physical contact with each other during the journey. The exception is when one or more of the robots is at their source or destination point. Note that this work permits backtracking. The paper provides a proof that even under this restriction a solution will still be produced if one exists for the more general problem. The paper does not provide proofs for higher numbers of robots but suggests that larger numbers of robots could be grouped together and moved as a unit.

Aronov et al. also address a more restricted case of robots with "bounded reach" in "low-density environments". Bounded-reach robots are ones where the largest minimal circle/ball required to enclose the robot in any physical configuration is bounded (and not too big). Intuitively, the density of an environment describes how many relatively large objects can be in a relatively small area. Under such conditions, they give an $O(n \log n)$ algorithm for computing a path.

The two main approaches to planning motions for a group are centralised and decentralised [AdBvdS+99]. Decentralised approaches solve smaller subproblems, for example the movement of one robot, and then attempt to combine them into a solution for the whole system. This means the subproblems to be solved have lower dimension but the overall algorithm is not guaranteed to be *complete*. That is, even if a solution exists, it is not guaranteed that a decentralised algorithm will find it. Lumelsky and Harinarayan [LH97] say that most of the literature has been focussed on centralised approaches.

However, decentralised approaches may be the only viable ones in situations where there is little or no communication with (or knowledge of), the other robots in the system. This is especially the case in multi-agent systems type problems. Lumelsky and Harinarayan [LH97] discuss such a scenario using a "cocktail party" as a motivating example. Their question is how to move through an environment with stationary objects and other moving robots given no communication and no central planning. No particular knowledge is assumed about the movement patterns of any of the other robots, although some general assumptions about things like maximum speed are made.

Schwarz and Sharir [SS83] discuss the "piano movers problem". The goal is to move a set of objects, which may or may not be connected to each other from one configuration to another without the objects colliding. In that paper, they consider circular objects. One reason for this is that non-circular objects can be replaced with bounding discs to try for a quick solution before using more complex algorithms. Schwarz and Sharir allow circle arcs

as well as line segments in the boundary of free space. Our treatment of circular objects, however, in Chapter 9, is restricted to polygonal boundaries. A solution to the piano movers problem is found using the following steps. First partition the free space into "non-critical" regions using "critical curves". A non-critical region consists of configurations where none of the objects are touching each other or the walls of the environment. A critical curve consists of those configurations where at least one of the objects is touching either a wall or another object. Next, produce a graph linking the non-critical regions. Finally, search the graph for a path between start and goal configurations.

2.5 Discrete Environments

A discrete environment is one where entities can be in one of a discrete set of positions. This is most commonly represented by some type of regular array of cells. Discrete environments are used in computer graphics as well as modelling, simulation and motion planning applications, where an exact analytic representation is not required or is infeasible. The "tile world" [Woo02] environment in agent systems research is a four-connected discrete grid environment consisting of obstacles, holes and tiles which can be moved to fill holes. As another example, some computer games take their maps as grids with a limited number of ground-based entities per grid cell (for example "StarCraft" [Ent]). These applications perform path-finding in environments with potentially large numbers of moving entities. An increase in the sophistication of this path finding can be seen by contrasting "StarCraft" with the more recent "Generals" [Gam]. For example, in the latter, stationary entities will be moved out of the way if they block a moving entity's path. Carpin et al. [CLW+07] use the commercial UnrealEngine2 game engine to produce a simulation environment for multiple robots in urban search and rescue.

Even in physical problems, discretising the environment or range of movement parameters is sometimes an alternative to modelling the system exactly. For example the work by Gabriely and Rimon [GR03] discussed previously. Tan and Clapworthy [TC03] discuss remotely controlled robots — specifically "Internet based teleoperation". In order to reduce communication costs, a virtual environment is used to model the environment in which a physical robot is operating. The state of the model is updated with information sent from the real environment. They note the problems of implementing tasks such as collision detec-

tion in virtual environments. The real world requires "adequate sensors" whereas a virtual environment requires "a considerable amount of computation". They deal with problems caused by differences between the orientation of the cell boundaries and orientation of moving objects by using spherical cells. A manifestation of this type of complexity appears in Chapter 10, in that case, lines of sight do not always align with cell boundaries.

Tan and Clapworthy [TC03] reduce the dimension of the configuration space by decomposing the path-planning task for a manipulator on a robot arm into two tasks. The first task ignores the orientation of the manipulator and plans a path to move it to the required position. The second task involves calculating the orientation of the manipulator as moved along the path. In this case this changes a six dimensional problem into two three dimensional ones.

2.6 Point Separability

In Chapter 8, our methods for producing schedules in the non-collinear case rely on finding two chords S and T. These chords need to be chosen so that all start points lie (non-strictly) on one side of S and all target points lie (non-strictly) on the other side. The same applies to T and we require that S and T do not cut. Further, S and T must pass through at least one point in their respective sets.

This is a question about the separability of two point sets in a polygon. Work in this area includes that of Demaine et al. [DEH+04] and Bose et al. [BDH+04].

Demaine et al. give an $O(n \log n)$ algorithm for finding a shortest path between two boundary points (a geodesic) which separates two sets (if such a path exists). Such a path is not guaranteed to be a single line though (and does not provide the other properties we need). On the other hand, Bose et al. produce *ham sandwich geodesics* which, instead of separating the sets divide both sets in half.

2.7 Models of Computation

When measuring the complexity of algorithms the model of computation should be considered. That is, what operations can our algorithms perform (at what cost) and how is data to be represented.

- RAM (Random Access Machine) - abstracts the operations and addressing modes of a modern computer. Each instruction is charged unit cost. (See for example Mehlhorn[Meh84])

- real RAM - Removes precision as a consideration. Each memory location is assumed to hold a single real number. Arithmetic, comparisons as well as trig functions, exponentials and logarithms are all assumed to have unit cost. [PS90].

In Chapter 10 we discuss a discrete setting so we will assume the RAM model. The rest of the thesis assumes the *real RAM* model is in use. Following Preparata and Shamos [PS90], we do not consider the cost to input (or output) the real numbers used in our algorithms.

2.8 Covisible Points

The closest work to this thesis was done by Mitchell and Wynters. The paper [MW90a] and an associated technical report [MW90b] discuss the following problem. Move two point-agents such that they remain visible to each other[9] and the paths the agents travel meet some minimality criterion. They consider three minimality criteria: minimising the total path length (MIN-SUM); minimising the longer to the two paths (MIN-MAX). They also give a bounded approximation for minimum total travel time.

The work can be divided into two subproblems. First, finding a pair of paths which meet the minimality criterion. Second, computing a schedule of movement for agents along these paths.

Since Mitchell and Wynters prove that a schedule exists for all "locally optimal paths" (that is, paths which cannot cannot be made shorter without taking a different route through the obstacles), the ability to produce a schedule is not dependant on the choice of paths.

The shapes formed by the paths[10] are classified into three types and schedules are produced using a combination of three types of motion.

The shapes are:

- Hourglass: The two paths do not intersect.

[9] Mitchell and Wynters describe this state as *covisible*. In this thesis we use the term *mutually visible*.

[10] Because the two paths can intersect, the "defining polygon" (formed by the line segment joining the starts points, the paths and the segment joining the target points) might not be a simple polygon. Mitchell and Wynters note this fact.

- Funnelglass: The two paths intersect at a vertex of the visibility graph (that is, a vertex of the environment). This leads to the paths (with the sightlines) forming two funnels with their apexes joined by a (possibly trivial) polyline. A funnel is defined by three points a, b and c and consists of the region bounded by the line segment from a to b and convex polylines from chains from c to a and b [GO97].

- Bowtie: The two paths intersect at a point but that point is not a vertex of the visibility graph.

That is, the classification is based on the interactions between the paths, not the interactions between the sightlines. In this thesis, the shapes are classified into two groups based on whether or not S and T cross. This covers all the shapes which could arise in a genus zero environment[11].

The motions are:

- Single step motion: One of the agents moves along one path segment while the other remains still.

- Double cross motion: The two agents move along their segments to reach an intersection point. This is necessary to deal with the case in bowties where path segments intersect.

- Telescoping see-saw motion: Two agents move such that the line segment joining them always passes through a fixed point. This is employed in cases where S and T cross (termed a *twisted* hourglass by Mitchell and Wynters). The agents are moved so that the sightline joining them always passes through the intersection point of S and T.

None of these motion types include backtracking (that is, an agents visiting a point more than once). In our case we use the first two types of motion but the third type will not work directly in our setting. The difficulty is that since we do not vary the speeds of the agents arbitrarily, we cannot guarantee that the sightline between them always passes through a given point. In fact, under our restrictions, these twisted hourglasses turn out to be significantly harder than other types of instances. Instead we employ a slightly different motion where the agents move in turns and the sightline between them must pass through a fixed line segment. That is, the intersection point between the sightline and the segment

[11] Mitchell and Wynters also allow hourglasses with paths which loop around an obstacle. This cannot occur in genus zero for shortest Euclidean paths.

moves from one end of the segment to the other as the agents move. If the segment is very short a large number of individual motions will be needed to complete the schedule. We will refer to this type of motion as *crab motion*. The overlap with this thesis lies in the second task (that is, producing the schedule).

We became aware of this work after independently developing the content in this thesis. One important aspect in which our work advances the state of knowledge left by Mitchell and Wynters lies in the movement functions for the agents. In our case, movement speeds are restricted to one of two values. Even under this stronger restriction schedules still exist for minimal Euclidean paths except for a set of pathological cases which we identify. In such cases we provide an efficient means to produce approximations to the minimal paths which do admit schedules. In Chapter 8, we relax the restrictions to allow two fixed speed ratios (in addition to one of the agents being held stationary) and show that this is sufficient to make all our instances solvable. In their open problems, Mitchell and Wynters include questions about larger numbers of agents under different visibility constraints. We address these questions in genus zero environments under both visibility constraints.

Some other differences between our treatments are:

- Mitchell and Wynters [MW90b] do not analyse the complexity of producing schedules. It is apparent however that they believe the cost of producing the schedule to be linear or at least lower than the cost of finding the paths. We give complexities for producing schedules for two agents.

- Mitchell and Wynters [MW90b] omit some details about schedule production. For example, twisted hourglasses appear to require the relative speeds of agents to be adjusted each time an agent reaches a vertex on its path.

- For instances not involving twisted hourglasses, both our solution and that proposed by Mitchell and Wynters amount to walking a triangulation. However, in our case the triangulation does not need to be explicitly computed by the schedule algorithm. It could be argued that in the general case, a triangulation would be needed anyway in order to calculate shortest Euclidean paths. In special cases though (such as the start and target points being on the boundary of the polygon) there may be alternative ways to compute shortest paths.

Part I

Point-Agents in Continuous Environments

Chapter 3

The Continuous Case for Two Point-Agents

3.1 Basic Terms and Notation

Points in the plane will be written as p_i. The line segment between points p_i and p_j will be written $\overline{p_i\, p_j}$. The *line through a segment* is the infinite line which includes all points on the segment. The Euclidean length of a segment $\overline{L}$ is the distance between its endpoints and is denoted $|\overline{L}|$. Throughout this thesis we will describe sets or sequences of points as being collinear. This means, that all points in the group lie on the same infinite line. For example "p, q, r are collinear" means that r lies on the line through the segment $\overline{p\, q}$.

The following definition of *polygon* is different from the standard definition in that the angle at a vertex could be equal to 180°. That is, three successive vertices could be collinear.

Definition 1 *A* simple polygon *is defined by a disjoint set of* $\mathcal{V}$ *vertices* $\{v_0, \ldots, v_{\mathcal{V}-1}\}$ *(with* $\mathcal{V} \geq 3$*) such that line segments joining successive vertices do not intersect any other segments except the preceding and succeeding ones and then only at their endpoints. That is,* $\forall i, j \in \{0, \ldots, \mathcal{V}-1\}, i \neq j$, $\overline{v_i\, v_{(i+1) \bmod \mathcal{V}}} \cap \overline{v_j\, v_{(j+1) \bmod \mathcal{V}}} = \emptyset$. *The exception is if* $i = (j+1) \bmod \mathcal{V}$ *or* $j = (i+1) \bmod \mathcal{V}$*, in which case the intersection is* $\{v_i\}$ *or* $\{v_j\}$ *respectively. The border of the polygon is considered to be part of the polygon.*

We will use $\mathcal{V}$ throughout the thesis to denote the number of vertices in the polygon.

All the polygons in this thesis will be simple, so we omit the qualifier from now on. That is, the polygon never properly self-crosses. In most chapters in this thesis the polygons will

be *genus zero*. That is, there are no "holes" in the interior of the polygon. Formally, the exterior of the polygon has only one connected component.

Definition 2 *A* polyline *is a sequence of line segments such that the end of one segment is the same point as the start of the next segment. No segment in the polyline may intersect the exterior of the polygon. The successor of a vertex u in a polyline is the endpoint of the segment which u starts.*

The Euclidean length of a polyline π is the sum of the lengths of the segments of π and is denoted $|\pi|$. Apart from the material in Chapter 10, path lengths will always be measured with the Euclidean norm [Rud74]. The number of segments in a polyline π is denoted $\|\pi\|$.

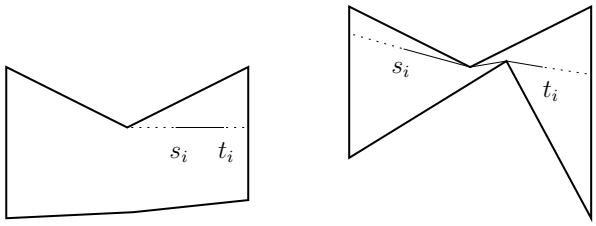
In this thesis, all the paths between points will be polylines. Note that Definition 2 is not as strict as the definition of polygon border, it does not, for example, exclude polylines which self-intersect. However, since none of the paths produced in this thesis will self-intersect, we will not discuss this point further. Paths which are the shortest path between their endpoints are of particular interest. Note that all the vertices on such paths (with the exception perhaps of their endpoints) will be vertices of the polygon [Ede87]. We denote as Π_i, the shortest path between two points s_i and t_i. If a path between s_i and t_i is not known to be the shortest possible path, then we label it π_i.

As with the definition of polygon, the existence of a vertex on a path does not mean that the path has changed direction at that point. In fact, we use the convention that the following types of vertices on paths must be preserved:

- Vertices on paths which are also vertices of the polygon. That is, if a path passes through a polygon vertex, then the path must also have a vertex at that point.[1]
- Vertices marking the start and end of the path.
- Vertices marking points of significance on other paths, for example where the path passes through the start point of another path.

The last type will probably not be present in the original input, since paths will be calculated without reference to other paths. Further, in this chapter, we do not need to consider such

[1] This contrasts with an assumption in some work in computation geometry that no three points under consideration can be collinear.



(a) (b)

Figure 3.1: Π_i extended to form ${}^{ext}\Pi_i$. (a) shows the single segment case, while (b) shows the multi-segment case.

vertices explicitly. In later chapters with more agents, detecting these situations (and adding vertices for them) becomes an issue.

Definition 3 *Two points in the interior of a polygon are* mutually visible *if the line segment joining them does not intersect the exterior of the polygon. In that case the segment is termed the* sightline *for those points.*

Throughout the thesis we refer to obstacles.

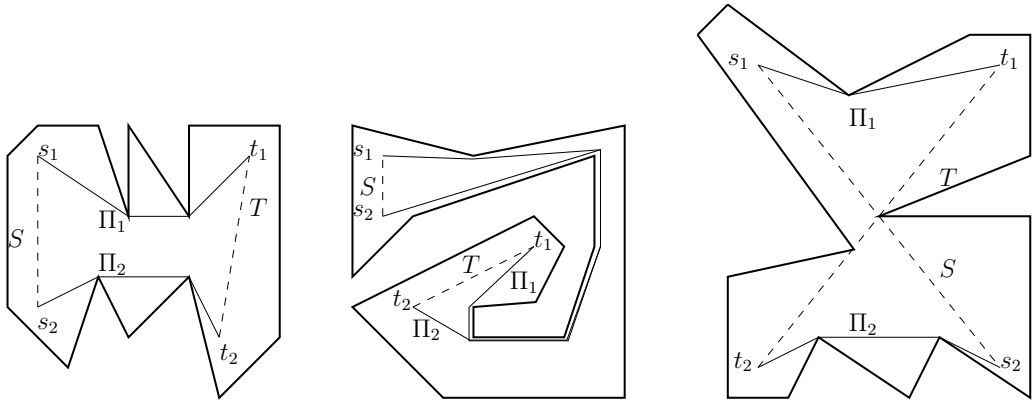
Definition 4 *An* obstacle *is a connected subset of the polygon boundary which blocks vision or movement along the line segment joining two points.*

So, for example, a vertex of the polygon and its associated edges could form an obstacle. Alternatively, a sequence of vertices and edges could form a single obstacle. Whether a piece of boundary is an obstacle depends on the points being considered.

Definition 5 *If Π_i consists of at least one segment, then the sequence ${}^{ext}\Pi_i$ of segments is produced by extending the initial and final segments of Π_i beyond s_i and t_i until they intersect the polygon boundary. Note that if both endpoints lie on the polygon boundary, then ${}^{ext}\Pi_i{=}\Pi_i$. Further, for a single segment the definition is equivalent to a chord. Figure 3.1 shows some examples. If Π_i consists of a single point, then ${}^{ext}\Pi_i$ is not defined.*

Definition 6 *A* point-agent *is a point which moves along a path according to a sequence of moves called a* schedule.

While an agent is technically distinct from the position it currently occupies, for brevity, when an agent is used in a context which requires a fixed point, the agent's current position should be substituted.



(a) (b) (c)

Figure 3.2: Some examples of MVPPs. Thin lines denote shortest paths, thick lines are polygon boundary and dashed lines are the sightlines. In the case of 3.2(b) paths are drawn with an offset from the boundary for illustration purposes only.

Definition 7 *A* Mutually Visible Path Problem (MVPP) *consists of a simple, genus zero polygon P with four distinguished points s_1, s_2 (the* start points*), t_1, t_2 (the* target points*) in P. A solution to an MVPP consists of paths from s_1 to t_1, and from s_2 to t_2 as well as a schedule to move agents a_1, a_2 along the paths. At all times during the schedule, the agents must be mutually visible. Each agent can either remain stationary or move at a constant speed, which is common to both agents. Note that the agents are allowed to occupy the same point.*

From this definition, an MVPP is not well formed if the start points (and target points) are not mutually visible. From this point on we will assume these points are mutually visible.

Definition 8 *The* start sightline *S is the segment $\overline{s_1\, s_2}$ while the* target sightline *T is $\overline{t_1\, t_2}$.*

The following assumption simplifies reasoning significantly.

Definition 9 *A* clipped *MVPP is an instance where the path Π_i for each agent only intersects S at s_i and T at t_i.*

We will assume that instances where this is not the case, (if the path travels along S or T) will be recast so that s and t are the last/first points of the path on S and T. This assumption will be referred to as the clipping assumption.

This is a reasonable assumption because once a solution to the new form is reached, the agents can be trivially moved to their targets with no concerns about visibility.

Definition 10 *An* ideal solution *for an MVPP consists of* Π_1, Π_2 *and a schedule which uses them. Such a solution is ideal because its distance cost is no higher than if the agents had moved to their targets ignoring the visibility constraint.*

Where possible, we wish to produce ideal solutions. If no ideal solutions exist, then we want to approximate an ideal solution. Some examples of MVPPs, with shortest paths marked, are shown in Figure 3.2.

Note that in this work, the cost of a schedule is the total length of the paths ($|\pi_1|+|\pi_2|$), not the number of pauses in the schedule or the number of segments in the paths ($\|\pi_1\|+\|\pi_2\|$).

The first question to ask is: "Do all MVPPs have a solution?" If so, the next question is: "Which MVPPs, if not all of them, admit ideal solutions?" In this chapter we show that the answer to the first question is yes. For the second question, it turns out that there are instances, like the one shown in Figure 3.2(c), that do not admit ideal solutions. All such instances do, however, admit solutions with costs which approximate $|\Pi_1| + |\Pi_2|$.

To distinguish between these types we need to know whether or not two paths cross. This is not equivalent to asking if the paths intersect. For example, if the intersection is a path endpoint, then the paths do not properly cross. We also exclude cases where the paths meet and then diverge again without actually crossing or where the paths run together for a while.

Definition 11 *We will write* $circle(p, \delta)$ *for a circle (that is, only the boundary) with centre at* p *and radius equal to* δ*. We will write* $disk(p, \delta)$ *for a closed disk with centre at* p *and radius equal to* δ*.*

Definition 12 *Two paths* π_i *and* π_j*, have an* isolated point intersection (ip-intersection) *at* p*, if* $\exists d > 0$ *such that* $\forall\, 0 < \delta < d$*,* $\pi_i \cap \pi_j \cap disk(p, \delta) = \{p\}$*.*

An ip-intersection is not the same thing as a singleton intersection, since two paths could have multiple ip-intersections. Admittedly, this would not happen if both paths were the shortest paths between their endpoints.

Definition 13 *Two paths* π_i *and* π_j*, containing at least one segment each,* cross *at point* p *if* $\exists d > 0$ *so that* $\forall\, 0 < \delta < d$*:*

- π_i *and* π_j *have an ip-intersection at* p*.*

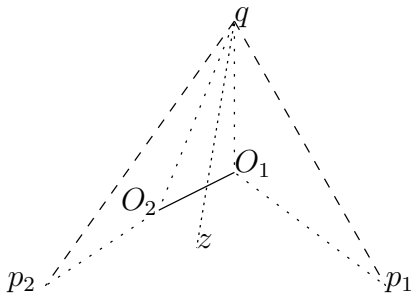


Figure 3.3: Points p_1, $p_2 \in \pi$ are visible to q. Point $z \in \Pi_1$ is not visible to q. Dashed lines denote sightlines, dotted lines denote lines which may be broken by obstacles and solid lines represent polygon boundary.

and

- *The intersection of circle(p, δ) with the two paths (that is, circle$(p, \delta) \cap (\pi_1 \cup \pi_2)$) contains exactly four points, $p_1, p_2 \in \pi_1$ and $q_1, q_2 \in \pi_2$ which appear in the order p_1, q_1, p_2, q_2 around the circle.*

Consider three points p, p_1 and p_2 such that p sees both the other two points. It is not hard to see that the shortest path from p_1 to p_2 is convex and does not cross outside the triangle formed by the points. However, since shortest paths will not always be usable, we need to identify a larger class of paths which still have these properties.

Definition 14 *Let p, p_1, p_2 be points in a genus zero simple polygon with $\overline{p\,p_1}$ and $\overline{p\,p_2}$ being sightlines. Let π be a path between p_1 and p_2. The path π is* p-restricted *if:*

1. *When its endpoints are joined, π forms either a straight line or a simple convex polygon.*
2. *No point on π lies outside the region bounded by $\overline{p_1\,p}$, $\overline{p_2\,p}$ and the shortest path between p_1 and p_2.*

We will refer to the following lemma as the triangle lemma.

Lemma 1 *Consider a path π between p_1 and p_2 in a genus zero polygon such that both points are visible to a point q. If π is either the shortest path or is q-restricted, then every point on π is also visible to q.*

Proof: Since the shortest path is trivially q-restricted we will only prove the restricted case.

Refer to Figure 3.3. We proceed by contradiction. Suppose $\exists z \in \pi$ not equal to p_1 or p_2 such that $\overline{q\,z}$ is not a sightline. There must be an obstacle blocking q's vision and it must have entered the wedge between p_1 and p_2 since the other two faces of the wedge are sightlines.

Choose a segment $\overline{O_1\,O_2}$ of the polygon boundary which cuts $\overline{q\,z}$ (that is, O_1 lies on one side of $\overline{q\,z}$ and O_2 lies on the other side). Since this is a genus zero polygon and π is contained entirely within the wedge, π must intersect the line segments from q to O_1 and O_2. (Note that we are not asserting that these segments are sightlines; they could be cut by obstacles). Further, any convex path (and p-restricted paths are convex) must travel on or above the segments $\overline{p_1\,O_1}$ and $\overline{p_2\,O_2}$ but z lies below them. So we have our contradiction since no convex π can reach z.

□

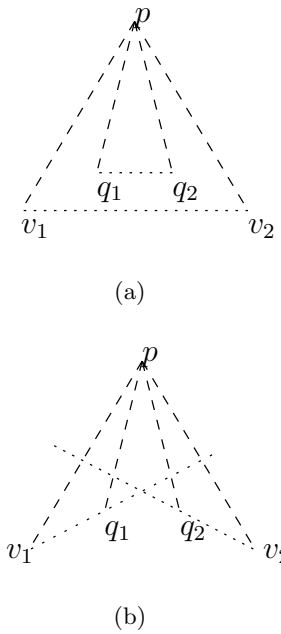


Figure 3.4: An illustration of the symbols used in Lemma 2

Now we show that smaller parts of a p-restricted path are also p-restricted.

Lemma 2 *If a path from v_1 to v_2 is p-restricted then for any q_1, q_2 on the path between v_1 and v_2 inclusive, the section of the path between q_1 and q_2 is also p-restricted.*

Proof: Refer to Figure 3.4. Denote the path from q_1 to q_2 as π_{part}. A connected subset of a convex chain must also be convex, so the only way π_{part} could fail to be p-restricted is if it cut one of $\overline{p\,q_1}$, $\overline{p\,q_2}$ or the shortest path between q_1 and q_2. Note that when the instance is oriented as shown in the figure, all turns made by π must be clockwise. Since convex paths only turn in one direction [Wei], an anti-clockwise turn would render the path non-convex. Further, at any point z, no clockwise turn can be further than the line between z and the end of the path. To do so would require an anti-clockwise left turn or a self-intersection to reach the target and give a contradiction.

Let Π_{part} denote the shortest path from q_1 to q_2. Consider the region bounded by $\overline{p\,q_1}$, $\overline{p\,q_2}$ and Π_{part}. If π_{part} travels below Π_{part}, then it must reconnect with it later in order to reach q_2. This means one of three things must occur:

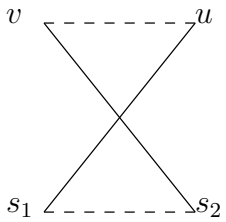


Figure 3.5: An instance where the segments to successor vertices cross.

- Path π_{part} makes an anti-clockwise turn. This is a contradiction.
- Path π_{part} self-intersects. This is a contradiction.
- Path Π_{part} turns clockwise to intersect π_{part}. This means that Π_{part} and π_{part} enclose a region containing no obstacles with π_{part} taking a shorter route. This contradicts Π_{part} being the shortest path.

Now consider Figure 3.4(b). To reach q_1 while making only clockwise turns, Π must travel along or above the line through $\overline{v_1\, q_1}$. This means that π_{part} must travel on or below the line after q_1. If the path is travelled in reverse, then we can see that π_{part} must also stay below the line through $\overline{v_2\, q_2}$. This means that π_{part} cannot even reach the segments $\overline{p\, q_1}$ and $\overline{p\, q_2}$, let alone cross them. Note that no vertex of Π_{part} can be above those lines since that would require an anti-clockwise turn from q_1 to navigate around the obstacle.

So π_{part} is convex and remains within the prescribed region so it is x-restricted.

□

We can now state the main results for this chapter. Section 3.2 establishes the existence of ideal solutions if S and T do not cross. We give a linear time algorithm for producing these solutions if Π_1 and Π_2 are supplied. Section 3.3 examines the instances where S and T do cross and shows when ideal solutions exist and when an approximation is required. We give algorithms to produce solutions in both cases. We also give a linear time algorithm to distinguish between the cases.

3.2 The Case When S and T do Not Cross

For instances where S and T do not cross we will prove the existence of ideal solutions. This means the paths we will be using in this section are Π_1 and Π_2. We will establish the

existence of a schedule by induction, so now we consider possibilities for the first move by each agent. We will show that it is always safe to move one of the agents. Once this is done, a new instance can be started using the new agent positions as start points.

Let u and v denote the successor vertices of s_1 and s_2 respectively ($u \in \Pi_1$, $v \in \Pi_2$). We will deal with the case where one of the vertices is the last on its path in Theorem 1. Let S' be the longest line segment through S, which does not intersect the exterior of the polygon. The possibilities for the relative positions of u and v are as follows:

- The current and successor vertices are equal. That is, $s_1 = s_2$ and $u = v$. Here both agents can be moved to their successor vertices immediately.
- Either u or v is visible to s_1 and s_2 and lies on S'.

  In such a situation, move the agent along the sightline to its next vertex. Since the movement occurs along an existing sightline there are no visibility issues.
- Vertices u and v lie on the same side S'. There are two subcases here:
  - The segments $\overline{s_1\, u}$ and $\overline{s_2\, v}$ cross. These instances are examined in Lemma 3.
  - The segments $\overline{s_1\, u}$ and $\overline{s_2\, v}$ do not cross. Lemma 4 discusses these instances.
- Vertices u and v lie on opposite sides of S'. These instances are discussed in Lemma 5.

Lemma 3 *Consider an MVPP with non-crossing sightlines with* $\|\Pi_1\| > 1$ *and* $\|\Pi_2\| > 1$ *with successor vertices denoted* u *and* v *respectively. If* $\overline{s_1\, u}$ *and* $\overline{s_2\, v}$ *cross, then a schedule exists to move both agents to their successor vertices.*

Proof: See Figure 3.5. The points s_1, s_2 and the intersection point of $\overline{s_1\, u}$ and $\overline{s_2\, v}$ form a triangle which is bounded on all sides by sightlines and so must be clear of obstacles. Move the agents to the intersection point. After both agents have arrived at the intersection point, the agents lie at the apex of another clear triangle so they can be moved to their successor vertices.

□

Lemma 4 *Consider an MVPP with non-crossing sightlines with* $\|\Pi_1\| > 1$ *and* $\|\Pi_2\| > 1$ *and successor vertices denoted* u *and* v *respectively. If* u *and* v *are on the same side of* S' *and segments* $\overline{s_1\, u}$ *and* $\overline{s_2\, v}$ *do not cross, then both of the following hold:*

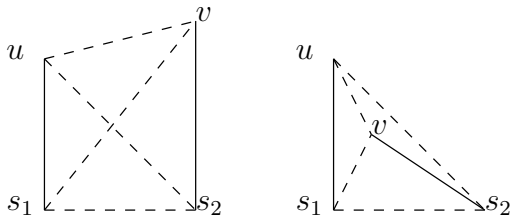


Figure 3.6: Instances where u and v lie on the same side of S'. Dashed lines are for illustration purposes and do not necessarily indicate sightlines.

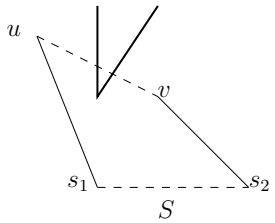


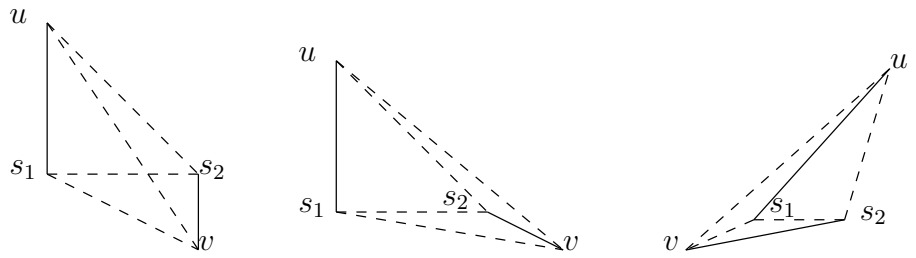
Figure 3.7: The paths for an MVPP cannot be arranged like this without one of them failing to be a shortest path.

1. *The nodes see their successors (s_1 sees u and s_2 sees v).*

2. *The quadrilateral defined by s_1, u, v, s_2 is free of obstacles and hence, $\overline{s_1\, v}$ or $\overline{s_2\, u}$ must be a sightline.*

Proof: See Figure 3.6. Point 1 being false would imply that there was an obstacle between a vertex on Π_i and its successor. This cannot be since it implies that there must be additional vertices between the vertex and its successor to get the path around the obstacle. This is a contradiction by the definition of *successor* (Definition 2).

To prove Point 2 we need to show that the edges of the quadrilateral are not cut and there are no obstacles in the interior of the quadrilateral. Segments $\overline{s_1\, u}$ and $\overline{s_2\, v}$ are unbroken from Point 1 and S is a sightline so the only edge left is $\overline{u\, v}$. We proceed by contradiction. Suppose $\overline{u\, v}$ is cut by an obstacle. See Figure 3.7. Since the polygon is genus zero the region bounded by S, Π_1, T, Π_2 cannot include any obstacles, that is, the paths must move around any obstacles. So, both target points must lie on one side of the obstacle or the other. That means one of the paths must backtrack to move around the obstacle. But a shorter path must



(a) (b) (c)

Figure 3.8: Instances where u and v lie on opposite sides of S'. Dashed lines are for illustration purposes and do not necessarily indicate sightlines

exist to get around the obstacle, and this contradicts the claim that Π_1 and Π_2 are shortest paths. So by contradiction $\overline{u\,v}$ must also be a sightline. This means we have a quadrilateral free from obstacles, so any internal diagonals ($\overline{s_1\,v}$ or $\overline{s_2\,u}$) must be sightlines as well.
□

Lemma 5 *Consider an MVPP where S and T do not cross but where u and v (the successors of s_1 and s_2) lie on opposite sides of S'. At least one of $\overline{u\,s_2}$ or $\overline{v\,s_1}$ is a sightline and hence one of the agents can be moved.*

Proof: The segment $\overline{u\,v}$ could cross S' either by crossing S itself (shown in Figure 3.8(a)) or by crossing on one side or the other (shown in Figure 3.8(b) and Figure 3.8(c)). The next figure will illustrate $\overline{u\,v}$ crossing S but the same reasoning applies to the other cases as well.

We proceed by contradiction. Refer to Figure 3.9(a) (page 54). Without loss of generality, assume that the instance is oriented as shown in the figure. In order for segments $\overline{s_1\,u}$ $\overline{s_2\,v}$ to both be broken without S also being broken, there must be at least one polygon vertex which has crossed the segments from each side. These are shown as thick lines in the figure. Pick a vertex from each obstacle such that they are visible to s_1 and to each other and join them with a chord (denoted $\overline{O}$) in the figure. At least one pair must exist because the only thing which could interrupt $\overline{O}$ is another obstacle so that one could be chosen instead. Neither path can intersect that chord since then the paths would not be shortest. For example, suppose Π_1 crosses $\overline{O}$. It cannot be the shortest path since there must be a shorter path to $\overline{O}$ than going up to u and then back down to $\overline{O}$. We also know that neither path can intersect S' once it leaves it, since then the path would not be the shortest. So we know that T must cross the chord $\overline{O}$ and S' but not S itself.

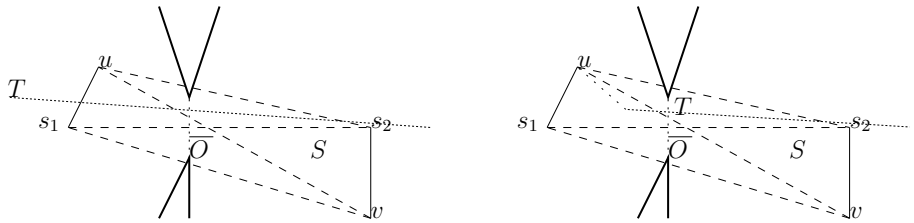


Figure 3.9: A hypothetical instance where $\overline{u\,v}$ crosses S.

Without loss of generality assume that T cuts O on the u side of S. There are two possibilities here: first that T intersects $\overline{s_1\,u}$ (See Figure 3.9(a)). In that case though, a shorter path to t_1 exists by moving along T. This contradicts Π_1 being the shortest path. Second, T does not extend far enough to intersect $\overline{s_1\,u}$. See Figure 3.9(b). The path Π_1 must connect with t_1 eventually, but even if Π_1 is able to take the direct route from u to t_1 (shown as sparse dots in the figure), it will still not be shortest overall. There must be a shorter path from s_1 to t_1.

We have our contradiction, so either $\overline{u\,s_2}$ or $\overline{v\,s_1}$ must be a sightline. This means one of the agents can see every point on the other agent's segment so the second agent can be moved to its successor vertex.

□

Lemma 3 proves that there is a schedule for paths which cross midsegment. This only works if a_1 and a_2 arrive at the beginning of the relevant segments in the same step. This raises the question: Could visibility problems arise if one agent moves through a crossing segment before the other agent arrives? This lemma shows that an agent will not move through a crossing segment if doing so would cause visibility problems. We will write the successor of vertex v as v'. The points b_1, b_2 are the vertices immediately before the crossing point.

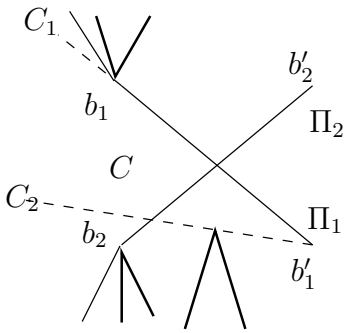


Figure 3.10: An MVPP with crossing sightlines. The algorithm will not attempt to move into the intersection until both agents are at b_1 and b_2.

Lemma 6 *Let b_1 and b_2 be the endpoints of crossing segments in shortest paths in an MVPP. That is, the vertices encountered first when travelling from the start points. Let p be a vertex in Π_2 before b_2 such that a_1 could move from b_1 to b_1' while a_2 either remains at p or moves to p's successor p'. The process described in this section would not attempt to advance a_1 in such a situation.*

Proof: Refer to Figure 3.10. First consider the case where a_2 remains stationary. In this case the points p, b_1, b_1' would form a triangle with interior free of obstacles. However, a contradiction arises when we consider possible positions for p.

Extend the line segment $\overline{b_1\, b_1'}$ past b_1 until it intersects the exterior of the polygon. This new segment is denoted C_1.

Rotate C_1 on point b_1' towards b_2 until it encounters an obstacle or it passes through b_2 (this segment is denoted C_2). In order to retain visibility of both b_1 and b_1', p would need to lie between C_1 and C_2. This region is labelled C in Figure 3.10, but any position for p in the interior of C implies that Π_2 is not a shortest path since Π_2 would travel downwards to reach b_2 then upwards again to reach b_2. There are no obstacles to force this behaviour so Π_2 can not be the shortest.

What if p was on C_2? If $\overline{b_2\, b_2'}$ intersects C_2 at a single point, then there must be a shorter path from p to b_2'. We know this because Π_2 would make an upwards turn without an obstacle to force it to do so. If $\overline{b_2\, b_2'}$ and C_2 have a non-singleton intersection, then Π_1 and Π_2 meet at $b_2' = b_1'$ and hence this lemma does not apply.

Thus, holding a_2 stationary at p results only in a contradiction. We do not need to consider moving the two agents simultaneously, since the only time that happens in our algorithm is when moving through a cross together and that cannot occur more than once. □

Later in the thesis we show how to determine if and where two shortest paths intersect (in Chapter 4).

Theorem 1 *An MVPP where the sightlines do not cross has an ideal solution.*

Proof: First we show that, if the start and target points for one of the agents are the same, then a schedule exists. Without loss of generality assume $s_1{=}t_1$. Since S and T are sightlines then a_1 must be able to see s_2 and t_2, that is, both endpoints of Π_2. So by the triangle lemma a_2 can be moved directly (along its path) to its target without losing visibility. If both agents start at their targets, then the schedule is empty.

Now we consider the cases where $s_i \neq t_i$. If S lies along one of the next segments, then that agent should be advanced. This check should be performed before the others. If $\overline{s_1\, u}$ and $\overline{s_2\, v}$ do cross, then Lemma 3 describes the steps to take. So now we show what to do when $\overline{s_1\, u}$ and $\overline{s_2\, v}$ do not cross. By Lemma 4 and Lemma 5 one of the start points must be able to see the successor vertex of the other start point. So, the other agent may be safely moved to its next vertex. For example if s_1 can see v, then move a_2 to v. Determining which agent to advance can be done by examining where the lines through $\overline{s_1\, u}$ and $\overline{s_2\, v}$ intersect. If the line through the segment of Π_2 intersects the segment of Π_1, then move a_2 and vice versa. Otherwise, move either of the agents. This test can be performed in constant time.

What if the paths Π_1 and Π_2 cross? Lemma 3 deals with the case when a_1 and a_2 arrive at the intersecting segments in the same step. We need to be sure that there are no visibility problems if one agent moves through the crossing segments before the other agent reaches them (this also applies to paths crossing at a vertex). Lemma 6 proved that the agents will not move through the crossing segments early if doing so would create a visibility problem. That lemma only considers the case where the intersection does not occur at a vertex. If the paths cross at a vertex, then when one of the agents (say a_1) moves to the intersection vertex, the sightline between the agents must run along Π_2 to the intersection point. If the sightline does not lie along Π_2, then the sightline is shorter than Π_2 and we have a contradiction. So the next moves will result in a_2 moving to the intersection point.

Now we have a new MVPP instance with $\|\Pi_1\|+\|\Pi_2\|$ reduced by one. Since the polylines are finite, repeating this process will eventually reach their targets and hence, we have a schedule and an ideal solution.

□

Note that if the shortest paths are supplied, then a schedule can be produced in $O(\|\Pi_1\|+\|\Pi_2\|)$ time. That is, linear in the overall number of segments in the paths[2].

3.3 The Case Where Sightlines S and T Cross

This case differs from those discussed in the previous section in that instances of this type are not guaranteed to have ideal solutions. We can however show that solutions exist where the total path length is as close to $|\Pi_1|+|\Pi_2|$ as desired. We will decompose instances into three subtasks, all of which must be completed for a path to be usable in a schedule.

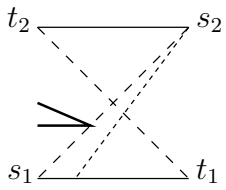


Figure 3.11: A case where point s_2 has non-trivial visibility of s_1 but s_1 has trivial visibility of s_2.

- The Starting task — Moving both agents off S.
- The Finishing task — Moving agents a small distance to their targets on T.
- The Connecting (or middle) task — Moving the agents from their positions after the starting task to positions where the finishing task can be performed.

We will show that the Finishing task is symmetrical to the Starting task. Note, we are not saying that the movements to start for a given instance are symmetrical to the movements to finish for that instance. As we will show, certain conditions in either the Starting or Finishing tasks can prevent the shortest paths from admitting a solution. Section 3.3.1 shows how to choose different paths in such cases. That is, new paths are chosen and the instance is restarted. Once this is done, the algorithms given will operate on the new path. For this

[2]We are producing schedules for particular paths. That is, the paths Π_1 and Π_2 form part of the input. So this cost is function of the size of the input.

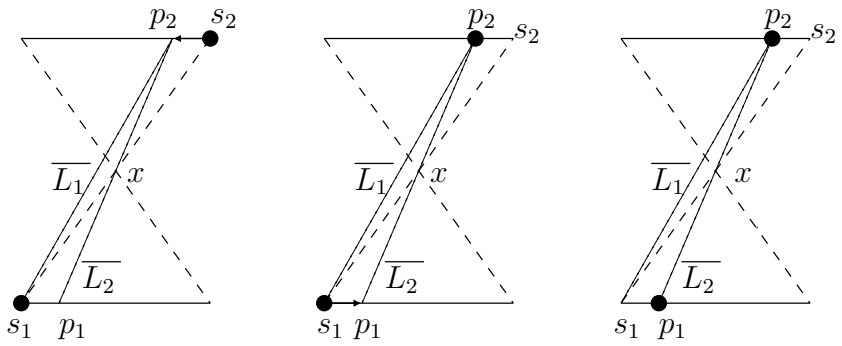


Figure 3.12: Steps to start a schedule.

reason, in this section we will use π_i to denote the paths from s_1 to t_1 and from s_2 to t_2 (even if they turn out to be shortest paths).

We need to consider cases where one agent can see the other, but any forward movement (no matter how small) on the part of the other agent will break the sightline. To describe this situation, we define trivial visibility.

Definition 15 *Consider a point p in the polygon and an agent a_i at another point q which is visible to p. If $\exists d > 0$ such that a_i can move distance d along its path while remaining visible to p, then p is said to have* non-trivial visibility *of q. If no such d exists, either because a_i is already at the end of its path or because the sightline from p to q would be broken, then p is said to have* trivial visibility *of q.*

Note that non-trivial visibility is not guaranteed to be symmetric. See Figure 3.11 (page 57) for an example.

Now we are ready to discuss the Starting task for instances where at least one of the start points has non-trivial visibility of the other start point. Where it is necessary to emphasise the distinction between line segments which comprise a path and segments between arbitrary points we will call the former *path segments*.

Lemma 7 *If in an MVPP where S and T cross, one of the start points has non-trivial visibility of the other, then both agents can be moved from their start points.*

Proof: Refer to Figure 3.12. Without loss of generality assume that s_1 has non-trivial visibility of s_2.

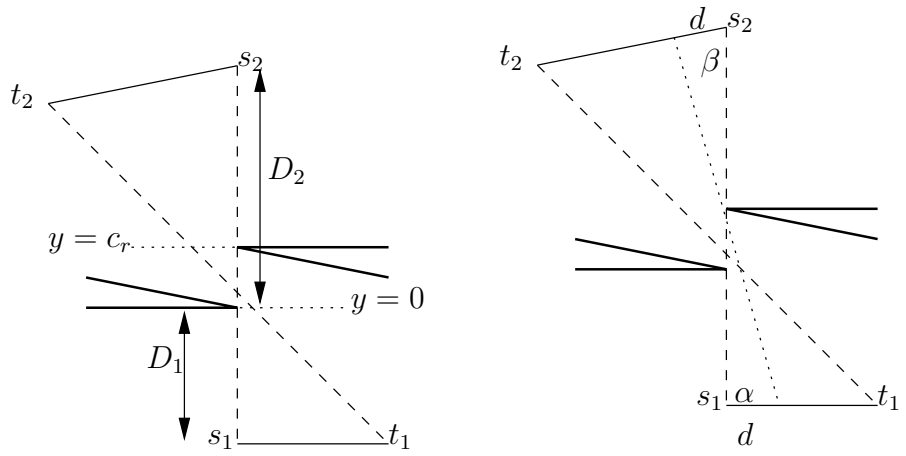


Figure 3.13: An MVPP where start points have zero measure visibility of each other. The highest left-touching obstacle is at the line $y = 0$, the lowest right-touching obstacle is at the line $y = c_r$.

Let the furthest point along π_2 which s_1 can see be denoted by p_2 and let $\overline{L_1}$ be the sightline $\overline{s_1\, p_2}$. Let the line segment from p_2 through x to a point on π_1 be denoted $\overline{L_2}$ (its endpoint on π_1 is denoted by p_1). If p_1 is not on the same path segment as s_1, then set p_1 as the end point of the first segment of path Π_1 and adjust $\overline{L_2}$ appropriately. Since the whole of $\overline{L_2}$ lies inside the region $<S, T, \pi_1, \pi_2>$, the segment $\overline{L_2}$ must be a sightline.

So both agents can be moved from their start positions.

□

Now we consider instances where both start points have trivial visibility of each other. It is under these conditions that the shortest paths may not admit a schedule.

For this section we assume that the instance is oriented as in Figure 3.13 and uses Cartesian coordinates. We will denote the intersection between S and T as x. The sightline S lies along the y-axis. The highest touch by an obstacle on S from the left is at the origin and the lowest touch from the right is at the line $y = c_r$. The fact that we only have trivial visibility guarantees that these contacts exist. The agents will travel for distance d on segments which make angles α and β with S for a_1, a_2 respectively. These angles must be greater than 0° and less than 180°. The distances from s_1 and s_2 to the the line $y = 0$ are denoted D_1 and D_2 respectively. Note that while D_2 is always positive, D_1 is positive only if a_1 is below the line $y = 0$, otherwise the sign of D_1 switches to negative. This negative condition never

applies in the two agent case but may apply in the n-agent case in Chapter 5. An equivalent condition is not required for D_2 since a start point below the line $y = 0$ would imply that a left obstacle cuts T.

Lemma 8 *In an MVPP where S and T cross and where s_1, s_2 have trivial visibility of each other, there exists $d > 0$, such that the agents can move from their start positions if and only if:*

1. $0 < (D_2 \sin\alpha - D_1 \sin\beta)/(\sin\beta + \sin\alpha) < c_r$

   or

2. $\sin(\beta - \alpha)/(\sin\beta + \sin\alpha) \geq 0$ *and* $(D_2 \sin\alpha - D_1 \sin\beta)/(\sin\beta + \sin\alpha) = 0$

   or

3. $\sin(\beta - \alpha)/(\sin\beta + \sin\alpha) \leq 0$ *and* $(D_2 \sin\alpha - D_1 \sin\beta)/(\sin\beta + \sin\alpha) = c_r$.

Proof: Refer to Figure 3.13. It is important to note that all distances less than our chosen d must also preserve visibility.

If the sightline between the agents intersects S outside the band between the line $y = 0$ and the line $y = c_r$, then the agents will not be mutually visible. In such a case the sightline must cut an obstacle. So first, we will compute the position of this intersection as a function of d (the distance travelled).

After moving distance d the positions of the agents are given by

$$p_1 = (d\sin\alpha, -D_1 + d\cos\alpha) \text{ and}$$

$$p_2 = (-d\sin\beta, D_2 - d\cos\beta).$$

The vector equation of the segment between p_1 and p_2 is

$$p(\lambda) = p_1 + \lambda(p_2 - p_1),$$

with $0 \leq \lambda \leq 1$. The individual coordinates of $p(\lambda)$ will be labelled $p(\lambda)[x]$ and $p(\lambda)[y]$ respectively. Calculating the intersection of S with $p(\lambda)[x] = 0$ gives

$$d\sin\alpha + \lambda(-d\sin\beta - d\sin\alpha) = 0.$$

This means the intersection occurs at

$$\lambda = \frac{\sin\alpha}{(\sin\beta + \sin\alpha)}.$$

Now we substitute this value into $p(\lambda)[y]$ to give the y coordinate of the intersection.

$$\begin{aligned} p(\lambda)[y] &= -D_1 + d\cos\alpha + \frac{\sin\alpha}{(\sin\beta+\sin\alpha)}(D_2 - d\cos\beta + D_1 - d\cos\alpha) \\ &= \frac{1}{\sin\beta+\sin\alpha}((-D_1 + d\cos\alpha)(\sin\beta+\sin\alpha) \\ &\qquad + \sin\alpha(D_2 - d\cos\beta + D_1 - d\cos\alpha)) \\ &= \frac{1}{\sin\beta+\sin\alpha}(-D_1\sin\beta + d(\cos\alpha\sin\beta - \sin\alpha\cos\beta) + D_2\sin\alpha) \\ &= \frac{1}{\sin\beta+\sin\alpha}(d\sin(\beta-\alpha) + D_2\sin\alpha - D_1\sin\beta) \end{aligned}$$

Taking the limit of $p(\lambda)[y]$ as d approaches 0 gives the intersection at

$$y = (D_2\sin\alpha - D_1\sin\beta)/(\sin\beta+\sin\alpha).$$

The sign of derivative of $p(\lambda)[y]$ with respect to d tells us which direction the intersection point moves as d increases. The derivative is given by

$$\frac{\partial p(\lambda)[y]}{\partial d} = \sin(\beta-\alpha)/(\sin\beta+\sin\alpha).$$

To make the notation less cramped we will write z for $p(\lambda)[y]$ at some value of d and ∂z for the derivative of $p(\lambda)[y]$ with respect to d.

Now we show that conditions in the Lemma's hypothesis are sufficient for the agents to start. We wish to ensure that z remains in the safe interval $[0, c_r]$.

- Item 1: $0 < (D_2\sin\alpha - D_1\sin\beta)/(\sin\beta+\sin\alpha) < c_r$

  means that the limit of z lies in the interval and has room to move in either direction. Note that the strict inequalities provide the space to ensure some movement is possible. A suitable d for this case must be small enough that z does not move out of the region and that the sightline is not broken by obstacles that are close to but not touching S. We give a procedure for choosing d after discussing the other items.

- Item 2: $\sin(\beta-\alpha)/(\sin\beta+\sin\alpha) \geq 0$ and $(D_2\sin\alpha - D_1\sin\beta)/(\sin\beta+\sin\alpha) = 0$

  and

Item 3: $\sin(\beta - \alpha)/(\sin\beta + \sin\alpha) \leq 0$ and $(D_2 \sin\alpha - D_1 \sin\beta)/(\sin\beta + \sin\alpha) = c_r$,

mean that the limit of z lies at one end of the safe interval but that either $\partial z = 0$ so z does not move on S as d increases, or the sign of ∂z indicates that z will move into the interval rather than out of it. For example, if z lies at the origin, and ∂z is positive, then z will move upwards (into the interval) rather than downwards (out of the interval). A safe d in this case just needs to be small enough that z does not move out the other end of the interval and that the sightline does not encounter any non-touching obstacles.

The following procedure provides a suitable d. We will ensure that z does not move out of the safe interval first, then we show that vision is not broken by other obstacles. If $\partial z = 0$, then the first step can be skipped, since z will not move out of the safe interval.

Let d' be the distance z can move without leaving the interval (choose negative values for downward movement). Equating d' with $d \cdot \partial z$ (the distance z moves when the agents move d) and solving for d gives the value of d which corresponds to z moving by d'. We will use d_m to denote this value of d which is,

$$d_m = \frac{d' \sin\beta - \sin\alpha}{\sin(\beta - \alpha)}.$$

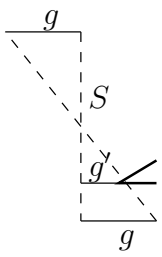


Figure 3.14: An instance where the agents have moved distance g and the sightline between them is blocked by a vertex at distance g' from S.

Now we ensure that the sightline between the agents will not cut any other obstacles. Calculate the minimum perpendicular distance from each polygon vertex to the line through S. Let g be the smallest absolute distance. Since we are assuming the instance is oriented with S along the $y-$axis, this means that g is the minimum absolute $x-$coordinate for vertices. Provided that z remains in the safe interval, the agents can move distance g without losing visibility. To see why, refer to Figure 3.14. For the sightline to be obstructed, the distance from the $y-$axis to some vertex must be smaller than the distance to the agents. The distance to the vertex is labelled g' and the distance to the agents is g. This is a contradiction since g is supposed to be minimal. The figure shows agents moving perpendicular to S but the same reasoning also applies to other angles.

We will choose d to be the minimum of d_m and g.

This establishes the sufficiency of Items 1, 2 and 3. We will now show their necessity using proof by contradiction. Moving either of the agents individually will cause visibility to be lost (since the agents have trivial visibility of each other). The only remaining option is to move them simultaneously. Suppose that there is a $d > 0$ such that agents moving this distance simultaneously do not lose visibility but that none of the cases listed in the lemma apply. This means that either the limit of z lies outside the safe interval, or that the limit lies at the end of the interval; the derivative is non-zero and its sign means z moves out of the interval. In both situtations a sufficiently small movement results in an intersection point outside the interval which means that the segment between the agents must be broken.

We have our contradiction and therefore being able to move from the start position implies one of the items in the lemma.

□

The computational complexity of this test is given in Section 3.3.2.

Corollary 9 *There exist MVPPs where S and T cross which do not admit ideal solutions.*

Proof: We need to describe an instance which violates the conditions from Lemma 8. Assuming all instances are oriented as in Figure 3.13, choosing an instance with $\sin\beta > \frac{D_2}{D_1}\sin\alpha$ gives $z < 0$. For example, such a β exists when $D_2 < D_1$ and $\alpha \neq 90°$.

□

Corollary 10 *For an MVPP where S and T cross there are angles of departure α and β which will allow the agents to start.*

Proof: Lemma 8 assures us that we can start if $0 < (D_2\sin\alpha - D_1\sin\beta)/(\sin\beta + \sin\alpha) < c_r$. If the current departure angles satisfy this constraint, then we are done. If not, then one of the angles should be decreased so they do satisfy the constraint. Decrease is necessary because increasing the angle might lead to the path cutting the polygon boundary.

If $D_2 > D_1$, then choose $\sin\alpha$ so that $\frac{D_1}{D_2}\sin\beta < \sin\alpha < \min\{1, \frac{D_1+c_r}{D_2-c_r}\sin\beta\}$. Note that the lower bound is less than 1 and that all quantities are positive. This can be rearranged to give $\sin\beta$ if required. To see the correctness of these bounds substitute them into

$$(D_2\sin\alpha - D_1\sin\beta)/(\sin\alpha + \sin\beta).$$

Substituting the lower bound $\sin\alpha = \frac{D_1}{D_2}\sin\beta$ gives

$$\left(D_2\frac{D_1}{D_2}\sin\beta - D_1\sin\beta\right) / \left(\frac{D_1}{D_2}\sin\beta + \sin\beta\right)$$

$$= (D_1 \sin\beta - D_1 \sin\beta) \,/\, \left(\frac{D_1}{D_2}\sin\beta + \sin\beta\right)$$
$$= 0.$$

Substituting the upper bound $\sin\alpha = \frac{D_1+c_r}{D_2-c_r}\sin\beta$ gives

$$\left(D_2\frac{D_1+c_r}{D_2-c_r}\sin\beta - D_1\sin\beta\right) \,/\, \left(\frac{D_1+c_r}{D_2-c_r}\sin\beta + \sin\beta\right)$$
$$= \left(\frac{D_2(D_1+c_r) - D_1(D_2-c_r)}{D_2-c_r}\sin\beta\right) \,/\, \left(\frac{(D_1+c_r)+(D_2-c_r)}{D_2-c_r}\sin\beta\right)$$
$$= (D_2(D_1+c_r) - D_1(D_2-c_r))/(D_2+D_1)$$
$$= (D_2+D_1)c_r/(D_2+D_1)$$
$$= c_r.$$

So $0 < (D_2\sin\alpha - D_1\sin\beta)/(\sin\alpha + \sin\beta) < c_r$.

If $D_2 \leq D_1$, then choose $\sin\beta$ so that $\frac{D_2-c_r}{D_1+c_r}\sin\alpha < \sin\beta < \min\{1, \frac{D_2}{D_1}\sin\alpha\}$. Note the lower bound is less than 1. Substituting these bounds also gives $0 < (D_2\sin\alpha - D_1\sin\beta)/(\sin\alpha+\sin\beta) < c_r$.

It remains to choose angles which meet the constraints since $0 < \sin < 1$ has multiple solutions. The angle should be chosen from the interval $(0°, 90°)$ since choosing a larger angle might mean the angle has increased, which may mean the polygon boundary is cut.

This gives us our start angles.

□

There is also a symmetric problem with finishing a schedule. In order to apply the above lemmas to the finishing task we need to show that there is a schedule which will move the agents into a position where they can finish.

Lemma 11 *Consider an MVPP where S and T cross and where paths π_1 and π_2 are both x-restricted. If the agents have both moved off S, then there is a schedule which ends with both agents on their final path segments and the same distance from their targets. This distance will be small enough that the finishing task is possible.*

Proof: Assume the instance is oriented as in Figure 3.15. Consider a line segment joining a_1 to some $p_2 \in \pi_2$. Choose p_2 to be as far as possible along π_2 without $\overline{a_1\,p_2}$ being blocked by an obstacle. Denote this segment $\overline{L}$ and the intersection between $\overline{L}$ and T as x_1. In some instances it may be that $x_1 = x$, that is, there is no gap between x and x_1. In this case, swap the agent labels. This case must work because x cannot be touched from both sides.

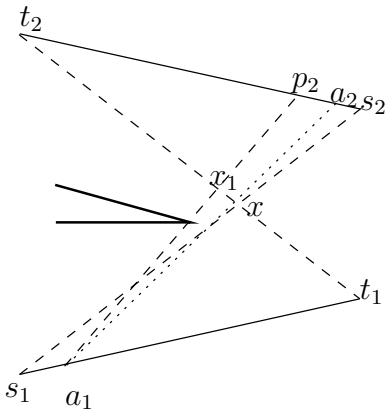


Figure 3.15: An MVPP with crossing sightlines where both agents have moved off S.

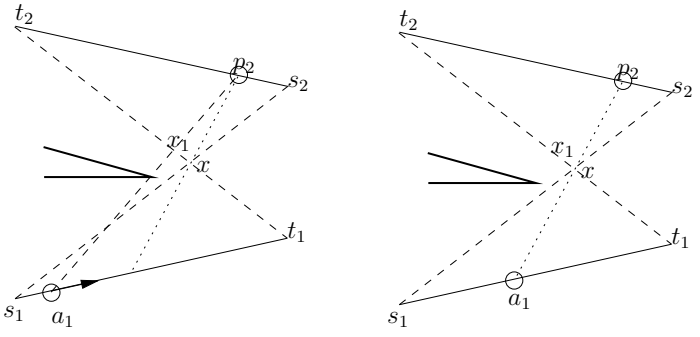


Figure 3.16: Steps in moving after a_2 has reached p_2 in Figure 3.15.

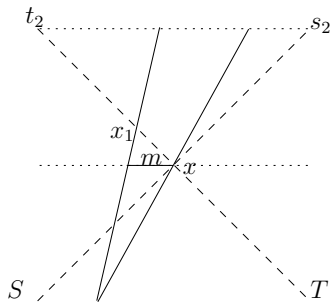


Figure 3.17: The segment marked m is the projection of the segment $\overline{x\,x_1}$ onto a line through x parallel to the segment $\overline{s_2\,t_2}$.

Move a_2 to p_2 since we already know that a_1 can see that far. At this stage we have the situation shown in Figure 3.16(a) (page 65).

The next step is to move a_1 until the sightline between a_1 and a_2 passes through x. The result is shown in Figure 3.16(b) (page 65). Visibility is preserved during this move because the section of path which a_1 moves along is x-restricted (by Lemma 2) and is therefore a_2-restricted as well, so we can apply the triangle lemma.

This process can be repeated as often as desired, sweeping $\overline{a_1\,a_2}$ between x and x_1 and swapping the labels of a_1 and a_2 as needed. Note that $\overline{a_1\,a_2}$ does not leave the region bounded by S, L, T, π_1, π_2 and so, provided that it does not cross either path it will be a sightline. We know that it will not cross either path because that would require one of the paths not to be x-restricted. Now we need to show that agents will actually travel the required distance. We will do this by showing that the amount of progress the agents make is bounded from below.

Refer to Figure 3.17. The amount of progress made by a_2 is bounded below by m. (Note that m depends on the position of a_1). Finding a lower bound on m will give a lower bound on the amount of progress made in a single step. Note that m depends on the position of a_1 and the closer a_1 is to T, the smaller m will be. However, provided a_1 is not on T, m will be greater than zero. Consider a position of the agents which puts a_2 at the desired distance from t_2 and positions a_1 to be collinear with $\{a_2, x_1\}$[3]. That is, as if a_2 had just stopped moving. The value of m which corresponds to this movement will be the smallest value so far encountered. This means that, given enough steps a_2 must reach the nominated point.

[3]That is, a_1 lies on the line through a_2 and x_1.

Now we need to move a_1 into position. If the sightline between the desired positions passes between x and x_1, then a_1 can be moved directly to its destination. If the sightline passes to one side of x, then the path section between a_1 and its desired position is x-restricted and, hence it is also a_2-restricted. So a_1 can be moved into position. A symmetric argument guarantees that a_1 will reach any desired distance as well.

□

Since a schedule can be produced which positions both agents any desired distance from their targets, a distance can be chosen which positions both agents on their final path segments. This means that Lemma 7, and Corollary 10 guarantee that the agents can finish their journeys.

If either the starting or finishing task prevents the use of shortest paths, then we want replacement paths which have combined lengths as close as desired to the shortest path lengths. We show how to calculate x-restricted paths (π_1 and π_2) approximating Π_1 and Π_2 in Section 3.3.1. These paths can be chosen so that $|\pi_1| + |\pi_2|$ is as close to $|\Pi_1| + |\Pi_2|$ as is desired.

3.3.1 Constructing Restricted Paths

Previously, we have assumed that in cases where there is no ideal solution, there is a set of x-restricted paths which will work with lengths close to the shortest paths. Now we show how to construct such paths.

First, note that none of the paths we will construct will travel outside the region bounded by $\overline{s\,x}$, $\overline{x\,t}$, Π so in order to show our paths are x-restricted we only need to show convexity. Let p_s and p_t be the endpoints of the starting and finishing tasks respectively. That is, the Starting task will move the agent from s to p_s and the Finishing task will move the agent from p_t to t. Consider the line segment $\overline{L}=\overline{p_s\,p_t}$, if Π does not cross $\overline{L}$, then $\overline{L}$ will do as our path. If not, then $\overline{L}$ should be bent around Π.

If Π does not intersect $\overline{L}$, then we are done, so assume this is not the case. Walk the vertices(v) of Π calculating the gradient of $\overline{p_s\,v}$. Choose the v with the highest gradient. The first segment of π is $\overline{p_s\,v}$. Repeat from the other end to find the last segment of π. In between these segments add all segments from Π which lie between the chosen v's.

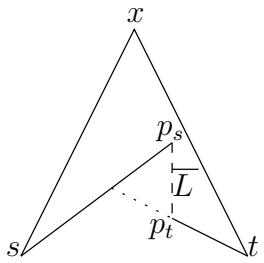


Figure 3.18: A case where start segment is too long and so the overall path is not x-restricted.

The final question is whether the π produced in this way is still convex when the start and finish segments are appended to it. Convexity could only be broken by a turn in the wrong direction where π joins to the starting and finishing segments. This can happen in two ways. The first is illustrated in Figure 3.18 and the second occurs if the starting or finishing segments move below Π. Provided the starting and finishing segments are short enough so that ${}^{ext}\overline{p_t\, t}$ does not cut $\overline{s\, p_s}$ (and vice versa) π, then π will not need to make a reflex turn when it connects to the finishing segment (Figure 3.18). The possibility that the starting or finishing segment moves below Π is excluded in the two-agent case because the departure angles are always less than or equal to those for Π, so π is x-restricted. This question will be re-examined in Chapter 5 when more agents are introduced. Given Π and the positions of p_s and p_t, π can be produced in $O(\|\Pi\|)$ time. To see that the combined length of π, the starting and finishing segments can be made as close as desired to $|\Pi|$, note the following. Either π includes at least one vertex from Π or it does not.

If π does include a vertex from Π, then π joins Π at its second vertex v (after s). That is, s, p_s and v form a triangle. As the length of the starting segment $\overline{s\, p_s}$ decreases, the combined length of sides $\overline{s\, p_s}$ and $\overline{p_s\, v}$ approaches that of $\overline{s\, v}$. Symmetric reasoning for the finishing segment gives an overall length as close as needed to $|\Pi|$.

If π does not include a vertex from Π, then s, p_s, p_t and t form a simple quadrilateral. Reducing the length of $\overline{s\, p_s}$ and $\overline{t\, p_t}$, brings the total length of the replacement path closer to $|\Pi|$.

3.3.2 Computational Complexities

The overall complexities for operations in this section are as follows ($\mathcal{V}$ denotes the number of vertices in the polygon). To test for trivial visibility requires $O(\mathcal{V})$ time. Each vertex is checked to see if it touches S or T and if not, how close it is. This distance information can be used to determine a safe d for starting — just choose d to be smaller than the distance to all non-touching vertices. Once this has been done, determining feasible start and finish

angles and distances is $O(1)$ time. That is, determination of whether a schedule exists for shortest paths can be done in $O(\mathcal{V})$ time overall. If new paths must be calculated, then this can be done in $O(\|\Pi_1\| + \|\Pi_2\|)$ time.

The cost to produce the schedule for the chosen paths (π_1, π_2) depends both on the number of segments in each path and on the size of the moves made at each step in Lemma 11. Each segment in each path must be checked in turn for intersection with the new sightline. Once this segment is found, previous segments do not need to be checked again. This gives time complexity of $O(\|\pi_1\| + \|\pi_2\|)$. Let D_1 be the distance between the end of the Starting task and the beginning of the Finishing task for a_1. Let m_1 be the value of m for the last move made by a_1. An upper bound on the number of moves N_1 for a_1 is D_1/m_1. If N_2 is calculated for a_2, then an upper bound on the number of steps N for both agents is $\max\{N_1, N_2\}$. Hence the overall time complexity to test for and produce a schedule for a given pair of paths is $O(\mathcal{V} + \|\pi_1\| + \|\pi_2\| + N)$.

In summary:

Theorem 2 *All MVPPs have either an ideal solution or one which approximates the ideal as closely as required.*

Proof: If S and T for the MVPP do not cross then we know there is an ideal solution by Theorem 1.

If S and T cross, then we know that the schedule can be started (Lemma 7 for non-trivial visibility cases and Corollary 10 for trivial visibility cases). A symmetric argument ensures that schedules can be finished. Finally, Lemma 11 ensures that agents can be moved from the end of the Starting task to the beginning of the Finishing task. Section 3.3.1 discusses how paths can be constructed to approximate the ideal if required. Thus, we have a schedule.

□

Chapter 4

MVPPs with Collinear Endpoints — the Non-Crossing Case

Now we present an extended definition of MVPP for n agents.

Definition 16 *A* Mutual Visibility Path Problem for n agents (MVPP-n) *is described by a simple polygon, a set of n point pairs $\{(s_i, t_i)\}$ and a visibility constraint. A solution to an MVPP-n consists of a set of paths $\{\pi_1, \ldots, \pi_n\}$ from s_i to t_i respectively and a schedule to move a set of agents $\{a_1, \ldots, a_n\}$ along π_i while preserving the visibility constraint. All of the paths $\{\pi_1, \ldots, \pi_n\}$ must remain inside the polygon. Each of the agents can remain stationary or move at a common fixed constant speed.*

Definition 17 *An* ideal solution *to MVPP-n is a solution which uses Euclidean shortest paths (Π_i).*

Where possible we shall attempt to produce ideal solutions or approximations to them. To simplify the discussion, we consider that n in an MVPP-n refers to the number of distinct agents. That is, if two agents share both start and target points, then they are treated as a single agent.

Adding more agents raises the question of how the visibility constraint should be defined for larger numbers of agents. In this thesis we discuss two possible constraints on the visibility graph: the *connected constraint* where the visibility graph must be connected and the *complete constraint* where the graph must be complete[1]. In instances where the connected

[1]These appear in Michell and Wynters' open problems [MW90b]. They use *clique visibility* rather than *complete visibility*.

constraint is applied, we do not require that the graph be a tree. That is, extra edges beyond what is needed to make the graph connected are permitted. With this in mind, we do not always treat the connected case separately if the complete case has already been dealt with.

In this chapter and Chapters 5, 6 and 7, we consider the instances where the start points for all paths are collinear and mutually visible (the segment joining the start points is denoted S). We assume the same for the target points (the line segment joining the target points is denoted T). Note that we apply the clipping assumption (Definition 9) to all paths. This chapter deals with instances where S and T do not cross (see Definition 13). We consider more general configurations of start and target points in Chapter 8.

Definition 18 *Two points in a set of collinear points* bound *the set if all the other points in the set lie between them or coincide with them.*

We want to employ the method from Chapter 3. To do this, we need to reduce the problem to one involving two agents. We do this by finding a pair of paths with all agent paths between them. Sections 4.2.1 and 4.2.2 give algorithms for constructing bounding paths from segments of a set of paths.

Definition 19 *Consider a set of paths $\{\pi_1, \ldots, \pi_k\}$ such that their start points are collinear and also their target points are collinear. Two non-self intersecting polylines π_a and π_b, with start points (s_a, s_b) and target points (t_a, t_b) on S and T respectively, are* bounding paths *for the set if the following are true:*

1. *All points on π_a and π_b belong to at least one of the paths in the set. That is, $\pi_a \cup \pi_b \subseteq \bigcup \pi_i$;*
2. *Point s_a coincides with an endpoint of S and t_a coincides with an endpoint of T. Point s_b lies on an end of S and t_b lies on an end of T;*
3. *Paths π_a and π_b do not cross (Definition 13 page 47);*
4. *No point on any of the paths is outside the region bounded by π_a, π_b, S and T.*

Since in this chapter, we will always be considering bounding the paths for all agents, the set being bounded will be omitted. Note that π_a and π_b could be members of $\{\pi_1, \ldots, \pi_k\}$ but are not required to be.

Item 2 does not necessarily mean the endpoints bound S and T. For example see Figure 4.1. Points s_a and s_b bound S but t_a and t_b do not bound T because they are not at

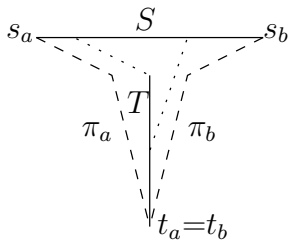


Figure 4.1: Instance where the endpoints of bounding paths π_a and π_b (shown dashed) do not bound S and T. Other paths are shown dotted.

.

opposite ends of T. Technically, some parts of the definition may be redundant because the clipping assumption removes some degenerate cases (for example, an instance where all start points are collinear with all target points). This definition does not preclude pairs of paths which intersect provided they do not cross. The definition for bounding paths is different in chapters 5, 6 and 7 but the same intuition will apply.

Definition 20 *The* initial bounding agents *for an MVPP-n are agents a_1 and a_2 such that s_1 and s_2 are the start points for the bounding paths. If there are multiple candidates, then choose the ones which contribute the longest prefix to the bounding paths. If there are still multiple candidates, then any of the remaining ones will do.*

Definition 21 Bounding agents *for an MVPP-n are agents a_1 and a_2 such that a_1 and a_2 are on the bounding paths.*

There should only be one bounding agent for each bounding path at a time. The idea, is to produce a schedule by walking along bounding paths, in the same way as in Chapter 3. Beginning with one of the initial bounding agents, when a bounding agent reaches the point where its path and the bounding path separate, another agent takes over the role of bounding agent.

In Section 4.1, we consider a subset of instances where Π_1 and Π_2 are known to be bounding paths. We do not claim, however, that this section identifies all such instances. Section 4.2 considers the remaining cases.

The following result is not new (for example see Mitchell and Wynters [MW90b]) but we prove it here for convenience.

Lemma 12 *The intersection between two Euclidean shortest paths is connected.*

Proof: Consider two Euclidean shortest paths Π_i and Π_k. If the intersection is empty, then it is considered to be connected. A disconnected intersection between two shortest paths would imply that there are multiple shortest paths between points in different (maximal connected) subsets of the intersection. One path taken by Π_k and another path taken by Π_i. This leads to a contradiction because shortest paths are unique in genus zero environments. So $\Pi_i \cap \Pi_k$ is connected.

□

First, though, we need some definitions and the following lemma.

Our general strategy will be to move bounding agents and then move the remaining agents so they lie on the chord between the bounding agents. After each movement of the bounding agents, the remaining agents will be moved to the chord between those agents. For this to work, we need to be sure that none of the agents will need to move back behind the chord once it has moved forward. We will rely on this lemma without further comment from this point on.

Lemma 13 *Consider a non-crossing MVPP-n where all agents not already at their targets are travelling on shortest paths and lie on a chord $\overline{L}$ between the bounding agents. No agent needs to move back behind $\overline{L}$ to reach its target.*

Proof: If the target is on the forward side of $\overline{L}$, then none of the paths can move behind $\overline{L}$. Doing so would lead to a disconnected intersection with $\overline{L}$.

If the target was behind $\overline{L}$, then the agents should have already reached it and so would not be considered.

□

We will need to describe interactions between paths which are not covered by our definition of *cross* (see Definition 13). Specifically, we need to include cases where the intersection between two paths occurs at an endpoint of one of the paths provided that the other path continues through the intersection. Such a case does not *cross* because it does not meet the requirement that points from the paths alternate around a circle.

Definition 22 *Two paths π_i and π_j* cut *at point p, if they cross at p or if:*

- $\|\pi_i\| > 0, \|\pi_j\| > 0$

  and

- *π_i and π_j have an ip-intersection at p (Definition 12 page 47);*

  and

- *$\exists d > 0$, such that $\forall 0 < \delta < d$, the intersection of $circle(p, \delta)$, π_i, π_j contains 3 points. (that is $\|circle(p, \delta) \cap \pi_i \cap \pi_j\| = 3$).*

Note that unlike cross, this definition is not symmetric.

Definition 23 *A* pinch *on a path is a segment with endpoints created by obstacles on opposite sides of the path. Since "side" is not well defined for endpoints of the path, a pinch cannot be the first or last segment of a path.*

Note that a pinch forms a chord of the polygon (for example Figure 4.2(a) on page 76).

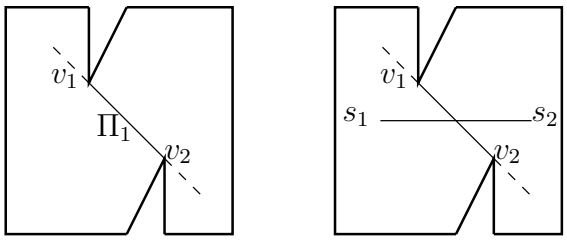
We will need to test for intersections between pairs of paths. Now we show how to do this in the two agent case.

Lemma 14 *In a two agent MVPP, if Π_1 and Π_2 intersect, then part of the intersection must occur in at least one of the following places:*

- *The first segment of Π_1 or Π_2;*
- *The last segment of Π_1 or Π_2;*
- *In the first pinch of Π_1 or Π_2.*

Hence testing for intersections can be performed in time $O(\|\Pi_1\| + \|\Pi_2\|)$.

Proof: Note that the endpoints of a segment are included in the segment for the purposes of intersection testing. Testing a path for intersection with a single segment can be done in time proportional to the length of the path. Once an intersection point is found the rest of the intersection can be calculated by walking along the paths (in both directions) until the paths separate. Since the intersection must be connected (Lemma 12) this will be all of it. So we can test for intersections on the first and last segments on both paths in $O(\|\Pi_1\| + \|\Pi_2\|)$ time.



(a) (b)

Figure 4.2: A pinch on Π_1 (which includes v_1 then v_2).

Now we consider the last bullet point. Assume that no part of the intersection is in the first or last segment of either path. Consider the first pinch on Π_1 which does not include s_1 or t_1 (assuming one exists). See Figure 4.2(a). In order for there not to be an intersection on the pinch, s_2 and t_2 must be on the same side of $\overline{v_1\, v_2}$. That is, the points s_2 and t_2 must both lie either to the left of $\overline{v_1\, v_2}$ or to the right of it ($\overline{v_1\, v_2}$ is a chord). If not, then Π_2 would intersect $\overline{v_1\, v_2}$.

So we have s_1 and t_1 on opposite sides of $\overline{v_1\, v_2}$ (in the figure, left and right respectively) as well as s_2 and t_2 on the same side as either s_1 or t_1. We know that the clipping assumption rules out sightlines running along $\overline{v_1\, v_2}$. If any endpoint lay on $\overline{v_1\, v_2}$, then there would be an intersection on that path's first or last segment. This leaves either S or T to cut $\overline{v_1\, v_2}$. Without loss of generality, assume S cuts $\overline{v_1\, v_2}$. However, for this to happen, s_1 would need to be below ${}^{ext}\overline{v_1\, v_2}$. (See Figure 4.2(b)). This leads to a contradiction because travelling along S to get to the intersection with $\overline{v_1\, v_2}$ must be shorter than going via v_1.

Pinches can be detected by walking the path provided some information about the direction of the polygon vertex is available in constant time. So all possibilities discussed so far can be checked with four traversals of each path. That is, in $O(\|\Pi_1\| + \|\Pi_2\|)$ time.

Now we must show that the intersection could not occur anywhere else. This leaves only one possibility for an intersection. It must occur before any pinches on both paths. This means that for the relevant parts of both paths, polygon vertices can only touch from one side (we are not counting the first and last vertices because they do not produce turns). So the paths must be either convex or just lines (possibly with vertices along them). Since a shortest path can only join another path at a polygon vertex and vertices from the polygon must appear on both paths, the only possible intersection is Π_2 crossing Π_1. See Figure 4.3.

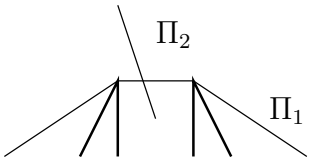


Figure 4.3: A hypothetical intersection between Π_1 and Π_2 which is not on the first or last segments.

Once Π_2 cuts Π_1 it cannot do so again (Lemma 12). But no point on or below the segment where the crossing occurs can be visible to t_1 since then T would be required to cut the obstacle which formed the vertex ending the segment. The one exception is the endpoints of the segment which will be visible if this segment connects directly to the first or last segments of the path. In that case though, if t_2 lies in that position, then it is also in the first or last segment and that possibility has already been excluded.

So, intersections can only occur in the places given in the hypothesis.

□

In the following, $\mathcal{I}$ denotes the number of intersections between paths.

Corollary 15 *Finding a point in each of the intersections between a shortest path (Π_i) and all paths in a set of n shortest paths with a total of $\mathcal{L}$ segments, can be performed in $O(\mathcal{I}+n)$ time with $O(\mathcal{L}+n)$ preprocessing time.*

Proof: The pinches for all paths can be found by walking the paths in $O(\mathcal{L}+n)$ time. In most cases, $n < \mathcal{L}$. The n term in the preprocessing cost is to account for pathological cases where some agents have paths containing zero segments. From Lemma 14, only (up to) three segments in each path matter for the purposes of intersection checking. The first segment, the last segment and the first pinch for Π_i can be tested against those segments for the other paths in $O(n)$ time. This gives an overall cost of $O(\mathcal{L}+n)$ time to test for an intersection or $O(\mathcal{L}+\mathcal{I}+n)$ time to report all intersections.

□

We will also want to find intersections between sets of lines.

Corollary 16 *Finding a point in all the intersections between paths in a set of n shortest paths with a total of $\mathcal{L}$ segments, can be performed in $O(\mathcal{I} + n \log n)$ time with $O(\mathcal{L} + n)$ preprocessing.*

Proof: Here we are only interested in finding a point which lies in each intersection (if the intersection is non-empty). If the entire intersection is required, then it can be found in time linear in the length of the path.

As in Corollary 15 pinches for all paths can be found with $O(\mathcal{L} + n)$ preprocessing.

To find intersections between all pairs of paths we will need to test for intersections between n sets of three segments. This could be done in $O(\mathcal{I} + n^2)$ time. However, Chazelle and Edelsbrunner [CE92] give an algorithm for finding the $\mathcal{I}$ intersections between n segments in $O(\mathcal{I}+n \log n)$ time. This gives an overall cost (including preprocessing) of $O(\mathcal{L}+\mathcal{I}+n \log n)$ time.

□

Instances will be classified depending on whether there are pairs of agents whose paths are bounding paths. Instances where this happens are addressed next. Instances where no such agents exist are dealt with in Section 4.2. The methods for solving the latter case also work on the former but they require extra computation.

4.1 Π_1 and Π_2 are Bounding Paths

Here we consider cases where Π_1 and Π_2 are bounding paths. A sufficient, but not necessary condition for Π_1 and Π_2 to be bounding paths is:

- Sightlines S and T do not cut, and
- points s_1, s_2 bound S, and
- points t_1, t_2 bound T, and
- paths Π_1 and Π_2 do not cross.

Lemma 17 *If S and T do not cut, and Π_1, Π_2 are bounding paths then, a schedule exists for the MVPP-n using paths $\Pi_1, \ldots, \Pi_n$ under the complete visibility constraint.*

Proof: First we consider instances where Π_1 and Π_2 have at least one segment each. We produce a schedule for a_1 and a_2 only, using the method from Chapter 3. This schedule is extended to include the other agents as follows.

At each step of the schedule, for the bounding paths, all the agents are collinear. Agents a_1 and a_2 move such that the sightline between them traces out a series of convex shapes (one for each stage of their schedule). Note that the degenerate case of a single point is also convex. Since the paths for the remaining agents never cut Π_1 or Π_2, they do not travel outside the convex shapes except through the sightline formed when a_1 and a_2 have completed their movement. That is, they exit through the ends of the shape, not the sides. Set the movement of each agent during that step, to be the portion of its path that lies inside the current convex shape. Since the shape is convex, all the agents are visible to each other at all times so the complete visibility constraint is preserved.

There is a special case to be considered here. Although it will not happen under the conditions described in this section, we note it to satisfy the preconditions of the lemma. If t_1 and t_2 do not bound T (for example situations like Figure 4.1), then it is possible for agents to reach their targets before either a_1 or a_2 does. Suppose agent a_f has reached its target. It must be able to see all the other agents' current positions and since it lies on T, a_f can see all the target positions as well. Since the paths are shortest, they will not cross sightlines from a_f to either their current positions or their target positions. Hence the rest of their journeys are visible by the triangle lemma. So for the purposes of an algorithm producing schedules, once an agent reaches its target it can be ignored.

Note that the definition of bounding path allows bounding paths to coincide provided that they do not actually cross. For example, see Figure 4.4 (page 80). These cases do not present problems.

Now we consider cases where either Π_1 or Π_2 is a single point. Without loss of generality assume $s_2 = t_2$. See Figure 4.5 (page 80). If both are single points then the instance is trivial. Since Π_1 must be s_2-restricted, form each step of the schedule by advancing a_1 to the beginning of its next segment. This will trace out a triangle with a_2. Move the remaining agents to be collinear with a_1 and a_2 (that is, the remaining agents, a_1 and a_2, all lie on a single line). This gives a solution under the complete visibility constraint. If only the

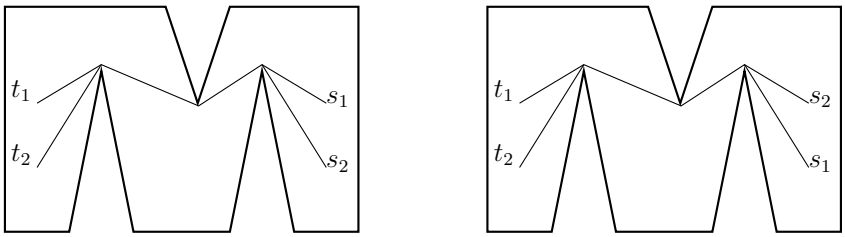


Figure 4.4: Two MVPP-n instances where Π_1 and Π_2 coincide but do not cross. Note that in the second example the order of the endpoints is reversed but the paths do not cross by our definition.

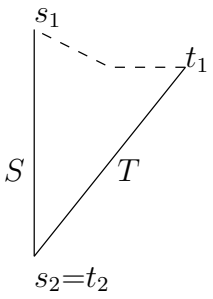


Figure 4.5: An MVPP-n with one end bounded by Π_1 and S and T meeting at a point.

connected visibility constraint is required, move all agents directly to their targets. Visibility is guaranteed by the triangle lemma based on a_2.

□

Now we analyse the complexity of the steps in Lemma 17.

The outer schedule can be produced in $O(\|\Pi_1\| + \|\Pi_2\|)$ time (Theorem 1 and Theorem 2). At the beginning of each step in the outer schedule, all agents will be on one face of a convex shape. At the end of the step all the agents will be on another face. So, for each step the final point of each agent's path within that shape must be computed. It may be that the end face does not intersect the current segment, in which case successive segments should be processed until the intersection point is found. However, once a segment is checked, none of the preceding segments needs to be checked again. So, each step of the initial bounding agents' schedule requires calculating n endpoints and possibly advancing a number of segments on

the internal paths. This gives an overall complexity of $O(n(\|\Pi_1\|+\|\Pi_2\|)+\Sigma_{k=3}^{n}\|\Pi_k\|)$, which is $O((\|\Pi_1\|+\|\Pi_2\|)n+L)$ where L is the total number of segments in all paths.

The next section considers instances where no pair of agents' paths are known to be bounding paths.

4.2 Split Bounding Paths

From this point on we will use τ to denote bounding paths. We will also use it to denote smaller connected parts of bounding paths, specifically split bounding paths which we define in this section. The context will make it clear which one applies.

Definition 24 *A path* exits *a current bounding path if it moves off the bounding path and outside the region bounded by S, T and the bounding path. The point where this occurs is denoted an* exit point.

We want a bounding path to build our schedule but the initial bounding paths may not bound the other paths for their entire journeys. A bounding path can be formed as follows. Add segments from the initial bounding path until the point where another path exits the initial bounding path. If this does not happen at a vertex of the initial bounding path, then add a segment up to the exit point. If multiple paths travel through the exit, then choose the segment which makes the largest non-reflex angle with the previous segment. Follow the current path, adding its segments to the polyline until it either reaches its target or another path moves across it. Repeat with each new path until one reaches its target. We give an efficient algorithm for calculating bounding paths later in this chapter.

A possible difficulty with this definition would be if another path cut the bounding path by travelling through the endpoint of the current path. This is avoided by the clipping assumption which ensures that all paths end once they reach T.

Definition 25 *The* split bounding path *is the polyline formed by taking the bounding path and removing the segments which lie on the initial bounding path. If all segments lie on the initial bounding path the split bounding path is not defined.*

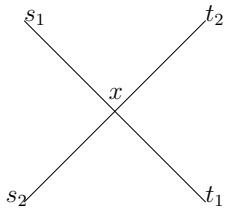


Figure 4.6: In this case the upper bounding path consists of the vertices s_1, x and t_2. This is not the shortest path between s_1 and t_2.

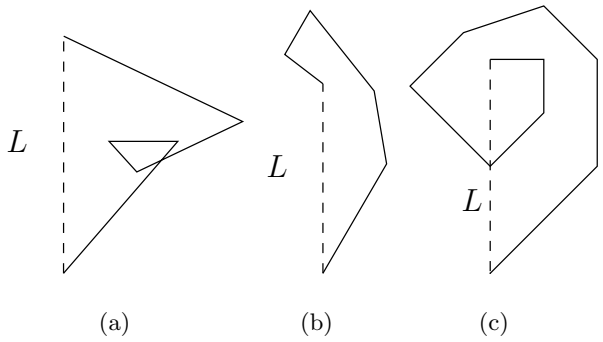


Figure 4.7: Illustration of why the preconditions in Lemma 18 are required. $\overline{L}$ is shown dashed.

Definition 26 *(From "Handbook of Discrete and Computational Geometry" [GO97]): A* Steiner point *is a "vertex not part of the input set".*

Split bounding paths may contain vertices which are not vertices of the polygon or path endpoints (Steiner points). For example, when a path exits the current bounding path in the middle of a path segment, a Steiner point will be added to the bounding path. This has implications for algorithms which walk along paths since it is not guaranteed that Steiner points have been added to all paths. So, points which are vertices on one path may not also be vertices on another path through the same points. However, since we will only be walking bounding paths, the next point can be checked against the current segment in constant time.

Now we are ready to deal with the case where the initial bounding paths are not bounding paths for their entire journey. Lemma 18 and Lemma 19 show that split bounding paths form convex chains. We use this information in our algorithm to construct bounding paths. Theorem 3 shows how to construct schedules using the bounding paths.

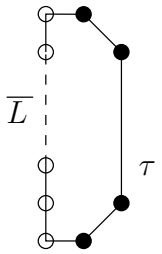


Figure 4.8: A simple convex polygon with a sequence of segments lying on the line through $\overline{L}$. Multiple successive segments collinear with $\overline{L}$ are shown with hollow circles.

Note that in the following proofs, τ might not be a shortest path. For example see Figure 4.6.

Lemma 18 *Consider a polyline (τ) and a line segment ($\overline{L}$) joining its endpoints. The polygon (P) formed from τ and $\overline{L}$ is either simple and convex or composed entirely of collinear points provided that the following three properties are all true.*

1. *Polyline τ turns in only one direction.*
2. *Polyline τ does not self-intersect.*
3. *Any intersection between τ and the line through $\overline{L}$ is contained entirely within sequences of (non-turning) segments with one end incident on its endpoints (the hollow circles in Figure 4.8).*

Proof: The prohibition on self-intersection is to prevent cases like Figure 4.7(a). Other non-convex possibilities are illustrated in Figure 4.7(b) and Figure 4.7(c) (which is also not simple). The exceptions to the prohibition on intersecting the line through $\overline{L}$ are to permit cases such as the one shown in Figure 4.8.

First we show that P is simple, then we show it is convex. Since we know that τ does not self-intersect, the only intersections to consider are those between τ and $\overline{L}$. Assume that the instance is oriented as shown in Figure 4.8, with τ starting at the lower end of $\overline{L}$. If τ leaves the line through $\overline{L}$ (downwards in the figure) or moves along it but away from the other endpoint of $\overline{L}$ (upwards in the figure), then P is simple since we know that no other intersections with $\overline{L}$ are permitted and that τ does not self-intersect. If τ travels along $\overline{L}$ towards the other endpoint, then either it coincides entirely with τ (and so all vertices in τ

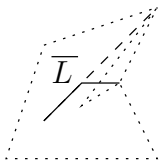


Figure 4.9: Polyline τ (shown as solid line) travels along $\overline{L}$ (shown dotted) then turns off. In order to connect to the other endpoint of $\overline{L}$ it must either make a reflex turn, self intersect or pass through the line through $\overline{L}$. These possibilities are shown dotted.

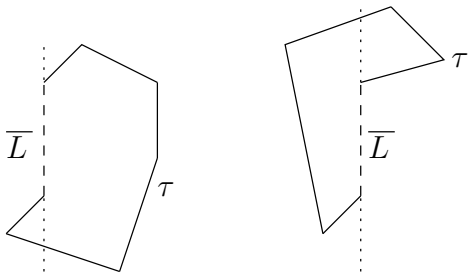


Figure 4.10: Hypothetical instances where part of τ lies on the right of the line through $\overline{L}$ (shown dotted). Here the polyline τ begins with a reflex turn (to the left of the line through $\overline{L}$). In order to reach the right side, τ must cross the the line through $\overline{L}$. However, τ can only intersect the line at segments which are collinear with $\overline{L}$ and hence cannot cross it.

are collinear and we are done) or it turns off $\overline{L}$. In that case, however, there is no way for τ to reach the endpoint without self intersecting, changing its direction of turn or intersecting the line through $\overline{L}$ at some disallowed point. See Figure 4.9 for an illustration.

Now we show that P is convex. Mathword [Wei] says that a simple polygon is "convex if and only if all turns from one edge vector to the next have the same sense." That is, all turns are in the same direction relative to the previous segment. In our case, though, some vertices of P may not actually turn. However, combining the segments incident on such vertices does not affect the convexity of P. For this reason, it is sufficient to show that all turns in P are in the same direction. All the turns in τ are in the same direction, so we need only consider the angles at the joining of τ and $\overline{L}$. If either of those angles was in the wrong direction (that is reflex angles), then τ could not connect to the endpoint of $\overline{L}$ because it would be

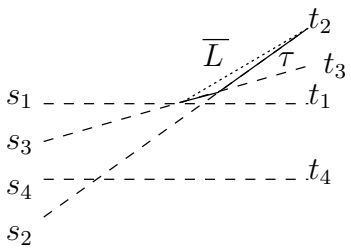


Figure 4.11: Illustration of Lemma 19. The split bounding path(τ) is shown as a solid line while $\overline{L}$ is dotted.

required to cross the line through $\overline{L}$ to get into a position to connect to the other end of $\overline{L}$. See Figure 4.10.

□

Lemma 19 *Consider an MVPP-n where the sightlines do not cross. If there is a split bounding path (τ), then either the vertices of τ are collinear, or the polygon (P), formed by joining the endpoints of τ, is simple and convex.*

Proof: Note: we are not claiming P contains no obstacles in its interior. If the split bounding path is a straight line (possibly with multiple vertices along it), then there is nothing to do. From now on, we will assume that the split bounding path does not form a line.

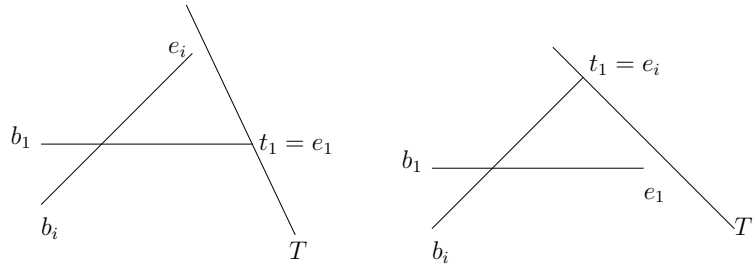
Denote the segment joining the endpoints of τ as $\overline{L}$. See Figure 4.11 for an illustration. In order to apply Lemma 18 we need to prove the following properties of τ:

1. It only turns in one direction — we do this by showing that obstacles can only touch τ on one side;

2. It does not self-intersect;

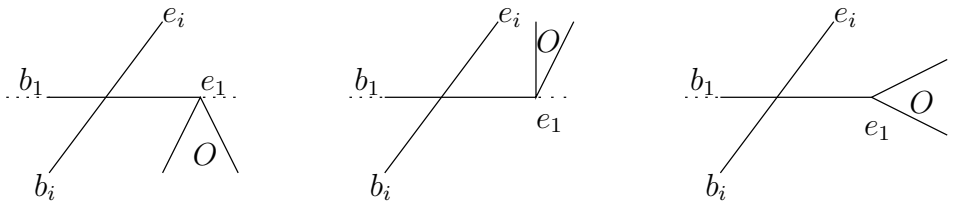
3. It does not intersect the line through $\overline{L}$, apart from a chain of segments lying in the line and ending at an endpoint of $\overline{L}$.

First, the direction of turns in τ. We begin with the exit point for τ. This is illustrated in Figures 4.12 and 4.13 (page 86). These figures show when the new bounding path exits mid-segment but the reasoning for the cases where the exit point is a vertex is identical. Suppose the initial bounding path is first cut on segment $\overline{b_1\, e_1}$ by segment $\overline{b_i\, e_i}$. This leads to the following possibilities:



(a) (b)

Figure 4.12: Cases where one of the crossing segments ends in a target point.



(a) (b) (c)

Figure 4.13: Possibilities for the position of obstacle O relative to e_i. The extension (C) of the segment $\overline{e_1\, b_1}$ is illustrated with a dotted line.

1. Either e_1 or e_i is a target point ($e_1 = t_1$ or $e_i = t_i$); refer to Figure 4.12;

2. Neither e_1 nor e_i is a target point. Denote ${}^{ext}\overline{b_1\, e_1}$ as C. The obstacle which created e_1, is denoted O. Figure 4.13 illustrates the following subcases:

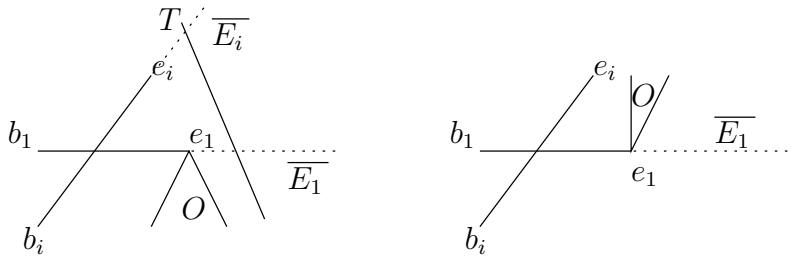
   (a) Obstacle O lies on the opposite side of C to e_i,

   (b) Obstacle O lies on the same side of C as e_i,

   (c) Segment C ends at O.

Case 1. If $e_1 = t_1$(Figure 4.12(a)), then τ can only have obstacles touching on one side, since obstacles cannot cut T or $\overline{b_1\, e_1}$. If $e_i = t_i$(Figure 4.12(b)), then τ can still only have obstacles touching on one side, since obstacles cannot cut T or $\overline{b_i\, e_i}$.

Now Case 2 (neither e_1 nor e_i is a target point). See Figure 4.13. Let $\overline{E_1}$ be the half-open line segment of C from e_1 (not including e_1) to the exterior of the polygon. (Note that this line segment may include polygon vertices). We now show that, of the cases shown in



(a) (b)

Figure 4.14: Greater detail of Figure 4.13(a) and Figure 4.13(b)

Figure 4.13, only Figure 4.13(a) is actually possible. Figure 4.14 shows $\overline{E_1}$ on Figure 4.13(a) and Figure 4.13(b).

First Figure 4.14(a) — O lies on the opposite side of C to e_i. Point t_1 must lie on or below $\overline{E_1}$. If this were not the case, then since Π_1 is a shortest path, it would need an downward pointing obstacle touching $\overline{E_1}$ from above to force Π_1 to turn up. By similar reasoning, we can see that t_i must be above $\overline{E_i}$ unless an obstacle touches Π_i from below. However, there is no way for such obstacles to exist because they would need to lie between the paths (since Π_i and Π_1 have already crossed they cannot do so again) and the polygon is genus zero. So, obstacles can only touch one side of τ. Instances fitting Figure 4.14(a) could exist, with t_1 below $\overline{E_1}$ and t_i above $\overline{E_i}$.

Now we examine the case where O lies on the same side of C as e_i (Figure 4.14(b)). If t_1 lies on, or above $\overline{E_1}$, then t_1 is not visible to any point above $\overline{b_1\, e_1}$ and to the left of O. Since e_i is in that position, Π_i would need to cross $\overline{b_1\, e_1}$ again, which is a contradiction (Π_i should be shortest). If t_1 lay below $\overline{E_1}$, then the another obstacle would need to touch $\overline{E_1}$ from below in order for Π_1 to be shortest. But in that case Π_i would still need to recross $\overline{b_1\, e_1}$ to gain visibility leading to the same contradiction. If t_1 is above $\overline{b_1\, e_1}$ and left of O, then Π_1 cannot be shortest either. So, there are no instances of the form shown in Figure 4.14(b).

If C ends at O (see Figure 4.13(c)), then we have a contradiction because Π_1 must continue and if it turned to either side of O, then it would produce a non-reflex turn with no obstacle inside. Such a path cannot be shortest. An obstacle inside the turn would mean the polygon of the MVPP-n would self intersect at e_1. Note that we are assuming that $e_1 \neq t_1$, since that case has already been dealt with above. So there are no instances of this type either.

So, we are left with only Figure 4.14(a). The next point on our list is that τ cannot self-intersect, which is part of the definition of bounding path.

Finally we address the question of intersection of the line through $\overline{L}$. If τ were to intersect the line (anywhere other than the permitted exceptions) it would need to turn away from the line (so as not to be part of an exception) and then later turn back towards it to connect with the endpoints of $\overline{L}$. However, this leads to a contradiction since it requires either that τ turns in more than one direction or that τ self-intersects.

Hence, by application of Lemma 18 the polygon is simple and convex.

□

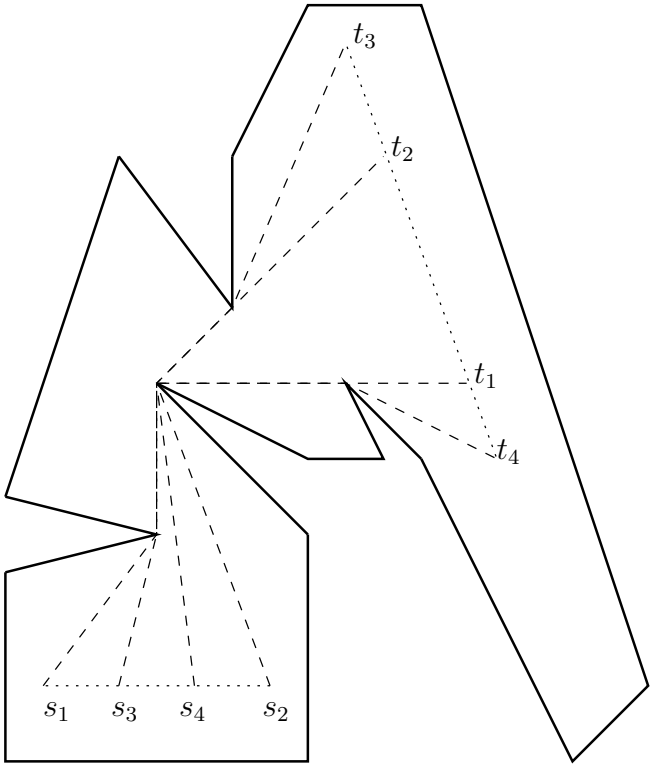
The roles of S and T could be reversed, giving a split bounding path in the reverse direction, which would also be convex. It is tempting to suppose, that these two split bounding paths could be combined to form an overall convex shape, but this is not always the case. For example, see Figure 4.15. One reason for this is that running the process in reverse would start with a different path and so the exit point might well be different.

Theorem 3 *An MVPP-n where S and T do not cross has a schedule using shortest paths under the complete visibility constraint.*

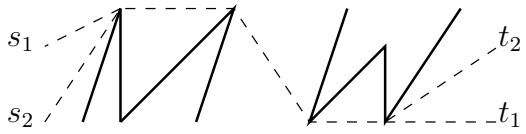
Proof: We follow the approach from Lemma 17. In fact, since we have Lemma 17, for the rest of the proof we only consider instances for which there are no Π_1, Π_2 such that Π_1 and Π_2 are bounding paths. This means that the agents acting in the role of bounding agents will change and also that some vertices on the bounding path may be Steiner points.

At each step, consider the current bounding agents in a two-agent MVPP. Consider a step of the schedule where the paths supplying the next segments for the bounding path are Π_a, Π_b. We know from Lemma 4, that at least one of the agents must be able to advance. If the agents do not advance a full segment from the original paths (for example, they may stop at a Steiner point), then this does not present a problem for visibility because the original reasoning was about moving in convex figures.

Once the bounding agents have advanced, the non-bounding agents move along their shortest paths so that they are collinear with the bounding agents and lie between them. It is possible that the next step of the schedule may require a different set of bounding paths.



(a)



(b)

Figure 4.15: Two MVPP-n instances where the "upper" bounding path contains a convex split bounding path but the overall bounding path is not convex.

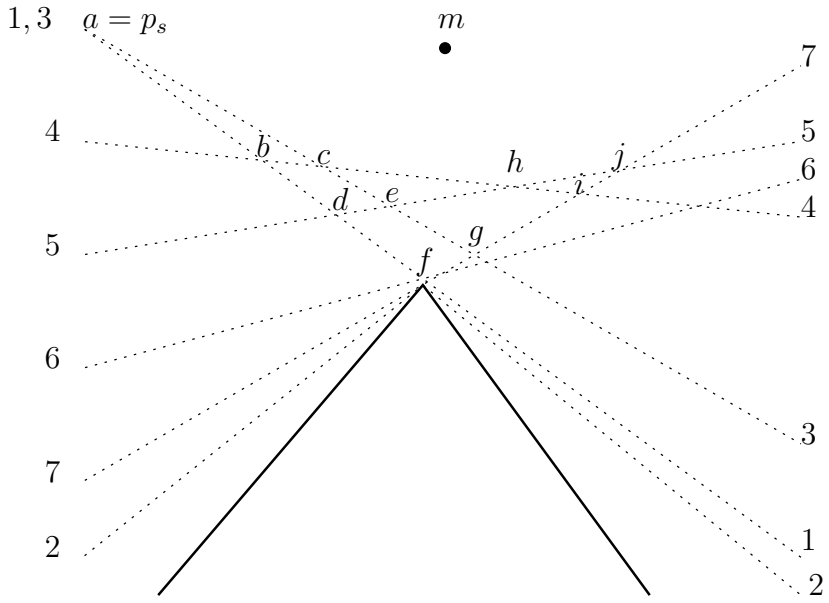


Figure 4.16: Part of an MVPP-n where a split bounding path exists. Intersections between paths are labelled $a \ldots j$ in radial order to m. Paths are labelled with their numbers.

This is not a problem since the new bounding agent must have advanced so it occupies the same point as the previous bounding agent.

□

We now give two algorithms to actually calculate a split bounding path.

4.2.1 Algorithm 1

The complete bounding path can be constructed by prepending the parts of the initial bounding path before the exit point. The correctness of this algorithm relies on the fact that a split bounding path forms a convex chain (Lemma 19). *Hence, in order to use this method you need to know that a split bounding path exists.* But that can be tested in $O(n + L)$ time (Corollary 15).

We need to know which end of T the split bounding path ends at (say t_i). Suppose Π_1 is the initial bounding path, if t_1 is on an endpoint of T, then t_i lies at the opposite end. (If t_i and t_1 were to coincide, then there would be no split bounding path.) If t_i is not at an end of T, then label as Π_f the first path to cut Π_1. Point t_i will be at the end of T which is closer to t_f than t_1 otherwise, Π_f would be crossed twice by the bounding path.

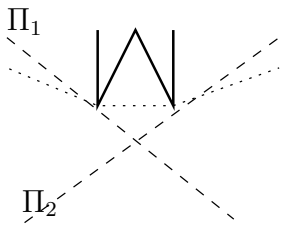


Figure 4.17: Vertices on paths outside the region bounded by the initial and final bounding paths.

The algorithm also needs to know the exit point (p_s) for the split bounding path. This can be found by determining the earliest crossing of the initial bounding path. Refer to Figure 4.16. Suppose we wish to construct the split bounding path τ from p_s to t_i.

1. Let V be a collection of all intersections between segments in the paths ($\|V\| = \mathcal{I}$). Any intersections which consist of whole segments should be removed.

2. Add the endpoints of all segments to V. This is necessary to include endpoints which lie outside the region bounded by S, T and the initial bounding path and final bounding paths. For example see Figure 4.17.

3. Take the midpoint (m) of the segment $\overline{p_s\, t_i}$. Sort V by angle (increasing) to m, with points with the same angle arranged in increasing distance from m. This could be done using a comparison function which checks both values. Alternatively, sort V first by distance to m and then by angle to m (the second sort would need to be stable[AHU74]).

4. Add p_s to τ and set the currentline to the next segment of the initial bounding path.

5. Repeat the following until p lies on T:

   i Let p be the next point in V.

   ii If p lies on the current line, then choose the segment ($\overline{L}$) through p which makes the most acute angle with the current line. Make the line through $\overline{L}$ the new current line and add p to τ. If p does not lie on the current line, then discard it.

To see why step 5 is correct, consider the situation where p does not lie on the current line. If p were on the opposite side of the current line to m, then to choose it would mean a non-convex τ. If p were on the same side of the current line as m, then p would have to

result from a segment which the algorithm has not encountered yet. This means at least one of the paths has turned back towards t_s and hence is not shortest.

To show that the algorithm produces bounding paths, we must show that τ can not self-intersect. Further, two distinct paths produced by the algorithm must not cut each other and their endpoints must bound S and T.

In order for τ to self-intersect, τ would need to make a reflex turn. Since τ is composed of parts of other paths and τ always takes the most acute angle, all the paths τ was following at the time would need to make a reflex turn. Shortest Euclidean paths could not do this unless there was an obstacle to move around. However, since the environment is genus zero τ could not connect with itself again. So, τ cannot self-intersect.

If τ were constructed for both sides (above and below in Figure 4.16), would all agent paths lie between them? Yes. A vertex between the upper bounding path τ and $\overline{p_s\, t_i}$ would be the result of a segment cutting τ (in which case that segment would have been chosen) or as the result of at least one path turning back towards its start point (hence not a shortest path). A path travelling above $\overline{p_s\, t_i}$ could not possibly be shortest. A symmetric argument means that no points could be below the lower bounding path.

The endpoint of each τ lies at an endpoint of T. If they did not, then there would have to be a path crossing before τ reached T and the algorithm would have followed that path. We do not need to consider cuts at the end of τ since such moves would be along T and the clipping assumption removes such cases. The algorithm starts from the exit point of the initial bounding path so when segments from the initial bounding path are prepended to the output of the algorithm a complete path which begins at an end of S is produced.

Finally, if the two τ were to cut, then that would mean there was a more acute choice for both paths and they did not take it. So we would have a contradiction.

Now we come to the complexity of the algorithm. The total number of segments in all paths is $\mathcal{L}$ and the number of intersections is $\mathcal{I}$. Note that $\mathcal{I}$ is bounded above by n^2.

- To find all intersections we require $O(n \log n + \mathcal{L} + \mathcal{I})$ time (Corollary 16). Including the segment endpoints this gives $O(\mathcal{I} + \mathcal{L})$ points.
- Sorting intersections requires $O((\mathcal{I} + \mathcal{L}) \log(\mathcal{I} + \mathcal{L}))$ time.
- Constructing τ from the intersections requires $O(\mathcal{I} + \mathcal{L})$ time, since each intersection needs to be checked.

So we have an overall worst case complexity of $O(n \log n + (\mathcal{I} + \mathcal{L}) \log(\mathcal{I} + \mathcal{L}))$ time.

The number of intersections, $\mathcal{I}$ is bounded above by n^2. The relationship between $\mathcal{L}$ and n is not so clear. In most cases, $n < \mathcal{L}$. To have $\mathcal{L} < n$ would require at least $\mathcal{L} + 1$ agents to have paths with zero length paths.

Now, we give an alternative algorithm which performs better for large $\mathcal{I}$.

4.2.2 Algorithm 2

Beginning with the initial bounding path, walk along the current bounding path to the next cutting point, then repeat with the cutting path. A candidate for the initial bounding path can be found in $O(n)$ time. As with the previous method, it does not matter if we do not choose the path which is a bounding path for the longest.

For each path, finding the next cutting path can be found in $O(n + \mathcal{L})$ (Corollary 15). In the worst case the bounding path will include segments from all paths so we have an overall worst case complexity of $O(n^2 + n\mathcal{L})$.

4.3 Summary

Lemma 14 showed that intersections between Euclidean minimal paths (with mutually visible endpoints) must occur in one of three places (the first segment, the last segment or on the first pinch).

Corollary 15 showed that a point in the intersections between one path and all the other paths in a set can be found in $O(n + \mathcal{I})$ time with $O(\mathcal{L} + n)$ time to find pinches.

Corollary 16 says that points in the intersections between all pairs of paths can be found in $O(n \log n + \mathcal{I})$ time with $O(\mathcal{L} + n)$ time to find pinches.

Lemma 17 shows that if two paths are bounding paths, then there is a schedule under the complete visibility constraint. This schedule can be produced in $O((\|\Pi_1\| + \|\Pi_2\|)n + \mathcal{L})$ time.

Theorem 3 shows how to construct schedules using bounding paths. Once bounding paths π_1 and π_2 are known, schedules can be produced in $O((\|\pi_1\| + \|\pi_2\|)n + \mathcal{L})$ time.

Algorithm 1 computes bounding paths in $O(n \log n + (\mathcal{I} + \mathcal{L}) \log(\mathcal{I} + \mathcal{L}))$. Algorithm 2 computes bounding paths in $O(n^2 + n\mathcal{L})$.

In the next three chapters, we discuss instances where S and T cross.

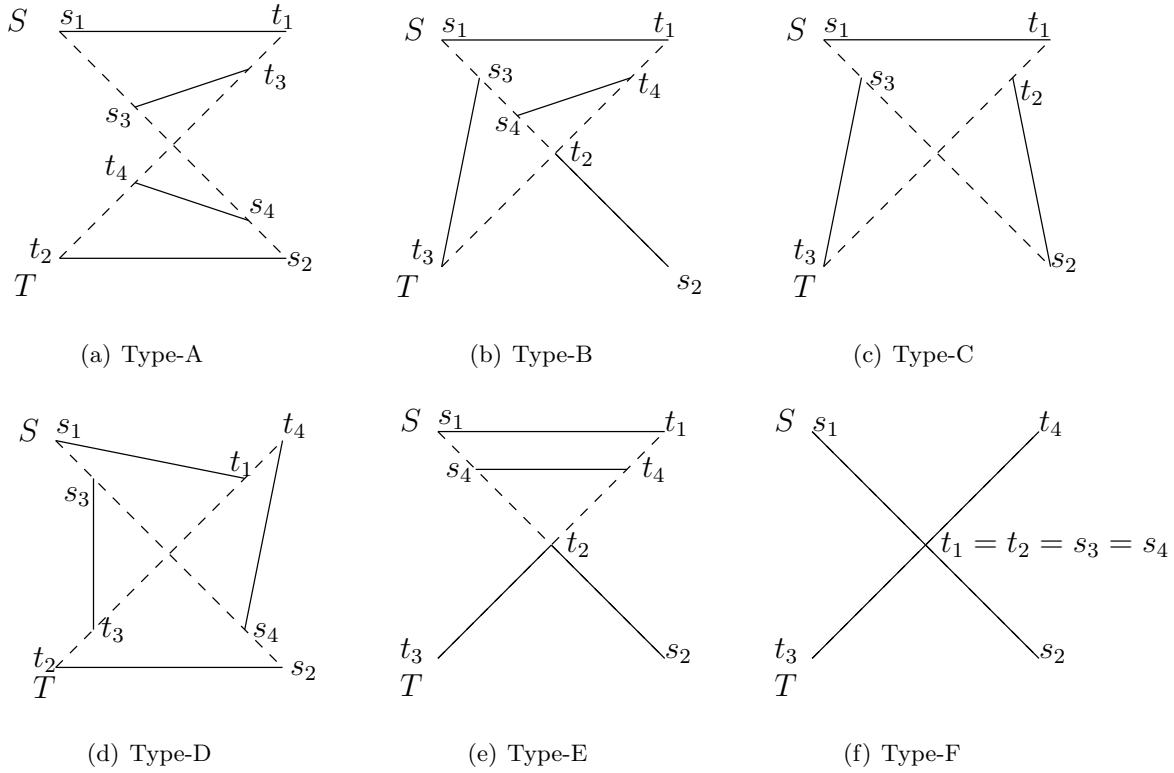
Chapter 5

Collinear Endpoints — the Crossing Case

Now we consider the case where the sightlines S and T cross. Throughout this chapter we denote the intersection of S and T as x. In this type of problem instance, there are more configurations that demand special attention than with the non-crossing and two agent crossing cases. In this chapter, the agents are numbered such that the initial bounding agents are a_1 and a_2 and that s_1 and s_2 are at opposite ends of S. Additional bounding agents if required will be numbered a_3 and a_4. The problem types that arise here are illustrated in Figure 5.1 (page 96) and we will address each case separately. A formal description of these types will be given after we have defined some terms. In this chapter, as well as in Chapters 6 and 7, we will prove the existence of schedules only. We do not measure the computational complexity of constructing them.

As with the two-agent case, some instances may have agents where the shortest path runs along the start or target sightline for part of its journey. In such cases the clipping assumption should be applied. If the path lies entirely within $S \cup T$, then both s and t should be set to x. Such adjustments will always be possible since it is contained within the starting sightline and remains a Euclidean shortest path.

This version of the clipping assumption means that Figure 5.1(b), Figure 5.1(e) and Figure 5.1(f) actually collapse to non-crossing instances. We will discuss 5.1(b) and Figure 5.1(e) anyway but not Figure 5.1(f) since it collapses to an instance where all agents lie at a single



(a) Type-A (b) Type-B (c) Type-C

(d) Type-D (e) Type-E (f) Type-F

Figure 5.1: Possible types of MVPP-n with crossing sightlines. Note that Type-B, Type-E and Type-F change due to the clipping assumption.

point. Type-C and type-D will be discussed in Chapters 6 and 7 respectively. The results for the cross case will be summarised in Section 7.2.

Producing schedules using the shortest paths (denoted Π_i) is not always possible (Corollary 9). We denote the paths used in this section π_i where it is not certain that shortest paths are being used. We are not allowing arbitrary paths, however. In the two-agent case, if the shortest path is not usable, then the size of the initial diversion only needs to be reasonably small. For the n-agent case, we impose an additional constraint:

> *During the start (and finishing) phase, all diversions must be small enough that they do not cross any other paths which start (or finish) closer to x than they do.*

That is, more generally, no path should have a disconnected intersection with any other path.

Definition 27 *A* sector *is one of the four regions (including borders) bounded by the sightlines S and T. A sector is* non-trivially occupied *if the path of at least one agent intersects the interior of the sector.*

Now we can formally describe the types in Figure 5.1.

- Type-A: Instances consist of two non-trivially occupied sectors opposite each other.
- Type-B: Instances consist of two non-trivially occupied sectors adjacent to each other. Since S must extend past T, there must be at least one agent with its start point on the far side of T and its target point at x.
- Type-C: Instances consist of three non-trivially occupied sectors.
- Type-D: Instances consist of four non-trivially occupied sectors.
- Type-E: Instances have one non-trivially occupied sector. Sightlines S and T are extended through x by agents which start on S or T and end at x (and vice versa).
- Type-F: All agents either start or end at x.

For the instances described in this chapter, we need to use a different definition of *bounding path* and *initial bounding path* since the ones used in Chapter 4 do not make sense in cases such as Figure 5.1(b).

Definition 28 *The* bounding path τ_K *for a sector K is a path of minimal length, composed of segments (and connected fragments of segments) from paths π_i such that, all points on all paths π_i lie in the closed region bounded by S, T and τ_K.*

Each non-trivially occupied sector has a bounding path. As in Chapter 4, we will use τ to denote both bounding paths and split bounding paths (the context will make it clear which is meant).

Definition 29 *The* initial bounding agent *for a sector, is an agent a_i, such that the bounding path τ begins at s_i and $\|\Pi_i \cap \tau\|$ is maximal. The path π_i for the initial bounding agent is termed the* initial bounding path.

The endpoint of the final segment in the bounding path for sector K is denoted t_K (the corresponding start point is denoted s_K). The set of agents whose paths enter the interior of sector K is denoted Ag_K.

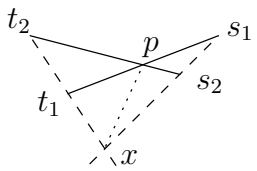


Figure 5.2: Path π_1 runs outside π_2 until p so it is the initial bounding path. A split bounding path is formed when π_2 cuts π_1 at p.

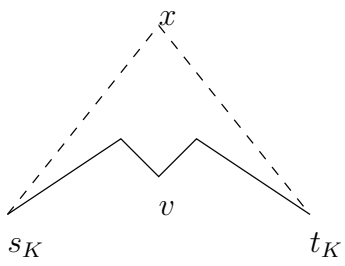


Figure 5.3: A hypothetical non-convex bounding path (shown with unbroken lines).

The definition of split bounding path is the same as in the previous chapter, that is: if the bounding path does not consist only of the initial bounding path, then the remainder of the bounding path is denoted the split bounding path. See Figure 5.2 for an example.

Lemma 20 *The bounding path for a non-trivially occupied sector is x-restricted.*

Proof: Choose a sector and denote it Sector K. Since there is no Π travelling through the sector which can leave it and be shortest all we need to show is that the bounding path is convex and lies between x and the shortest path between s_K and t_K.

Denote the shortest path between s_K and t_K as Π_K. For the purposes of contradiction, suppose that the bounding path is not convex. Let v be a vertex of the bounding path where the path makes a non-reflex turn. See Figure 5.3 (page 98) for an illustration. In order for the bounding path to reach v, there must be a Π which travels to v. Since the turn at v is non-reflex, Π turns away from t_K (that is, upwards in the figure). If it did not, then the bounding path would follow Π rather than turning. But there are no obstacles on that side (above in the figure) of the bounding path to force the path to divert like that, so we have our contradiction.

□

If shortest paths are in use, then bounding paths can be constructed using the algorithms given in Chapter 4. The paths to be checked for intersections can be limited to those which lie in the relevant sector. However, for type-A instances, the paths might not be shortest so Lemma 14 does not apply. This does not invalidate the overall algorithms but it does mean that all segments in each path must be checked for intersection. The complexity will increase to $O(\mathcal{L}\log\mathcal{L}+\mathcal{I})$ (Algorithm 1) and $O(n\mathcal{L}\log\mathcal{L})$ (Algorithm 2) time in the worst case.

As with the non-crossing case we need to be sure that positioning agents on the line between two bounding agents does not result in any of the agents needing to move back behind the line. This is established by the following lemmas. From this point on, we will rely on these lemmas without further comment.

Lemma 21 *Consider a crossing MVPP-n with agents in opposite sectors with all agents in those sectors which have not reached their targets moving on x-restricted paths and on a segment $\overline{L}$ between the bounding agents. No agents are required to move behind $\overline{L}$ in order to reach their targets.*

Proof: To turn back behind $\overline{L}$, would require a turn away from the target. Since all paths are x-restricted and therefore convex, this is not possible.
□

Lemma 22 *Consider a MVPP-n with agents moving in two adjacent sectors with all agents not at their targets, moving on x-restricted paths and on a segment $\overline{L}$ between the bounding agents. No agent is required to move behind $\overline{L}$ in order to reach its target.*

Proof: By contradiction. Suppose a_i at p will need to move behind $\overline{L}$ to reach t_i. Let q be the intersection point between $\overline{L}$ and S.

Note that $\overline{p\,q}$ lies outside (or on the border of) the region bounded by T, $\overline{p\,x}$ and the remainder of π_i. Since π_i is x-restricted, π_i cannot move behind $\overline{p\,x}$ (and hence $\overline{p\,q}$).

But what about moving behind the part of $\overline{L}$ past p? Since $\overline{L}$ is defined by the bounding agents, a path moving behind that section of $\overline{L}$ would mean that the bounding path was invalid and we have a contradiction.
□

5.1 Analysis of Type-A Instances

Instances of this type have two opposite sectors non-trivially occupied.

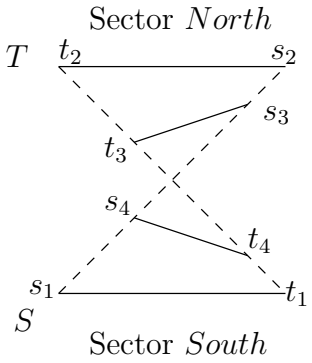


Figure 5.4: A type-A instance.

Consider an MVPP-n where S and T cross, s_1 and s_2 bound S while t_1 and t_2 bound T. For this section, we will orient figures so that the sectors containing a_1 will be at the bottom. We will designate these sectors containing a_1 and a_2 as Sector *South* and Sector *North* respectively. Unlike the two agent case, we must consider the situation where the bounding path contains a split bounding path and thus different agents might be the bounding agent at different times. This does not actually present a problem as long as schedules pause at vertices of the bounding path to allow a change of bounding agent. This process is assumed to be implicit from this point on.

Note that the the clipping assumption has implications within this instance type. See Figure 5.5 and Figure 5.6 for examples.

5.1.1 Starting (and Finishing)

The problem of starting and finishing a schedule that was noted in the two-agent case still applies here. Moreover, it is more complicated now because there are additional agents to consider. We show how to start under various conditions. Unlike the two agent case, it is not known whether it is always possible to construct approximate shortest paths under the complete constraint. We do, however, limit the cases under which this failure could occur.

Starting a schedule in this n-agent case means all agents have moved along their paths such that either they no longer lie on S, or they have reached their target point and comply with the prevailing vision constraint.

While finishing a schedule is a symmetric problem to starting one, we will still need to show that the agents can be moved to a position to begin the finishing procedure. Also, determining whether Π_i can be used for a_i requires checking both starting and finishing conditions. For now we describe how to start a schedule.

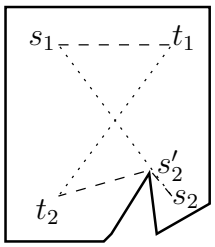


Figure 5.5: An instance where a shortest path lies along S. The adjusted start point is labelled s_2'.

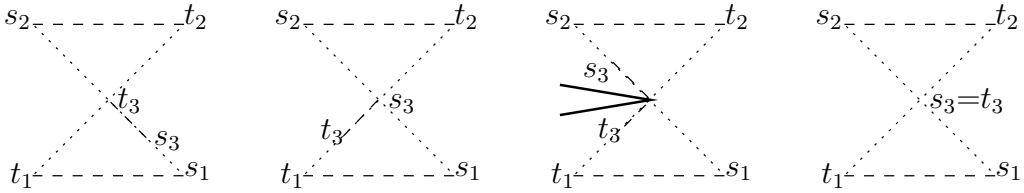


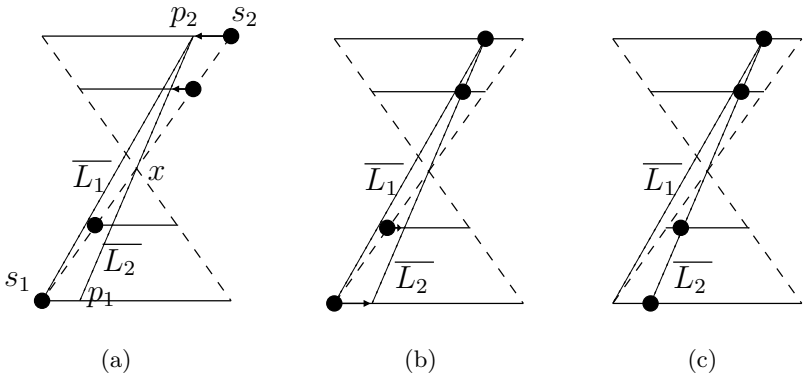
Figure 5.6: Examples of MVPP-n instances with shortest paths which lie entirely within the sightlines. The clipping assumption converts all of these into (d) where s_3 and t_3 both lie at the intersection of S and T.

First we will show that starting under either constraint in non-trivial visibility is possible. Section 5.1.3 covers the Starting task with trivial visibility under the connected constraint. Section 5.1.4 covers the complete constraint.

5.1.2 Starting Under Either Constraint is Possible with Non-Trivial Visibility

The following lemma proves that starting under either constraint is possible in non-zero measure.

Lemma 23 *In a type-A MVPP-n where at least one of the initial bounding agents has non-trivial visibility of the other, it is possible to start a schedule using x-restricted paths under the complete constraint.*



(a) (b) (c)

Figure 5.7: Steps to start a schedule under trivial visibility.

Proof: Refer to Figure 5.7. Note that shortest paths must be x-restricted. Without loss of generality, assume s_1 has non-trivial visibility of s_2. In this proof we wish to distinguish between line segments which comprise the path (which we shall refer to as *path segments*) and line segments between arbitrary points in the polygon.

Once this is done, let the furthest point along π_2 which s_1 can see (or the end of the first segment, whichever comes first) be denoted p_2. Let the sightline to p_2 be $\overline{L_1}$. Let the segment from p_2 through x to a point on π_1 be denoted $\overline{L_2}$ (its endpoint on π_1 is denoted p_1). If p_1 is not on the same path segment as s_1, then set p_1 as the end point of the first path segment. Since the whole of $\overline{L_2}$ lies inside the region S, T, π_1, π_2 it must be a sightline.

Now we produce the schedule. Move all agents in Sector *North* so that they lie on $\overline{L_1}$. See Figure 5.7(b). This preserves visibility since, for the whole step all the agents are inside the triangle s_1, s_2, p_2. Next, move all agents in a_1's sector to lie on $\overline{L_2}$ (Figure 5.7(c)). All points lie in the triangle s_1, p_2, p_1.

Thus, all agents have moved off S (or are at x) and are mutually visible.

□

Notice that apart from any agents starting at x, all the agents are collinear at the end of this process.

5.1.3 Starting Under the Connected Constraint with Trivial Visibility

Lemma 24 *Consider a type-A MVPP-n where the initial bounding agents have trivial visibility of each other. It is possible to start a schedule (although perhaps not an ideal one) under the connected constraint with x-restricted paths.*

Proof: We know that with only two agents it is always possible to start, even if the shortest paths cannot be used (Corollary 10). The first step is to start the bounding agents. Provided the moves made are small enough, the bounding agents will not change direction during this process, so there is no danger of losing sight of agents still on S. The area swept out by the sightline between the bounding agents forms two triangles which meet at their apexes, each containing a bounding agent. Now move all the internal agents along their shortest paths to the edges of their triangles.

□

Lemma 25 *A type-A MVPP-n has a (possibly non-ideal) schedule under the connected constraint.*

Proof: If one of the agents starts at x, then move all agents directly to their targets. This completes the schedule since all the paths will be x-restricted. The agent at x will ensure visibility.

Suppose none of the agents start at x. We know from Lemma 24 that there exist pairs of bounding paths which allow a schedule to be started (and if necessary, finished). In this case, however, the agents which start moving along the bounding paths might not be the same agents which finish on the bounding paths. With this in mind, choosing paths might entail choosing one path which starts and finishes or two paths, one of which starts and the other finishes. Either way, we assume that a suitable set of paths has been chosen.

First, calculate a schedule for the two bounding paths and move the bounding agents along it as per the two-agent case. At each vertex of the bounding paths, have the bounding agents pause. Move the remaining agents in the moving agent's sector to lie on the sightline between the two bounding agents. Continue this process until the bounding agents are on their last segments and are the same small distance from their targets. At this stage the remaining agents should move directly to their targets and the bounding agents should execute the two-agent finishing procedure.

□

5.1.4 Solutions with Trivial Visibility (Complete Constraint)

If shortest paths do not work, then replacement paths need to be constructed to meet the following criteria:

- They need to be able to start;
- They need to be able to finish;
- They not have disconnected intersections with other paths;
- The paths must be x-restricted. This requirement may be relaxed to apply only to the section between the start and finishing tasks if visibility during those tasks can be guaranteed by other means.

Since starting and finishing are symmetric tasks we will focus on starting. Divide the agents into Ag_A and Ag_C (for start points on each side of x).

We will move agents in such a way that agents in the same sector remain in a triangle known to be clear of obstacles. This will ensure that agents are visible to the others in their sector. We also need to show that each agent moves so as to be visible to all agents in the other sector. This can be tested by applying Lemma 8 to each pair of agents $a_i \in Ag_A,\ a_j \in Ag_C$. This gives an $O(\mathcal{V}+n^2)$ time test to determine whether visibility is trivial and if so, whether starting is possible using the shortest paths.

So we end up with a system of inequalities (the meaning of the symbols is shown in Figure 3.13 (page 59)):

$$\forall a_i \in Ag_A, a_j \in Ag_C \quad 0 \leq (D_j \sin\alpha_i - D_i \sin\beta_j)/(\sin\beta_j + \sin\alpha_i) \leq c_r. \tag{5.1}$$

If only the departure angles are changed, then this gives $\|Ag_A\| \cdot \|Ag_C\|$ double sided inequalities. An additional $\|Ag_A\| + \|Ag_C\|$ double sided inequalities are required to constrain the sines of the angles to the required interval. In this case the inequalities are linear in the sine of the departure angles and have $\|Ag_A\| + \|Ag_C\|$ unknowns. A solution to such a system would allow us to use the method from Chapter 3 to produce a solution which approximates the ideal. If the start points themselves are allowed to move (giving varying D values), then the system has twice the number of unknowns, and hence is more flexible, but it is no longer linear and does not appear to guarantee a solution which approximates the ideal.

The interval for the sines of the departure angles cannot be larger than $(0, 1]$ but may be smaller to prevent paths running into obstacles. Lemma 8 imposes extra constraints on solutions to the equations which lie at the extremes of the safe interval. The value of $\sin(\beta_j - \alpha_i)/(\sin\beta_j + \sin\alpha_i)$ must be ≥ 0 or ≤ 0 for values equal to 0 and c_r respectively. For this reason if the parameters need to be adjusted, then we would prefer a set of parameters lying in the interior of the solution region.

For now, we focus on the linear system. Corollary 10 showed that it is possible to adapt a departure angle on one side of x to a fixed angle on the other side. This means that any instance with only one agent on one side of x will always have a solution. Fix the lone agent's departure angle and adapt all the other agents to it.

For another example consider the following instance:

Agent a_1 at $y = 10$.	Agent a_2 at $y = 2$.
Agent a_3 at $y = 50$.	Agent b_1 at $y = 15$.
Agent b_2 at $y = 20$.	c_r equal to $\frac{1}{12}$.

This leads to the following system[1].

$$
\begin{array}{rrlrrl}
0 < & 15a_1 - 10b_1 & < \tfrac{1}{12}(a_1 + b_1), & 0 < & a_1 & < 1, \\
0 < & 20a_1 - 10b_2 & < \tfrac{1}{12}(a_1 + b_2), & 0 < & a_2 & < 1, \\
0 < & 15a_2 - 2b_1 & < \tfrac{1}{12}(a_2 + b_1), & 0 < & a_3 & < 1, \\
0 < & 20a_2 - 2b_2 & < \tfrac{1}{12}(a_2 + b_2), & 0 < & b_1 & < 1, \\
0 < & 15a_3 - 50b_1 & < \tfrac{1}{12}(a_3 + b_1), & 0 < & b_2 & < 1, \\
0 < & 20a_3 - 50b_2 & < \tfrac{1}{12}(a_3 + b_2) &&&
\end{array}
$$

Mathematica solved this system[2]. However, if a single solution is required rather than a description of all possible solutions, other tools or methods should be employed such as the simplex method (see Section 5.4). Part of the solution region was:

[1] Note the strict inequalities since we want an answer which lies in the interior of the solution region. For brevity, the unknowns represent the sines of the departure angles.

[2] The Mathematica 4.1 *Timing* function returned 1874.55 seconds ($\approx$ 31 minutes) on a PowerMac G4 (2x1GHz processors) with 1Gb of RAM running OSX 10.4.

$$\begin{aligned}
0 &< a_1 &&< \frac{239}{1202}, \\
\frac{239}{1210}\, a_1 &< a_2 &&< \frac{601}{3025}\, a_1, \\
\frac{1195}{242}\, a_1 &< a_3 &&< 25\, a_2, \\
\frac{179}{121}\, a_1 &< b_1 &&< \frac{3}{10}\, a_3, \\
\frac{239}{121}\, a_1 &< b_2 &&< \frac{2}{5}\, a_3
\end{aligned}$$

If there are start points located in the gap between 0 and c_r, then the start procedure for those agents changes. Since such agents would have non-trivial visibility of all other start points, the remaining agents can be moved (small distances) without considering them. That is, the number of equations in the system can be reduced.

Once the other agents have been moved, the agents in the gap between 0 and c_r are moved as follows. Each agent should be advanced to the sightline between the two bounding agents. This starts all agents with the exception of an agent which lies on the intersection of the sightline and S. While this agent has not technically started it can be moved when the sightline moves off it. If the sightline runs through that start point for the entire journey, then there can be no visibility problems and the finishing procedure can handle it.

What if there is no solution to the system of equations (5.1)? A startable instance can be produced, eventually, by repeatedly applying some combination of the following methods:

- Moving an agent into the gap between 0 and c_r.
- Moving an agent so it shares a start point with another agent and shares the same departure angle.

Either of these steps will reduce the number of equations in the system and so we must arrive at a solvable system eventually. However, using these two steps means that the resulting paths might be significantly longer than the shortest paths. Section 5.4 has more discussion on solving these systems and the existence of solutions on the border requiring the extra constraints.

Once we have paths which satisfy the Starting and Finishing tasks, we only need to show how to schedule movement between these two tasks. As noted earlier, it is possible that a solution to the inequalities for Starting and Finishing tasks does not give x-restricted paths. For example, if π_i starts at a wider angle than Π_i. This does not actually present a problem

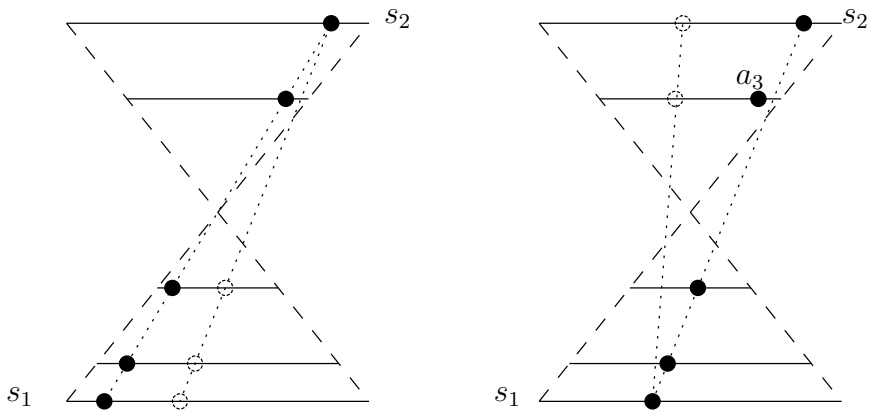


Figure 5.8: Two steps for a schedule under the complete constraint. Sightlines S and T are shown in bold, other sightlines are dotted and paths are shown normal.

since this will only happen in the measure zero case and then start/finish simultaneous moves take care of visibility. The section of path between those two tasks must be x-restricted, though. If the method for constructing paths given in Chapter 3 is used, then this will be the case.

Theorem 4 *A type-A MVPP-n with paths π_i with the properties listed at the beginning of this section (5.1.4 page 104) has a schedule under the complete constraint (and therefore also under the connected constraint).*

Proof: We show this by construction. Once the instance is started, the agents can be moved so that they all lie on the line segment between the bounding agents. Refer to Figure 5.8. Consider the schedule produced by solving the problem for the initial bounding agents (Chapter 3). Without loss of generality assume a_1 is first to move. Move a_1 until it reaches the next vertex of the bounding path or until further movement would break the sightline between a_1 and a_2. Any remaining agents in a_1's sector should move as far as possible along their paths in the region bounded by the sightlines from a_2 to a_1's old and new positions. There are no problems with visibility here since all agents are in the triangle with vertices at a_2's current position, a_1's old position and a_1's new position. This triangle is known to be free of obstacles. If a_1 stopped due to a bounding path, rather than to prevent loss of visibility of a_2, then it should move again.

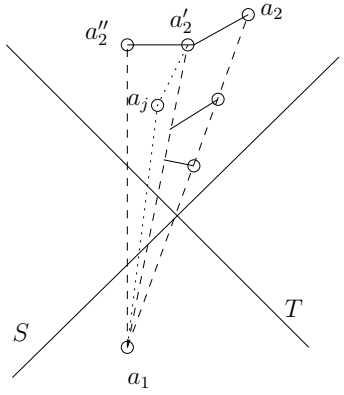


Figure 5.9: Agent a_2 moves from a_2' and later to a_2''. Agent a_j is to the left of $\overline{a_1\, a_2'}$ and so remains stationary until a_2 moves past a_2'.

Moving agent a_2 is illustrated in Figure 5.8(b). Now the agents are all contained within the region swept by $\overline{a_1\, a_2}$ so the same movements can be applied with one small caveat. Since a_2 must pause at vertices, one of the other agents might already be on the far side of the sightline between a_2 and a_1. For example a_3 in Figure 5.8(b). If a_2 made a very small move before encountering a vertex, then a_3 would already be ahead of the sightline. In this case, the algorithm should ignore such agents until they need to be moved forward in order to lie on the sightline.

Each time the bounding agent pauses, the remaining agents in its sector should be advanced to the new sightline between the bounding agents. To see that there are no visibility problems during this step note that all the moving agents in Sector $North$ remain within the triangle swept out by $\overline{a_1\, a_2}$. Further, for all a_i in Sector $South$, segments drawn from a_i to the old and new positions of any moving agent in Sector $North$ are also inside the triangle. For any stationary agent a_j in Sector $North$, see Figure 5.9. Combining the triangles $<a_1$, a_2, $a_2'>$ and $<a_j$, a_1, $a_2'>$ (both of which lie inside $<a_1$, a_2, $a_2''>$) gives a convex quadrilateral which contains the path sections of all the moving agents in Sector $North$.

This second step can be repeated (with the labels for bounding agents switched) as required. All that remains is to show how the schedule can be finished. First, the case when at least one of the ends of the bounding paths has non-trivial visibility of the other. In this case, continue the steps until one of the bounding agents can see the endpoint of the other agent. Then run the process from Lemma 23 in reverse.

Now we come to the case when both target points have trivial visibility of each other. If the agents can be positioned such that they are all on the final segments of their paths and are the same (small) distance from their targets, then they can move there simultaneously (we know this is possible from the lemma statement). Choose a d which is small enough so that when all agents are within d of their targets they must all be on the final segments of their paths. If all agents are mutually visible and agent a_k is at the desired distance, then a_k need not be considered further. This is because the points which are distance d from targets on all other paths must be visible to a_k from its current position. So since all the agents are moving along single line segments, the triangle lemma ensures visibility between a_k and all other agents. Continue the steps above until the bounding agents have reached distance d from their targets. At this stage all agents are either at distance d from their targets or are collinear with the bounding agents. Now since the bounding agents can see all the required points in the opposite sector, the remaining agents in the opposite sector should be moved directly to their targets. The sightlines from the remaining agents in a_k's sector will be inside the triangle thus produced, so sight is preserved for them as well.

Once all agents have reached distance d from their targets they can move to their targets simultaneously.

□

Theorem 5 *Consider a type-A MVPP-n where one of the initial bounding agents has non-trivial visibility of the other and where one of the final bounding agents has non-trivial visibility of the other. Such an instance has an ideal solution under either constraint.*

Proof: Lemma 23 ensures that the agents can start (and by symmetry finish). All that remains is to move the agents between the two tasks. This can be done using the procedure from Theorem 4.

□

In summary, type-A instances under the connected constraint always have a schedule which approximates the ideal solution. Instances under the complete constraint can always be transformed into an instance which has a solution, but the paths may not be close to the original instance's ideal length. If suitable paths are found, then a schedule exists.

5.2 Analysis of Type-B Instances

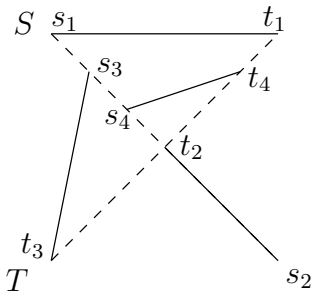


Figure 5.10: An example of a type-B MVPP-n.

In this type of instance, precisely two sectors are non-trivially occupied and those sectors are adjacent. The clipping assumption reduces this to a non-crossing instance. However, we discuss this type here anyway, since the algorithm given here will be adapted for type-C instances. An example of this case is shown in Figure 5.10. Note that the adjacent sectors form a funnel. Since in this type of instance, individual shortest paths do admit a schedule, we use Π_i rather than π_i here.

Lemma 26 *A type-B MVPP-n has an ideal solution under the connected vision constraint.*

Proof: In this type (after clipping) there must be an agent (say a_2) at x. Since a_2 can see both S and T and all Π_i will be convex since they can only be touched on one side, a_2 must be able to see all points on Π_i by the triangle lemma. So move all agents directly to their destinations.

□

Lemma 27 *A type-B MVPP-n has an ideal solution under the complete vision constraint.*

Proof: Any agents at x can be ignored since visibility between them and the other agents is assured by the triangle lemma.

Now we consider the remaining agents. First consider the case where the initial bounding paths (Π_1 and Π_3) form the bounding paths. That is, there are no split bounding paths. Let $I = \{i \in \{1, \ldots, n\} : s_i \neq x\}$. Project each vertex in Π_1 and Π_3 onto S using lines parallel to T. Combine these projections with s_i for $i \in I$ to form a sequence $\{v_1, v_2, \ldots\}$ ordered by decreasing distance to x (as shown in Figure 5.11). The sequence may contain repeated

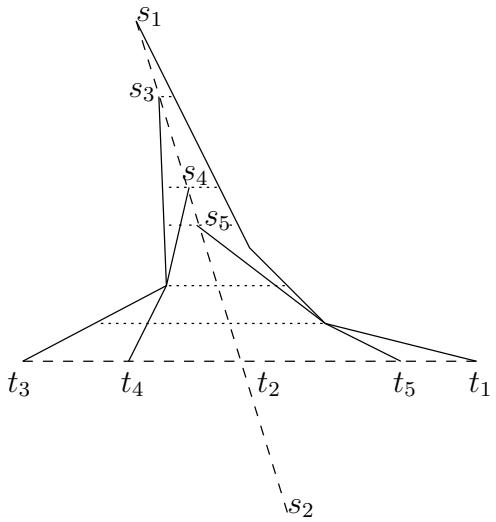


Figure 5.11: A type-B MVPP-n showing lines parallel to T through vertices.

points, but this does not affect the algorithm and they can be removed or ignored at the implementor's discretion.

Move any agents at v_1 along their paths until they reach the line through v_2 (parallel to T). Wait for all moving agents to arrive, then move the agents to the next line (including any agents with startpoints at v_2. Repeat the process until all agents $a_{i\in I}$ have reached their targets.

We need to show that visibility is preserved in this process. Since the bounding path must be convex, at each stage all agents $a_{i\in I}$ are in a trapezoid (which is convex), so they have visibility. So we have a schedule which preserves visibility.

Now we address the case where one (or both) of the bounding paths has a split bounding path. Whether this case applies can be determined by checking if t_1 and t_3 bound $\{t_{i\in I}\}$. Since obstacles can only interact with the paths on at most one side, a convex chain can be constructed using the same method given in Chapter 4. The vertices for this chain can be used instead of the ones from the shortest paths in the previous part and the algorithm can be performed as before.

□

5.3 Analysis of type-E Instances

While it is true that the clipping assumption turns this into a non-crossing instance, if an instance is known to conform to this type, then the solution may be more accessible than the general case.

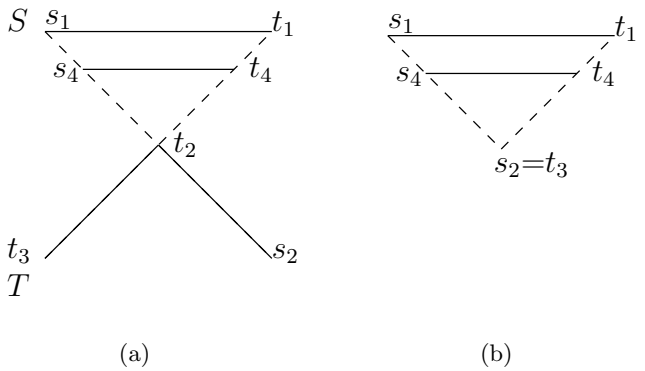


Figure 5.12: An example of a type-E MVPP-n and the result of applying the clipping assumption.

Lemma 28 *A type-E MVPP-n has a schedule under both constaints.*

Proof: We begin with the connected constraint. This type of instance will be reduced to a non-crossing instance with at least two agents at x (as shown in Figure 5.12(b)). Since these agents can see along both S and T, they can see every point on all the paths (by the triangle lemma). Thus the next step is to move all other agents directly to their targets. This completes the connected constraint schedule.

Now we discuss the schedule for the complete constraint. Move the bounding agent along the bounding path (this may be Π_1 or it may include a split bounding path) one segment at a time. Move the internal agents as far along their paths as possible without crossing the sightline between the bounding agent and x. Since all the agents will always be in a triangle formed by x and the endpoints of the segment of the bounding path all points will be mutually visible. This gives the schedule for the complete constraint.

□

5.4 Discussion

We do not know whether there are instances in type-A which have no solution to the linear inequalities given in Section 5.1.4. A lot of work on solving inequalities seems to be geared towards optimisation problems. Typically these would be solved using the simplex method [Had94] to find an extreme point on the boundary of the solution region. In our case, however, a point in the *interior* of the region is desirable since such a point would not have the extra derivative constraints. This raises a question as to whether the derivative constraints actually restrict solutions or not. The fact that we want an interior point rules out the weak inequalities used by the simplex method. However, the solution region (if it exists) must be convex, so the average of a set of vertices in the region must also lie in the region. Further, if the vertices do not all lie on the same face, then the average will lie in the interior of the region. So, calculating the average of a vertex and its immediate neighbours will give a point in the interior (provided the interior exists). If the solution region was flat, then the average would still be on the border.

Chapter 6

Collinear Endpoints — the Crossing Case, Type-C Instances

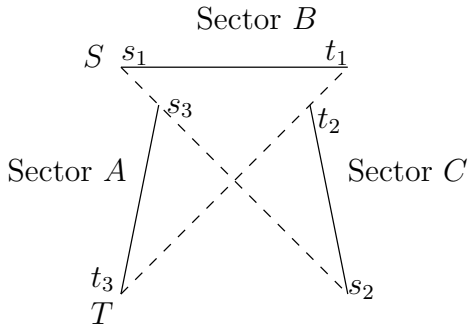


Figure 6.1: A type-C instance.

In this type there are 3 non-trivially occupied sectors labelled A, B, C with the empty sector lying between A and C. As with Chapters 5 and 7 we do not give the computational complexity of constructing schedules. The start points for sectors A and B lie on one side of x while the startpoints for sector C lie on the other side of x. Type-C (and type-D) instances require us to consider an additional challenge.

Definition 30 *The Starting and Finishing tasks are said to* interfere*, if starting (under a given procedure) requires moving agents past the positions for the Finishing task. Interference does not imply that a particular instance is insolvable for the given paths. It just means that the current procedure does not produce a schedule.*

Definition 31 *The* forward projection *of a segment onto a polyline is the intersection of the polyline with a ray along the segment extending through the end of the segment but not its beginning. The forward projection of a point on a path onto a polyline is the forward projection of the path-segment the point lies on. In the case of vertices joining two path-segments, the later segment is used.*

A symmetric definition gives back projection*. The back projection of a segment onto a polyline is the intersection of the polyline with a ray along the segment extending through the beginning of the segment but not its end. The back projection of a point on a path onto a polyline is the back projection of the path-segment the point lies on. In the case of vertices joining two path-segments, the earlier segment is used.*

Given this definition, a projection onto a general polyline may not be a point and may not even be connected. With this in mind, when we talk about not moving past a projection we mean not moving past the last point in the first connected part of the projection. Moving to a projection means moving to the first point in the projection. However, in this chapter and Chapter 7 we will be projecting between bounding paths in adjacent sectors. Since these paths will be x-restricted, each projection for each bounding path can only be onto one sector, for example points on τ_A can only forward-project onto τ_D. For this reason we may omit the target of the projection.

By application of the clipping assumption, we know that s_B and s_C have non-trivial visibility of each other. The visibility questions which remain are $s_A \rightarrow s_B$, $s_B \rightarrow s_A$, $s_A \rightarrow s_C$, $s_C \rightarrow s_A$ (remember, non-trivial visibility is not necessarily symmetric.)

Lemma 29 shows that ideal solutions are possible under the connected visibility constraint. The remainder of the section discusses the complete visibility case.

Lemma 29 *Any type-C MVPP-n has an ideal solution under the connected constraint.*

Proof: We will use the same basic method for the complete schedule from type-B case for Sector A and Sector B (Lemma 27), and will ensure that an agent from Sector C is always visible to an agent from Sector B. We are not using the connected constraint method from type-B since it relied on having an agent at x. When a_A moves, the remaining agents in Ag_A should move so they lie on the segment $\overline{a_A\, a_B}$. As with the previous cases, we compute τ_C and project the segments forwards onto bounding path for Sector B. These extra points will be treated as vertices of τ_B when producing the schedule for A and B.

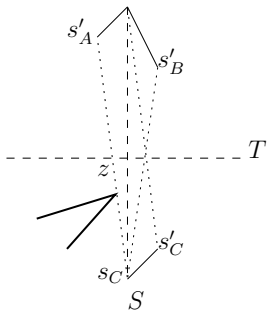


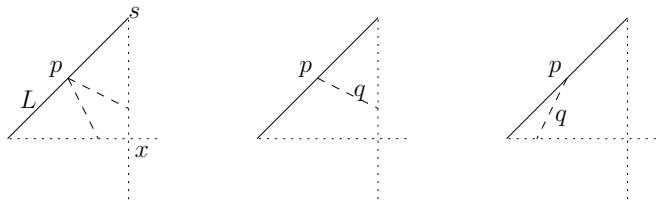
Figure 6.2: Points s_A, s_B, s_C have non-trivial visibility of each other. Thick lines indicate polygon boundary.

A schedule can be produced for Sector C and Sector B by projecting from τ_B onto τ_C and applying the same method in reverse. That is, producing a schedule for moving from T to S. This schedule should then be reversed. The two schedules can be combined as follows. The agents in sectors A and B move as per their schedule but whenever a_B moves, a_C should move to the forward projection of a_B's current segment. The remaining agents in Ag_C move to the line between a_B and a_C. If an agent reaches its target before the bounding agents, it can be ignored for the remainder of the schedule since the triangle lemma guarantees visibility from then on. Once a_B or a_C reaches its target then the triangle lemma ensures that the agents in the other sector can be removed directly to their targets. Visibility is guaranteed since a_A and a_C are always visible to a_B and the remaining agents always see the bounding agent for their sector. This gives the schedule.
□

Now we need to define a point we will use throughout this section. Consider a segment from s_C to a point p on τ_A such that the segment is a sightline and p is as far as possible along the path without losing visibility. We define point z as the point of intersection between the segment and T. See Figure 6.2 for an example. If s_C has trivial visibility of s_A, then $z = x$.

Definition 32 *A* convex arrangement *for a type-C MVPP-n means that all agents lie on their paths, that points a_A, z, a_C and a_B form a convex figure and the remaining agents are either at their targets or lie on segments $\overline{a_A\, z}$, $\overline{a_B\, a_C}$. None of the bounding agents can be at their start or target points. See Figure 6.5(d) (page 120) for an example.*

In this chapter and the next we wish to determine whether a sightline between bounding agents can cut the bounding paths. This is to avoid having convex arrangements enclosing



(a) (b) (c)

Figure 6.3: An illustration of symbols from Lemma 30

any part of a bounding path. As part of this check, we need the following lemma. The symbols used in this lemma are illustrated in Figure 6.3.

Lemma 30 *Let p be a point on an x-restricted path π from s to t. Let L be the line through the line segment $\overline{s\,p}$ and $\overline{L_2}$ be a line segment from p to a point between s and x on S or a point between x and the endpoint of L on T. The segment $\overline{L_2}$ cannot cross π.*

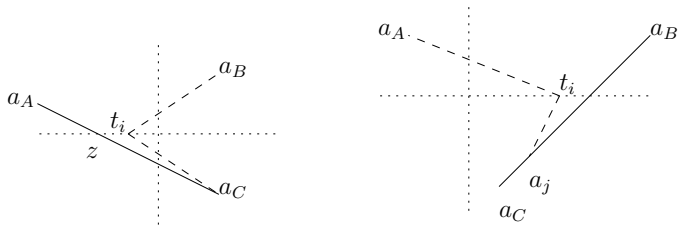
Proof: Without loss of generality assume p is in Sector A. We proceed by contradiction. Suppose $\overline{L_2}$ crosses π at point $q \neq p$. There are two cases:

1. Segment $\overline{L_2}$ is vertical or slopes right of vertical (see Figure 6.3(b));
2. Segment $\overline{L_2}$ slopes left of vertical (see Figure 6.3(c)).

First suppose q comes before p on π. In both cases this means π has turned too far clockwise and will need to make a non-convex turn in order to get back to T. If q comes after p on π, then the first case means π has turned too far, whereas the second case requires an anticlockwise turn.

□

Apart from a few special cases, our overall approach is to determine convex arrangements produced by the Starting and Finishing tasks. We then produce a schedule which moves from one arrangement to the other by moving one bounding agent at a time (that is, the Connecting task). If the starting and finishing arrangements *overlap* (positions on the starting arrangement lie after those for the finishing arrangement) we will have *interference*. In Section 6.1, we will show how to start an instance where all start points have non-trivial visibility of each other and $s_A = s_B$. Starting/finishing for more complicated cases is discussed in Section 6.2.



(a) (b)

Figure 6.4: Illustration of an agent a_i reaching t_i before its bounding agent. The first figure shows an agent in Sector A, the second shows Sector B or Sector C.

First however we need to consider what happens when an internal agent reaches its target before the rest of the agents. When this happens, we assume that the agent is removed from the relevant Ag set and is not considered for the rest of the schedule. For this to work though, we need to be sure that once an agent reaches its target, it can see all points on the remaining path for all other agents.

Suppose $a_i \in Ag_A$ reaches t_i. Refer to Figure 6.4(a). The moving agents in Ag_A are arranged on a line through z which slopes downwards with positive direction of the x-axis. This means t_i lies on or to the right of z. So for all moving agents $a_j \in Ag_A$, the segment $\overline{a_j\, t_i}$ cannot cut π_j by Lemma 30 and hence all π_j are t_i-restricted. This means visibility is assured by the triangle lemma. Visibility of agents in Ag_B and Ag_C is also given by the triangle lemma. Since all the paths are x-restricted they must also be t_i restricted.

Now consider an agent a_i in Ag_B or Ag_C reaching target t_i. Refer to Figure 6.4(b). Paths in Sector A are x-restricted so they must be t_i-restricted and the triangle lemma applies. The only way for the path a_j in $Ag_B \cup Ag_C$ could not be t_i-restricted (by Lemma 30) is if $\overline{t_i\, a_j}$ slopes upwards in Sector C (downwards in Sector B). In that case however, in order to remain x-restricted, parts of π_j left of $\overline{t_i\, a_j}$ must lie in the triangle x, a_j, t_i which is free of obstacles. This, plus the triangle lemma gives the required visibility.

Now we have established this, we will not mention these agents further.

6.1 A Complete Constraint Solution

Now we show how to solve instances where all start points have non-trivial visibility of each other and $s_A = s_B$. Lemma 31 shows how to start (and by symmetry finish) such an

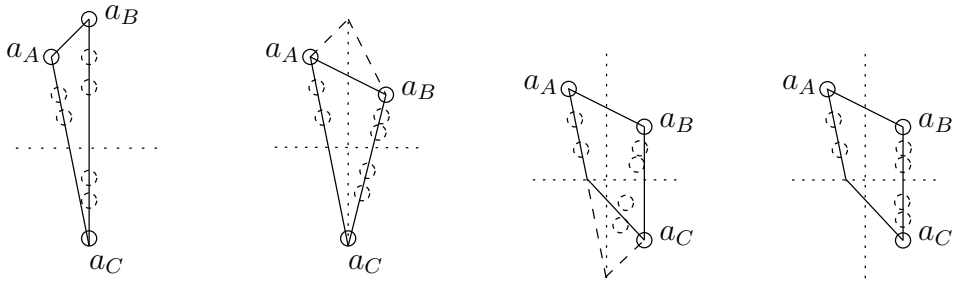


Figure 6.5: Agent positions for the steps in starting a schedule in Lemma 31. Bounding agents are shown as solid circles.

instance. Lemmas 32 and 33 show how to move agents from one convex arrangement to another. Lemma 34 ties the results together.

Lemma 31 *Starting (by producing a convex arrangement with no bounding path in its interior) a type-C MVPP-n where s_A, s_B and s_C all have non-trivial visibility of each other and $s_A = s_B$; is possible under the complete visibility constraint using x-restricted paths.*

Proof: Refer to Figure 6.2 (page 117). The points s'_i denote points on the first segment of the respective bounding paths some small, non-zero distance from the start points. These distances should be such that s'_B should be before(to the left of) the forward projection of s_C and vice versa. Also $\overline{s'_A\, s_C}$ should pass through or to the right of z. This represents a_A moving while remaining visible to s_C.

The first step is to move a_A to s'_A. The initial bounding agents form a triangle with all the other agents inside it so visibility is preserved. Agents in Ag_A should now advance to $\overline{s_C\, s'_A}$. See Figure 6.5(a). Since we know that $\overline{s_C\, s'_B}$ is a sightline and the quadrilateral shown in Figure 6.5(b) is convex, a_B can be moved to s'_B. The remaining agents in Ag_B can be advanced to $\overline{s'_B\, s_C}$ (this also applies to Ag_C).

Now we wish to make the move shown in Figure 6.5(c) but we need to be sure that $\overline{s'_C\, s'_B}$ does not cut a bounding path. Note that since s'_B and s'_C lie before the forward projections of their respective start points we can apply Lemma 30 to show that $\overline{s'_B\, s'_C}$ cannot cut τ_B or τ_C. Finally, we advance the agents in Ag_C to $\overline{s'_C\, s'_B}$ and we have the situation shown in Figure 6.5(d). If agents in Ag_A do not lie on $\overline{a_A\, z}$, then they can be safely moved there. We

have moved all agents (other than those at x) into a convex arrangement (and hence off S) and the visibility constraint is preserved, so we are done.

□

The following lemma shows that it is always possible to move at least one bounding agent by following certain conditions listed in the lemma. These conditions do not mean that these are the only circumstances under which movement is possible. However, we show that moving under these conditions is sufficient. Other moves may be possible, for example, if the bounding paths are not shortest.

Lemma 32 *For a complete constraint type-C MVPP-n which is in a convex arrangement with no bounding path in its interior, it is always possible to move at least one of the bounding agents. The shape formed by the bounding agents after the move, will be convex and have no bounding path in its interior.*

Proof: First we show how to move the agents and when it is safe to do so. When we talk of safety here, we mean that the sightlines between bounding agents will not cut bounding paths. This ensures that the sightlines cannot cut obstacles. Later we will show that one agent must be movable. The general approach is to move one of the bounding agents, then move the remaining agents to the boundary of the convex shape delineated by them. This forms a new convex arrangement, unless one of the bounding agents reaches its target.

If a_A lies above ${}^{ext}\overline{a_C\, z}$, then it is safe to move a_A from the point of view of a_C, provided a_A pauses at each vertex to allow the other agents in Ag_A to advance. We also need to ensure that $\overline{a_A\, a_B}$ remains a sightline. This is all we need to do because a_A and a_B will always be the highest agents in their respective sectors. Provided the back projection of the segment a_A will move along lies above (or through) a_B, we are safe. See Figure 6.6(a) (page 122) for an illustration.

Agent a_B should not move past the forward projection of a_C. We also need to ensure that a_B does not move out of sight of a_A. Consider the back projection of the segment which a_B moves along. Provided that the segment passes above (or through) a_A we are safe. See Figure 6.6(b).

For sightlines relevant to the movement of a_C see Figure 6.8. Agent a_C cannot move past the forward projection of a_B. The sightlines from agents in Ag_A to a_C cannot be cut from below since that would require agents in Ag_C to have moved further away from T (their paths

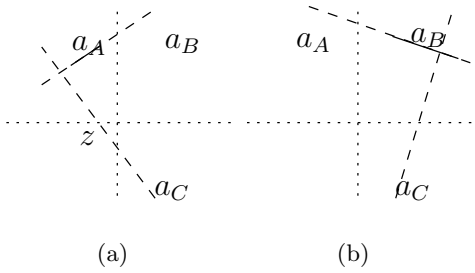


Figure 6.6: Illustrations of safe movement constraints.

were not shortest). We also need to show that the sightlines from agents in Ag_A are not cut from above. In the figure this appears to be impossible but consider the case where a_B lies on $\overline{a_A\, a_C}$. So, do not move a_C past the line through $\overline{a_A\, a_B}$.

Now we show that at least one of these moves is always possible. Suppose we cannot move any of the bounding agents. Not being able to move a_C implies at least one of the following:

- Agent a_C is about to move past the forward projection of a_B onto τ_C.

  This could be fixed by attempting to advance a_B but we assume we cannot do this either. To prevent a_B from moving, requires that a_B is about to move past the forward projection of a_C or there is a problem with a_A. The first (a_B and a_C blocking each other) cannot happen without a situation such as that shown in Figure 6.9 and such paths would not be x-restricted. So a_B must be about to move onto a segment with a back projection below a_A. This situation (shown in Figure 6.7) leads to a contradiction because the figure formed by z, a_A, a_B, a_C cannot be convex.

- Agent a_C is about to move out of sight of a_A.

  The only restriction on a_C from a_A is that a_C should not move past the line through $\overline{a_A\, a_B}$ so a_A, a_B and a_C must be collinear. So we must be able to move a_A, which is a contradiction.

□

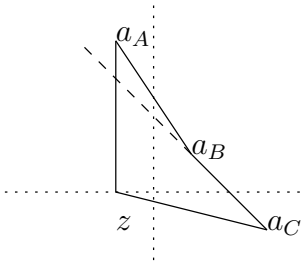


Figure 6.7: Agent a_C is about to move past the forward projection of a_B onto τ_C and a_B can't move.

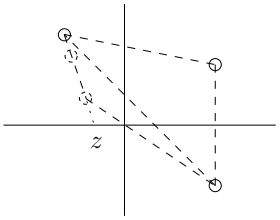


Figure 6.8: Sightlines relevant to the movement of a_C in Lemma 32.

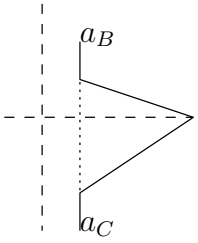
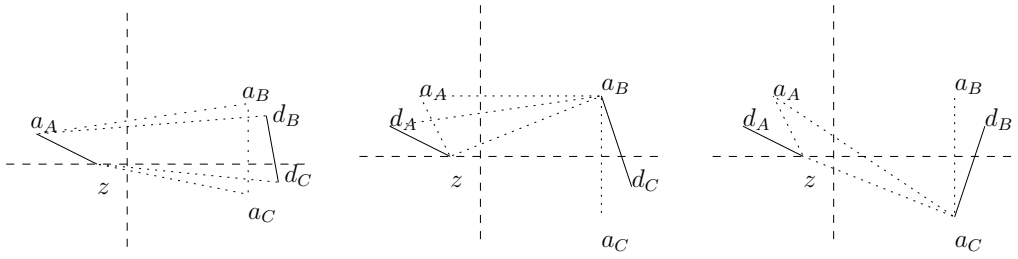
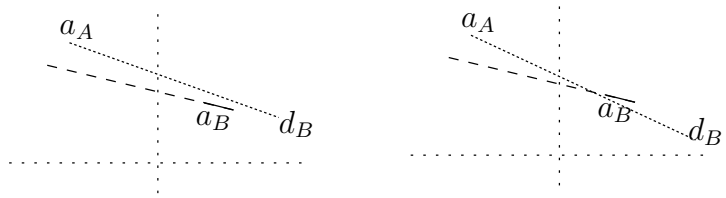


Figure 6.9: A case where forward projections (shown dotted) are equal.



(a) (b) (c)

Figure 6.10: Effect of an agent finishing before the others.



(a) (b)

Figure 6.11: The position of segment $\overline{d_A\, d_B}$ relative to a_B. The back projection of the segment a_B moves on is shown dashed.

Lemma 33 *Consider a type-C MVPP-n and two convex arrangements such that all the bounding agents in the second one are further along their paths than those in the first one. Further, no points on any of the bounding paths lie in the interior of either arrangement. A schedule exists to move the agents along their paths from the first arrangement to the second.*

Proof: We already know that movement is possible, if none of the agents are held in place (Lemma 32). We need to show that an agent being held in place because it has reached its destination does block other agents. We also need to show that sufficient progress is made for all agents to reach their destinations. First, we consider blocking.

Suppose a_A is at d_A. See Figure 6.10(a). Agent a_A could only block a_B if the back projection of the segment a_B was about to move on, ran below d_A. This gives two possibilities, either $\overline{d_A\, d_B}$ passes above a_B or it passes through or below a_B. If $\overline{d_A\, d_B}$ is above a_B (see Figure 6.11(a)), then the destination arrangement contains part of τ_B and we have a contradiction. If $\overline{d_A\, d_B}$ runs through or below a_B (see Figure 6.11(b)), then τ_B must turn

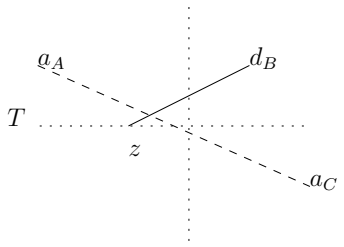


Figure 6.12: The segment $\overline{a_A\, a_C}$ must cut $\overline{z\, d_B}$ once a_B has reached d_B.

back towards S (clockwise in the figure) in order to reach t_B. This is not possible for an x-restricted path. Further, since a_A cannot block a_C, agent a_A arriving at its destination does not cause a problem.

Suppose a_B is at d_B. See Figure 6.10(b). Agent a_B at its destination cannot block a_A by the same reasoning as above. For a_C to be blocked by a_B, the segment $\overline{d_B\, d_C}$ would need to cut τ_B which is a contradiction.

Suppose a_C is at d_C. See Figure 6.10(c). Agent a_C at its destination cannot block a_B for the same reason that a_B cannot block a_C. For a_C to block a_A, the point d_A would need to lie below ${}^{ext}\overline{d_C\, z}$. This leads to a contradiction because it would make the final shape non-convex (there would be a reflex vertex at z).

We know from Lemma 32 that a move is always possible so it remains for us to show that agents can move far enough to reach their destinations.

Consider the ways in which agents stop moving. They stop when they reach a vertex on their bounding path or a forward or back projection from another agent. However, each of these can only stop an agent once, and hence these conditions give a finite number of stops. The only other cases are a_A about to move past ${}^{ext}\overline{d_C\, z}$ or a_C about to move past ${}^{ext}\overline{a_A\, a_B}$. These conditions could stop a_A or a_C more than once, however there must be a move by another agent before those conditions can stop the agent again. This means that the only case we need to consider is an infinite sequence of movements where a_A moves to ${}^{ext}\overline{d_C\, z}$ and then a_C moves to ${}^{ext}\overline{a_A\, a_B}$. We will show this cannot happen using proof by contradiction.

Suppose there is such an infinite sequence of moves. There must be a term in the sequence after which there are only moves by a_A and a_C. The segment $\overline{a_A\, a_C}$ would then always intersect the segment $\overline{z\, d_B}$ (see Figure 6.12). Choose agent a_i to be a_A if $\overline{a_A\, a_C}$ passes through z. Otherwise choose a_i should be a_C. Move a_i backwards along its bounding path

until it intersects $\overline{z\, d_B}$ at one of its endpoints. Consider the change in distance Δ from a_i to T. Now, x-restricted paths are monotonic with respect to distance from T and the gradients of successive segments get monotonically less step relative to T the closer they get to T. This means that Δ is a lower bound on the amount of progress a_i makes towards T.

Finally, since no segment on an x-restricted path can be parallel to T or to S, total progress parallel to T is also bounded below (by a symmetric argument from the start of the instance rather than its end). So, together all the movement restrictions can only require a finite sequence of moves.

□

In order to produce a schedule using Lemma 31 and Lemma 33, we need to ensure that none of the agents' positions in the finishing arrangement are earlier in their paths than the positions from the starting arrangements. It turns out there is only one set of visibility restrictions in which this occurs and it does not affect the current case (that is, $s_A = s_B$ and all start points have non-trivial visibility of each other).

In order for interference to occur, the bounding agent must be forced to move some minimal distance in order for another agent to start moving (as opposed to being able to move arbitrarily small distances in a normal start). Further, this would need to happen both for Starting and Finishing tasks. Note that for finishing purposes the roles of sectors A and C are reversed. In which sectors could such a thing happen?

- Sector A:

  If s_B lies strictly between s_A and x, then a_A will need to move to the back projection of s_B in order to allow a_B to start moving.

  If a simultaneous finish for Sectors A and C is also required, then we have the situation shown in Figure 6.13. Figure 6.21 (page 134) shows combinations of visibility restrictions and how to start (or finish) schedules under those conditions. We will discuss Figure 6.21 in detail later. If we have the situation in row four of Figure 6.21, then a_C will need to move to keep a_A visible. This can also happen if sector D is not blocked and a finishing procedure as shown in row five of Figure 6.21 is required.

  In any case, provided the sightlines between a_A and a_C at the completion of the Starting task and the beginning of the Finishing task are chosen such that they intersect at x there is no overlap in the arrangements.

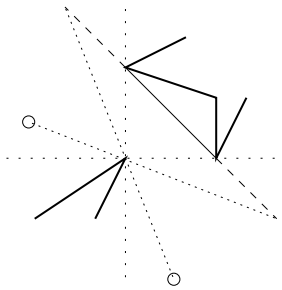


Figure 6.13: Possible positions for a_A and a_C (shown as circles) which allow the opposite agent to move far enough to let a_B start (and finish). Back projections of s_B and t_B are shown dashed, dense dots show the sightlines to the positions closest to S and T for a_A and a_C.

- Sector B:

  If s_A lies strictly between s_B and x, then a_B will need to move to the back projection of s_A in order to allow a_A to start moving. A symmetric case arises with finishing and t_C. An example is shown in Figure 6.15 (page 129). So sector B could exhibit overlapping arrangements.

  This particular case is different from the others because it may not be possible to start without moving a bounding agent to its target. However, Lemma 35 shows how to produce a complete schedule in such cases.

- Sector C:

  This case is symmetric with Sector A so it cannot exhibit overlapping arrangements either.

Since this issue does not affect our current case we defer discussion of possible overlapping arrangements until the next section.

Lemma 34 *Consider a type-C MVPP-n where all start points have non-trivial visibility of each other; the target points have non-trivial visibility of each other;* $s_A = s_B$ *and* $t_B = t_C$. *Such an instance has a schedule using shortest paths.*

Proof: Lemma 31 (Starting task) allows us to move the agents to a convex arrangement. Using the same lemma in reverse on the target points gives a convex arrangement which we

	s_A	s_B	s_C
s_A		T	
s_B	?		
s_C	?		

(a) Blocking s_B would block s_C as well.

	s_A	s_B	s_C
s_A			T
s_B	?		
s_C	?		

(b) Blocking s_C would block s_B as well.

	s_A	s_B	s_C
s_A		T	T
s_B	T		
s_C	?		

(c) At least one of s_A, s_B must have non-trivial visibility of the other.

Figure 6.14: Illegal combinations of visibility restrictions. The table cells indicate the type of visibility the row label has of the column label. For example the cell in the top right corner indicates the visibility s_A has of s_C. A cell containing 'T' indicates trivial visibility while a blank cell indicates non-trivial visibility. Cells containing ? do not affect the legality of the arrangement.

can finish from. Finally, Lemma 33 tells us that a schedule exists for moving between the two arrangements.

□

Now we need to consider cases where visibility is not so favourable.

6.2 Remaining Cases

First note that s_C and s_B must always have non-trivial visibility of each other. Most legal visibility combinations and how to start them are shown in Figure 6.21 (page 134). Lemma 36 discusses the case which requires a simultaneous start. Illegal visibility combinations are shown in Figure 6.14. We will discuss the special cases first, beginning with Figure 6.15.

Lemma 35 *Consider a type-C MVPP-n where s_A lies strictly between s_B and x and from the finishing point of view, t_C lies strictly between t_B and x. Further, τ_B intersects the forward projection of the last segment of τ_C before it intersects the back projection of the first segment of τ_A. In such a case there is a schedule using x-restricted paths.*

Proof: Refer to Figure 6.15. The overall approach is to move the agents in Ag_B and Ag_C while leaving the agents in Ag_A at their starting positions until a_C reaches its target. Only once this is done will the agents in Ag_A move.

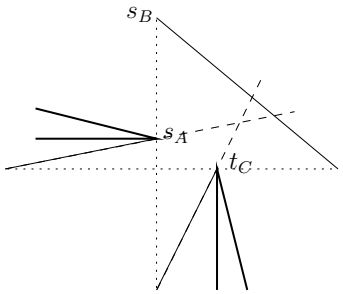


Figure 6.15: An instance where the start and finishing arrangements cross over.

Note that, since the back projection of s_A slopes upward, the section of τ_B between s_B and its intersection with the projection must be s_A-restricted. This is because there is no way an x-restricted path could cut the projection and then intersect it again later. We need to show that the other agents in Ag_A can see the agents in Ag_B. This will be ensured by having the agents in $Ag_B \cup Ag_C$ always in the convex figure defined by s_A, a_B, a_C, x. Also note that since $\overline{s_A\, t_C}$ lies between the two projections of s_A and τ_C, it must be a sightline. Hence τ_C (and $\pi_i\ \forall i \in Ag_C$) are s_j-restricted $\forall a_j \in Ag_A$. Also, since the back projection of s_A cannot slope downwards, the part of τ_B leading up to the projection must be s_A-restricted. Hence it is also s_j-restricted $\forall a_j \in Ag_A$.

As described in previous lemmas, at each vertex on the bounding paths the bounding agent pauses to allow the internal agents to advance to $\overline{x\, a_C}$ or $\overline{a_C\, a_B}$ as appropriate. Move a_C to the forward projection of s_B then move a_B to its next vertex. Repeat until a_C reaches t_C.

Once this is done, we have a situation such as the one shown in Figure 6.16 (page 130). Move a_B to the back projection of s_A and then advance the internal agents. Advance a_A one segment and move the internal agents to $\overline{a_A\, x}$. Agents a_A and a_B should then be moved in turn, pausing when they reach the back projection of the other agent. When one of them reaches its target, the other agent can move to its target.

This concludes the schedule.

□

Now we consider an instance which requires a simultaneous start.

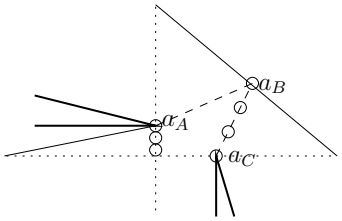
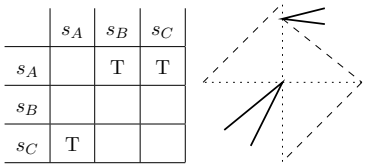
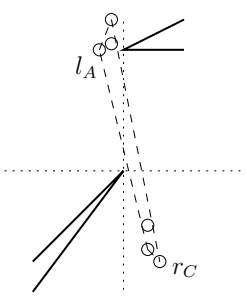


Figure 6.16: See Lemma 35. All the agents in Ag_C have finished and agents in Ag_A have not started yet.

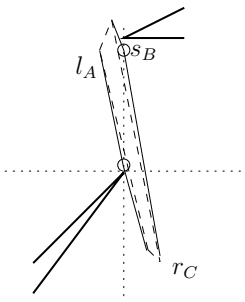


	s_A	s_B	s_C
s_A		T	T
s_B			
s_C	T		

Figure 6.17: This case requires a simultaneous start. See Lemma 36



(a) Agents after a simultaneous start in sectors A and C. Dashed lines indicate the convex hull of $Ag_A \cup Ag_C$.



(b) Extending the hull to include Ag_B. Dashed lines indicate the convex hull of $Ag_A \cup Ag_C$.

Figure 6.18: Step 1 in Lemma 36 — hull H is extended to H_2 by including s_B.

Lemma 36 *Consider a type-C MVPP-n which cannot start without simultaneous movement in Sector A and Sector C and where the starting problem for Sector A and Sector C has a solution using x-restricted paths. There is a schedule to move agents in all three sectors to a convex arrangement with no bounding path in its interior.*

Proof: Note that s_A must have trivial visibility of s_B in order for s_C's vision to be obstructed. See Figure 6.17 for an illustration.

We move the agents into a convex arrangement using the following steps.

1. Start agents in $Ag_A \cup Ag_C$ simultaneously.
2. Move the agents in Ag_A so they are all on a single line through x.
3. Move the agents in Ag_C so they are all on a single line through s_B and r_C.
4. Move agents in Ag_A (and if necessary, Ag_C) to allow a_A to reach the back projection of s_B.
5. Move agents in Ag_B forward a small distance, forming a convex arrangement with no bounding path in its interior.

To ensure visibility for all agents, we will ensure that all agents are contained within a convex figure (which contains no obstacles). However, the restriction on not containing bounding path is not required until the final step.

Step 1: *Start agents in $Ag_A \cup Ag_C$ simultaneously.* Since the distance travelled in a simultaneous start can be chosen to be arbitrarily small, we assume a_A is still above the back projection of s_B and no agent in Ag_C is on or past the forward projection of s_B. We need to ensure that visibility is preserved for agents in Ag_B while agents in $Ag_A \cup Ag_C$ move. Consider the convex hull H of $Ag_A \cup Ag_C \cup \{x\}$ and refer to Figure 6.18(a). The simultaneous start procedure ensures that $Ag_A \cup Ag_C$ are mutually visible (so their hull is free of obstacles). Lines from x to agents in $Ag_A \cup Ag_C$ cannot be broken by obstacles because all paths must be x-restricted and hence all points are visible to x by the triangle lemma. Let l_A be the endpoint of the left edge of H in Sector A. Now consider the convex hull H_2 of $H \cup Ag_B$. See Figure 6.18(b). We need to show that H_2 contains no obstacles. If s_B is inside H, then $H_2 = H$ and we have no obstacles. If s_B is not inside H, then consider a ray starting at a_A along the border of H_2 to the right. This ray cannot be below $\overline{a_A\, s_B}$ and so cuts S and

not T. By Lemma 30, the ray cannot be crossed by τ_A and so no obstacle can enter H_2 in Sector A. Since no agent in Ag_C is past the forward projection of s_B no bounding path (and hence no obstacle) can enter H_2 in Sector B. The segment joining s_B to H in Sector C has the wrong slope (Lemma 30) to be cut by τ_C. So, H_2 is free of obstacles.

Step 2: *Move the agents in Ag_A so they are all on a single line through x.* We cannot simply add the section of τ_A to H_2 because the new shape might not be convex. See Figure 6.19(a). Instead, move all agents in Ag_A above $\overline{a_A\, x}$ along their paths to $\overline{a_A\, x}$. This movement lies within H_2, otherwise the paths would need to move closer to S or further from T — neither of which is allowed for an x-restricted path. Hence, there are no visibility problems. See Figure 6.19(b).

Next we will walk a_A along τ_A until it reaches ${}^{ext}\overline{l_A\, x}$. Let p be the vertex on τ_A following a_A. See Figure 6.19(c). Consider a new shape H_3 defined by x, l_A, p, a_A and the rest of H_2 in Sector B and Sector C. Since the section of τ_A we are discussing (from a_A to p) cannot cross ${}^{ext}\overline{a_A\, s_B}$ (doing so would require τ_A to move closer to S and/or further from T which cannot happen for x-restricted paths) or ${}^{ext}\overline{l_A\, x}$, H_3 is convex and contains all the agents. Move any agents in Ag_A between $\overline{a_A\, x}$ and $\overline{p\, x}$ to lie on $\overline{p\, x}$. Continue with this process, advancing p and a_A along τ_A until all agents lie on ${}^{ext}\overline{l_A\, x}$. See Figure 6.19(d).

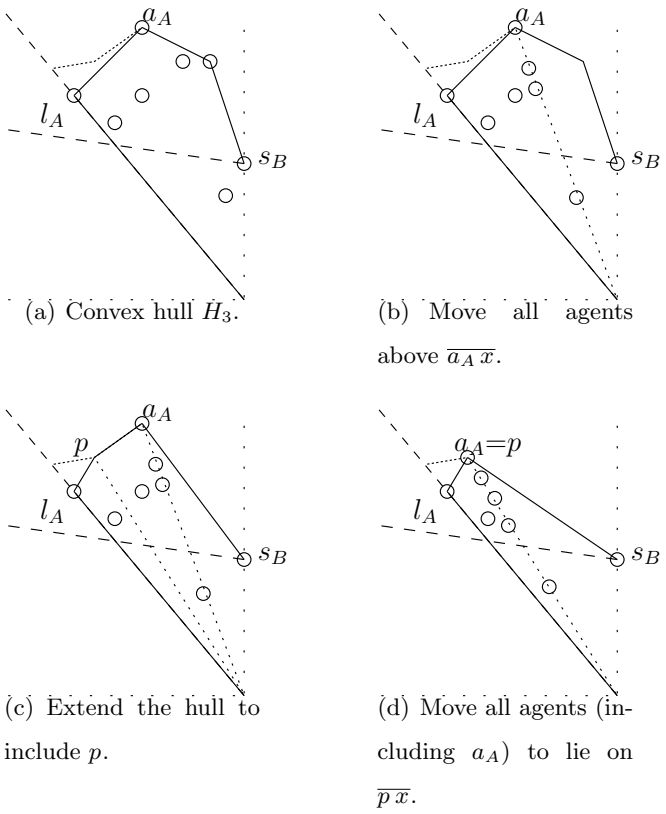
Step 3: *Move the agents in Ag_C so they are all on a single line through s_B and r_C.* Let v be the intersection point of ${}^{ext}\overline{x\, r_C}$ with τ_C. A symmetric procedure to that in Step 2 can move the agents in Ag_C to $\overline{x\, v}$ (excluding any agents which are already in front of it). From there, all the agents can be moved to lie on ${}^{ext}\overline{s_B\, r_C}$.

Step 4: *Move agents in Ag_A (and if necessary, Ag_C) to allow a_A to reach the back projection of s_B.* See Figure 6.20 (page 134) for an illustration at this stage. The next step is to move a_A to the back projection of s_B, by performing the following:

1. If $\overline{a_A\, a_C}$ is above x, move agents in Ag_A forward until $\overline{a_A\, a_C}$ passes through x or a_A has reached the projection.

2. If a_A has not reached the back projection of a_B, then move a_C forward one segment, or until it reaches the line through $\overline{a_A\, a_B}$ or the forward projection of a_B, whichever comes first.

Repeat these steps until a_A reaches the back projection of a_B.



(a) Convex hull H_3.

(b) Move all agents above $\overline{a_A\, x}$.

(c) Extend the hull to include p.

(d) Move all agents (including a_A) to lie on $\overline{p\, x}$.

Figure 6.19: Operations in Step 2 of Lemma 36. Dashed lines show the extension of $\overline{l_A\, x}$ and the back projection of s_B, solid lines show the hull H_2, dense dotted lines show part of τ_A. Sparse dotted lines show the sweep lines.

Step 5: *Move agents in Ag_B forward a small distance, forming a convex arrangement with no bounding path in its interior.* Finally, move a_B some small distance along its path and move the agents in $Ag_B \cup Ag_C$ to $\overline{a_B\, a_C}$. This produces the convex arrangement.

Now we show that the arrangement has no bounding path in its interior. Notice that $\overline{a_A\, s_B}$ slopes downwards and hence cannot be crossed by the bounding path (Lemma 30). Since a_C will not move past the forward projection of s_B segments $\overline{a_A\, s_B}$ and $\overline{s_B\, a_C}$ cannot be crossed either.

□

Now we discuss the cases shown in Figure 6.21 (page 134). In order to follow the pattern of Lemma 34, we need to describe the start procedures and show that they do not produce arrangements with bounding path in their interior. The ways in which a bounding path could be in the interior of an arrangement are shown in Figure 6.22.

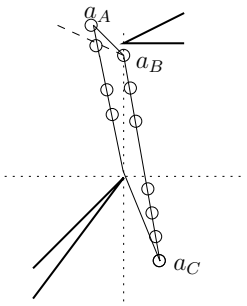


Figure 6.20: The beginning of Step 4 in Lemma 36. All the agents are arranged on two lines but a_A has not reached the back projection of a_B (shown dashed) yet.

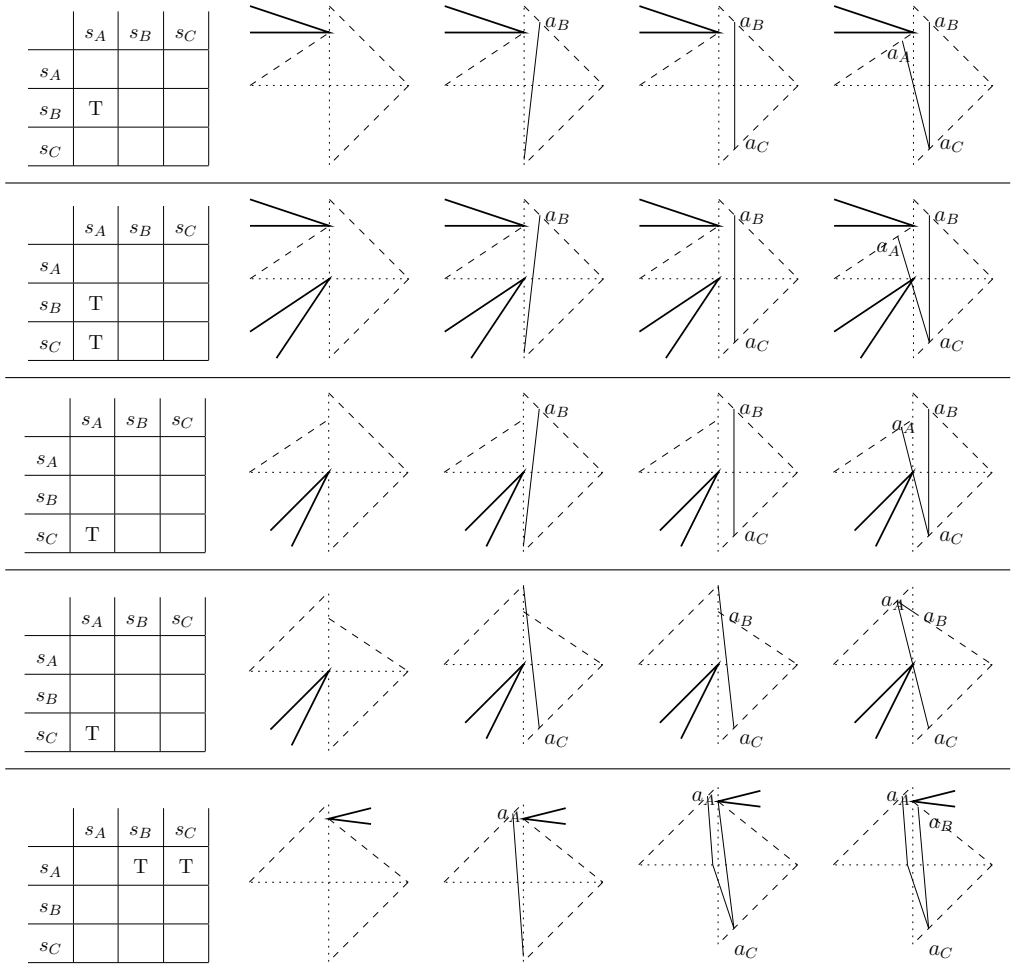


Figure 6.21: Combinations of visibility restrictions and how to start under them.

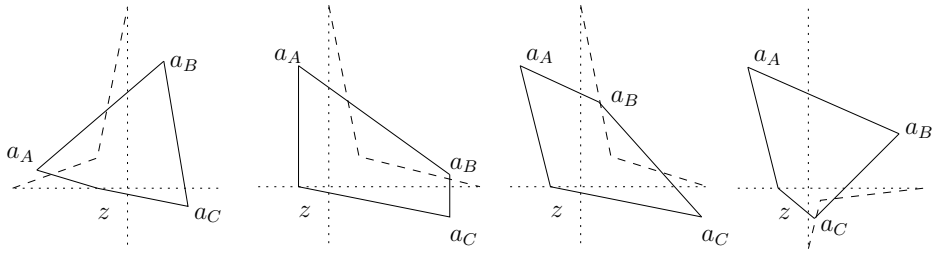


Figure 6.22: Ways in which a bounding path could appear in the interior of a convex arrangement.

Lemma 37 *For all instances of the form shown in Figure 6.21, there exist schedules to move all agents into a convex arrangement (or to their target points).*

Proof: Consider the first three rows of Figure 6.21. In those cases, we want to move a_B to the back projection of s_A in order to ensure a_A can move. This is not strictly necessary in the instance shown in row three but doing so will ensure that s_A is not in the interior of the arrangement. We assume that the case described in Lemma 35 does not apply here since we have already shown how to deal with that case.

The steps are as follows. Agent a_B should move to the back projection of s_A or to the forward projection of a_C. If a_B has not reached the projection yet, move a_C to its next vertex and repeat (a_B will be on the forward projection of a_C so a_C can move at least one vertex safely). Since $\overline{a_C\, x}$ cannot be cut (τ_C is x-restricted) and $\overline{s_A\, a_B}$ cannot be crossed by τ_B (it slopes upwards), all the agents will be in a convex figure and mutually visible with no bounding path inside it.

Row four requires different handling. If it is treated as if there is an obstacle between s_A and s_B, then it looks like we require a simultaneous start. However, it is not quite the same, since we can allow the sightline $\overline{a_A\, a_C}$ to cut τ_B as long as we remove all cuts before the Starting task is over. The goal is to move a_C into a position in which it can see the back projection of s_B onto τ_A. The position we want for a_C is such that the line from a_C to the back projection of s_B onto τ_A passes through x. Such a position must be before the forward projection of s_B onto τ_C since $\overline{a_A\, a_C}$ must be above $\overline{a_C\, x}$ in sector A. Further, since a_C will be before the forward projection of s_B, segment $\overline{s_B\, a_C}$ lies between S and the first segment of τ_B. Hence, there will be no problems with a_C losing sight of s_B or with parts of τ_B being in the interior of the arrangement because τ_B is convex. To be sure that a_C does not move out

of sight of a_A on the way to its position, a_C and a_A should alternate movements so that the sightline between them always (once a_A has made its first move) intersects S between x and s_B. This will require calculating backwards from the positions we want the agents to end at. Once a_A reaches the back projection, a_B can be moved some small distance, the remaining agents updated and we are done.

Now consider row five. Agent a_A must be able to move some distance along τ_A. If this is not enough to reach the back projection of s_B, then a_C must be moved to provide a_A with more room. Agent a_C should move to its next vertex or the forward projection of s_B, whichever comes first. Agent a_A can then move again. Repeat as necessary. Since the bounding agents will form a convex figure with x and at each step the remaining agents should be moved to lie on the edges there are no visibility issues. Once a_A reaches the back projection, a_B can be moved some small distance. If a_C has not moved yet, then it can be moved forward some small distance. The remaining agents can then be moved to lie on the border of the arrangement and we are done.

□

Lemma 38 *A type-C MVPP-n, which does not require a simultaneous start or finish, has an ideal solution.*

Proof: Lemma 35 showed how to deal with cases where the Starting and Finishing tasks interfere, so we will ignore those cases. We have shown how to start and finish any such instance using any x-restricted paths (Lemma 37) and Lemma 33 (the Connecting task) also works with x-restricted paths. Since shortest paths are x-restricted a schedule must exist.

□

Lemma 39 *A type-C MVPP-n which requires a simultaneous start (and/or finish) has a solution.*

Proof: If either the Starting or Finishing tasks do not require simultaneous moves then Lemma 37 shows hows to produce a schedule for the task. We know from the analysis of type-A that there will always be a solution to the Starting (or Finishing) problem for Sector A and Sector C even if it is not close to ideal. From Lemma 36, this can be converted into a solution for sector B as well. Lemma 33 shows how to produce a schedule for the Connecting task.

Note that if the subproblems for Sector A and Sector C admit a solution which approximates the ideal then so does the type-C problem.

□

So in summary, the difficulty lies in finding solutions to the simultaneous starting problem from type-A. Apart from that and some careful positioning of the agents so that sightlines run through x in poor visibility cases, schedules exist for x-restricted (including shortest) paths.

Chapter 7

Collinear Endpoints — the Crossing Case, Type-D Instances

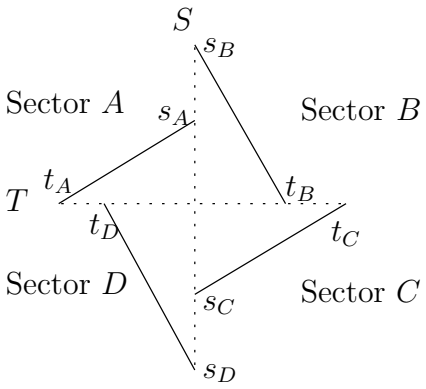


Figure 7.1: A type-D MVPP-n. The start and target sightlines (S and T) are shown dotted.

This chapter examines MVPP-n instances with paths through the interiors of all four sectors. As with previous chapters, we give proofs of the existence of schedules using x-restricted paths but do not give complexities for producing those schedules. Unlike Chapter 6, however, we do not consider instances where the Starting and Finishing tasks interfere under the complete vision constraint.

The sectors for type-D instances are labelled as shown in Figure 7.1. In contrast to Figure 5.1, we will assume that all instances in this chapter are oriented so the target sightline (T) is horizontal. The set of agents which travel through the interior of sector K is denoted Ag_K. So $Ag_B \cup Ag_C$ denotes all the agents which move through Sector B or Sector C. The

bounding path τ_K for Sector K has endpoints s_K and t_K. Recall that s_K and t_K might not belong to the same path.

Definition 33 *Two bounding agents are* neighbouring *if they belong to sectors which are immediately clockwise or anti-clockwise of each other.*

Lemma 40 *A type-D MVPP-n has an ideal solution under the connected constraint.*

Proof: We employ the same method as in Lemma 29. Use that procedure to produce a schedule for sectors A, B and C. The schedule can then be extended to include the agents in sector D in the same way as was done for sector C. Forward project the vertices of τ_D onto τ_A. When a_A reaches one of these projected vertices pause the other sectors until a_D moves to the forward projection of a_A or the next vertex. Whenever a_D moves, the rest of the agents in Ag_D should move to a line through a_D and parallel to T. If a_A reaches its target before a_D does, advance agents in Ag_D to their targets.

This gives the schedule. Visibility is guaranteed by the fact that each sector's agents can always see the bounding agent for their sector and the bounding agents can always see each other.

□

Since the method in Lemma 40 does not have separate Starting and Finishing tasks, interference is not an issue.

7.1 The Complete Constraint

Type-D instances are actually less complex to describe in some cases than type-C because of the high level of symmetry in this type. We will prove some results relative to only one sector. The results can be generalised by relabelling the agents. However, some care should be taken when doing this, since a simple rotation produces incorrect results. Instead, the symmetries are reflectional, so swapping $A \leftrightarrow C$, $B \leftrightarrow D$ and/or $A \leftrightarrow B$, $C \leftrightarrow D$ will produce the desired results.

Unlike the other types, for this type we will only be discussing instances in which the starting and finishing tasks do not interfere. So, procedures for starting are not guaranteed not to interfere with the finishing task for those paths. Instances where interference occurs are left for future work.

As with Chapter 7, we will not move if the new arrangement would cut a bounding path. This will ensure visibility but also that the arrangement does not contain a bounding path.

Definition 34 *Bounding agent* a_i blocks *bounding agent* a_k *if* a_k *cannot move forward without* $\overline{a_i\, a_k}$ *crossing a bounding path.*

We will now prove a result which is used in a number of proofs. We want to show that, once some of the agents have moved from their start points, the four bounding agents cannot all block each other.

Lemma 41 *Suppose the bounding agents are mutually visible and form a convex shape with no bounding path in its interior. Also suppose that at least two bounding agents have moved from their start points. In this case, at least one of the bounding agents is not blocked.*

Proof: In this proof we will only consider bounding agents, so we will drop the "bounding" qualifier for this argument. Notice that we are not considering other reasons why the agents might be prevented from moving, such as the agent having already reached its intended position.

Since at least two of the agents have moved from their start positions and the agents form a convex figure, it is not possible to have two neighbouring agents (for example a_A and a_B) block each other. This would require the forward projections for the two agents to coincide, which could only happen if both agents were moving towards each other along the sightline between them. This possibility is eliminated by the clipping assumption(as argued in Lemma 32).

We proceed by contradiction. Suppose none of the agents can move because they are blocked by another agent. Without loss of generality, we start with agent a_A, which could be blocked by a_B or a_D. For now, suppose a_B is blocking, that is, a_A is about to move onto a segment which has a back projection below a_B. This block could be resolved if a_B were to move forward so as to lie on (or below) the new back projection. Since we claim that a_B is blocked, it must be blocked by a_A or a_C. Agents a_A and a_B cannot both block the other at the same time (see Figure 6.9 (page 123)), which leaves a_B being blocked by a_C. So a_B must be about to move past the forward projection of a_C. Since a_C cannot move to improve the situation, it must be blocked by a_D. That is, a_C must be about to move onto a segment with a back projection above a_D. Finally, a_D must be blocked by a_A. So a_D must be about to move past the forward projection of a_A.

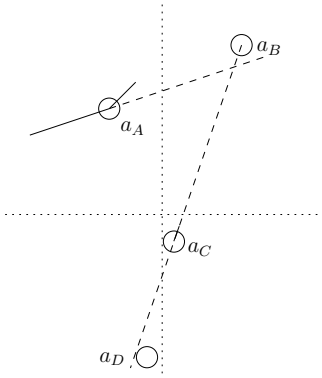


Figure 7.2: A demonstration of Lemma 43.

All this means that the line through a_A's next segment and the line through a_C's next segment must intersect in Sector B. See Figure 7.2. The same two lines must also intersect in Sector D. This is not possible so we have a contradiction.

The situation where a_A is blocked by a_D follows by symmetry.

□

We will also need the following property of back projections.

Lemma 42 *No back projection of any point on an x-restricted path lies on T.*

Proof: Such a projection would come from a segment which was moving away from T. Such a path could not be convex (and hence x-restricted) because to reach a point on T it would need to turn in the wrong direction or self-intersect.

□

Definition 35 *A* convex arrangement *for a type-D MVPP-n means that all agents lie on their paths; a_A, a_B, a_C, a_D form a convex figure and the remaining agents are either at their targets or lie on the segments $\overline{a_A\, a_D}$, $\overline{a_B\, a_C}$. None of the bounding agents can be at their start or target points.*

When moving the agents from one arrangement to another, we will follow the notation from type-C and refer to the positions in the second arrangement as *destinations* rather than targets.

Lemma 43 *Consider a type-D MVPP-n in a convex arrangement where the agents are in one convex arrangement and are moving to another (non-overlapping, non-identical) convex arrangement and where neither of the arrangements encloses a bounding path. It is always possible to move at least one bounding agent.*

Proof: The non-identical constraint ensures that at least one agent is not at its destination yet. First we show that we cannot have a situation where an agent needs to move past the forward projection of an agent already at its destination. Suppose a_I needs to move past the forward projection of a_J. Agent a_J cannot be at its destination because that would result in an arrangement which crosses τ_I or where a_I and a_J cannot see each other. Also remember that as shown in Figure 6.9 (page 123), two bounding agents cannot both block the other at the same time.

Now to give the moves themselves. We know from Lemma 41 that there is at least one bounding agent which can move. We will illustrate using a_A (to generalise reflect through S and/or T as described at the beginning of the chapter). One of the following must be true:

- Agent a_A lies above the back projection of a_B;
- Agent a_A lies above the forward projection of a_D.

Now, a_A can be moved to whichever of those projections is next on τ_A (or to its destination if that is next).

□

Lemma 44 *Consider a type-D MVPP-n and two convex arrangements, such that the bounding agent positions of one are not before the positions of the other arrangement and neither arrangement has a bounding path in its interior. There is a schedule to move between the two arrangements.*

Proof: Lemma 43 ensures that it is always possible to move an agent. We now need to show that, if agents move as far as they are allowed at each step, they will actually reach their destinations. To see this, consider the ways in which an agent can be stopped. Either it has reached a vertex on its path, or it is about to move past the forward projection from another agent. Each forward projection can only stop an agent once and the number of projections is determined by vertices on another path. Since the total number of path vertices is finite, we

	s_A	s_B	s_C	s_D
s_A		4	4	
s_B	3			3
s_C	2			2
s_D		1	1	

Figure 7.3: Visibility restrictions with matching labels must occur together. Empty cells show combinations which must be non-trivial.

must run out eventually and after that there will be nothing to prevent the agents reaching their destinations.

□

Notice that this means that we do not need to explicitly consider the agents arriving at their destinations early as a potential problem. We also cannot have the crab motion (see page 40) for agents a_A and a_C that could occur in type-C instances. We are not excluding instances which require crab motions to start or finish. What we are saying is that the Connecting task between two convex arrangements does not require such motions.

When considering possible starting conditions, we need to note that some combinations cannot actually occur. For example, s_A and s_D must have non-trivial visibility of each other (as must s_B and s_C). Further, if for example, s_A has trivial visibility of s_B then it must also have trivial visibility of s_C. The reverse is also true, trivial visibility of s_C implies trivial visibility of s_B. These pairs of visibility restrictions are shown in Figure 7.3. Some visibility restrictions cannot occur together. The conditions labelled 1 and 2 cannot occur together since this would imply that neither s_C nor s_D have non-trivial visibility of the other. Similarly, the conditions labelled 3 and 4 cannot appear together. Finally, reflectional symmetries swapping $A \leftrightarrow C$, $B \leftrightarrow D$ and $A \leftrightarrow B$, $C \leftrightarrow D$ reduce the number of legal combinations to be considered to those shown in Figure 7.4 (although some combinations have a number of variants).

Now we give the starting procedures for the rows of Figure 7.4. We are not considering interference between the Starting and Finishing tasks in this chapter (and hence not restrict-

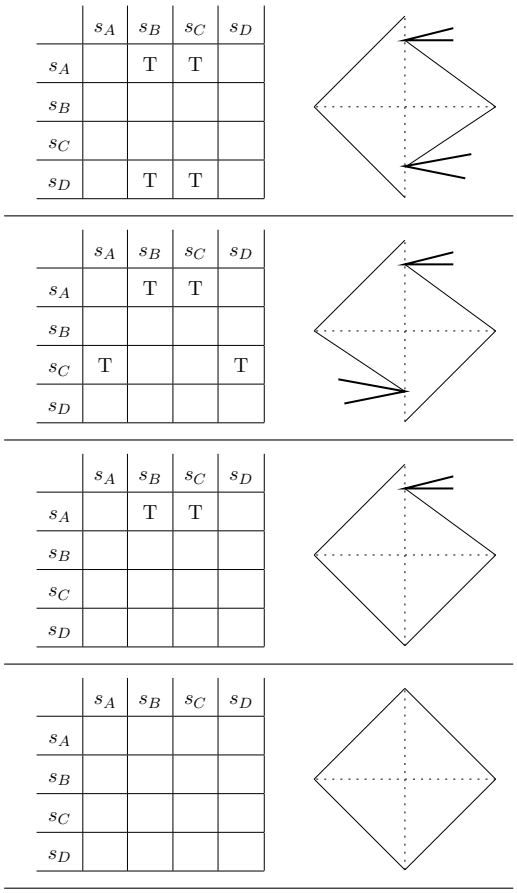


Figure 7.4: Legal combinations of visibility restrictions (up to isomorphism).

ing the distance moved in a start or finish so much). So care is needed to ensure that none of the bounding agents reach their targets during the Starting task.

The following four lemmas detail starting procedures for the cases shown in Figure 7.4. They all produce schedules which end with a convex arrangement with no bounding path in its interior. When agents move (except when otherwise specified) they should not move so far that any sightline between bounding agents cuts a bounding path. When agents move (except when otherwise specified) they should stop moving if any sightline between the bounding agents is about to cut a bounding path.

Lemma 45 *Cases represented by the first row of Figure 7.4 can be started on x-restricted paths.*

Proof: Throughout this proof, we will assume that, when any of the bounding agents move, the agents in $Ag_A \cup Ag_D$ will be moved to the segment $\overline{a_A\, a_D}$ and the agents in $Ag_B \cup Ag_C$

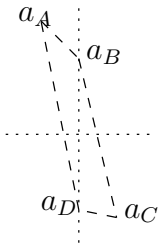


Figure 7.5: Convex hull of $Ag_A \cup Ag_C$ extended to include a_B and a_D.

will be moved to $\overline{a_B\, a_C}$. If the back projection of s_B onto τ_A and the back projection of s_C onto τ_D can see each other (without the sightline cutting a bounding path), then move a_A and a_D to those points. Care must be taken not to move a_A past the forward projection of a_D and vice versa while doing this or a_A and a_D may lose sight of each other. Once a_A and a_D have reached the relevant back projections, a_B and a_C can be moved a small distance. The agents in Ag_B and Ag_C should then be moved to the segment $\overline{a_B\, a_C}$.

Now we address the situation where the line segment joining the back projection of s_B onto τ_A and the back projection of s_C onto τ_D cuts bounding path. Move a_A and a_D alternately (ensuring they do not move past each other's forward projection) until one of them reaches the relevant back projection. Without loss of generality, suppose a_A reaches the back projection of s_B before a_D reaches the back projection of s_C. When this happens, move a_B one segment along its path or until it reaches the line through a_D and s_C, whichever happens first.

From now on, move the agents a_A, a_B, a_C in turn, as described in Lemma 43. Continue until a_D can safely move to the back projection of s_C, then move a_C forward a small distance.

This concludes our schedule for the Starting task.

□

Row two requires a simultaneous start of the agents in Ag_A and the agents in Ag_C. We know that there will always be a pair of x-restricted paths for which this is possible, even though they may be far from the individual shortest ones(Theorem 4). We will assume such paths are in use here.

Lemma 46 *Cases represented by row two of Figure 7.4 can be started (using x-restricted paths) provided those paths are a solution to the $Ag_A \cup Ag_C$ starting subproblem.*

Proof: Since the distance moved in simultaneous starts can be as small as desired, choose a value for d which allows s_B and s_D to retain visibility of the agents in Ag_C and Ag_A without

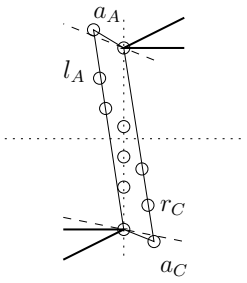


Figure 7.6: Agents in Ag_A and Ag_C on the edges of the hull in Lemma 46. Dashed lines show the forward and back projections of s_B and s_D.

the sightlines crossing bounding paths. We will follow the approach from type-C (Lemma 36). That is, the convex hull of $Ag_A \cup Ag_C$ can be extended to include s_B and s_D (and hence $Ag_B \cup Ag_D$ as well). See Figure 7.5. In this case we need to deal with the agents in Ag_D. Symmetry with Ag_B ensures that the hull with Ag_D added contains no bounding path (and therefore no obstacles).

Next move Ag_A and Ag_C to the segments $\overline{s_D\, l_A}$ and $\overline{s_B\, r_C}$ respectively (that is, the segments on the edges of the hull). This can be done (for both Ag_A and Ag_C) using the procedure for Ag_C (Step 3) from Lemma 36. We do not use Step 2 from that lemma because it arranges agents on a line through x, not the edge of the hull including Ag_D.

Now we have the situation shown in Figure 7.6 and so there must be room for either a_A or a_C to move again.

After each move by a_C, agents in $Ag_B \cup Ag_C$ should be advanced to $\overline{a_B\, a_C}$. Do the same with a_A and agents in $Ag_A \cup Ag_D$. We will assume that the internal agents will be moved in this manner throughout this proof. Continue this process until either a_A or a_C reaches the back projection of s_B or the back projection of s_D respectively. During this process the intersection between $\overline{a_A\, a_C}$ and S will be moving between s_B and s_D. (If either of those points were inside the original hull then the sightline should move into the interval on its first move and not move out again.)

First though, we need to be sure that it is possible to have either a_A or a_C reach the relevant back projection without moving a_B and a_D. If this were not possible it might not be possible to start at all. We proceed by contradiction. Suppose it is not possible to move a_A to the back projection of s_B without moving a_C past the forward projection of s_D. In that case, the lines through the first segments of τ_B and τ_D must cross in sector A. See

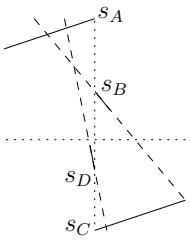


Figure 7.7: Demonstration that a_B and a_D cannot combine to deadlock both a_A and a_C. Projections are shown dashed, solid lines denote paths.

Figure 7.8: Agent a_A is at the back projection of s_B, the back projection of s_D is labelled p. Path segments are shown as solid lines, projections are dashed.

Figure 7.7 (page 148) for an illustration. Since the lines cannot cross twice, a_C will not be prevented by a_B from reaching the back projection of s_D. A symmetric argument applies to a_C.

So we have one agent at a back projection, now we need to get the other agent to the relevant back projection. Without loss of generality, assume a_A has reached its projection first. Now we need to show that a_C can reach the back projection of s_D without $\overline{a_A\, a_C}$ crossing a bounding path (or an obstacle).

Let p be the back projection of s_D onto τ_C. There are two possibilities here, either p is to the left of the forward projection of s_B (See Figure 7.8(a)) or p is to the right of the forward projection of s_B (See Figure 7.8(b)). The case where p is on the projection can be included in the Figure 7.8(a).

In the first case, the segment $\overline{a_A\, p}$ cannot be crossed by τ_B. We can also see by Lemma 30 that it cannot be crossed by τ_C either. This means that the section of τ_C from a_C's current position to p is both s_B-restricted and a_A-restricted. So, a_C can be moved to p without the sightlines to a_A or s_B crossing bounding paths.

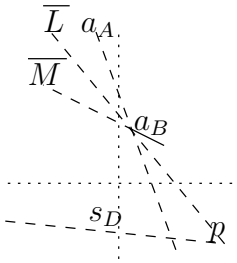


Figure 7.9: Used in Lemma 46. Projections are shown dashed.

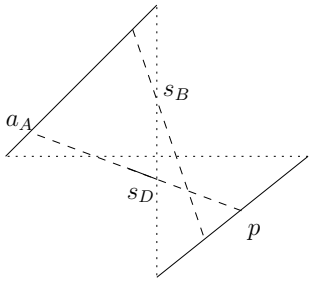


Figure 7.10: An instance where p is right of the forward projection of s_B. The points p, s_D and a_A are collinear.

What if p lies to the right of the forward projection of s_B (Figure 7.8(b))? We can advance a_B one segment without risking visibility. This may be enough to put p on the left of the new forward projection of a_B, but either way, we still cannot move a_C unless we can move a_A to give it more room. The main concern here is: can we move a_A without moving a_D? See Figure 7.9. Move a_A to either, the intercept of τ_A with $\overline{p\,a_B}$ (labelled $\overline{L}$), or to the back projection of a_B (labelled $\overline{M}$), whichever comes first. Advance a_C as far as possible and repeat as required until the forward projection of a_B runs through or to the right of p. Does a_D need to move while this is happening? Refer to Figure 7.10. Moving a_D would only be necessary if a_A was at the forward projection of s_D. If a_A lies on the forward projection of s_D, then it must be able to see p, so there is no reason for a_A to move past the forward projection of a_D. Since a_B has moved to have the forward projection of a_B to the right of p, and a_B does not move until a_A is on or forward of its back projection, the sightlines $\overline{a_A\,a_B}$ and $\overline{a_C\,a_B}$ do not cross bounding paths.

So far, we have an arrangement of agents where a_A and a_C have definitely moved, a_C is at the back projection of s_D, but a_B and a_D might not have moved. If a_A is still at the

back projection of s_B then a_B and a_D can both safely be moved a *small* distance along their first segments (if they have not already done so). Note that the forward projection of a_A cannot be at s_D so a_A does not block a_D from moving. Such a move will be safe because the sightlines have the wrong slope to be crossed by τ_A or τ_C. (Lemma 30).

If a_A is past the back projection of s_B then a_B must have moved already. Further, if a_A has reached the forward projection of s_D, then a_A, a_D, a_C are collinear so a_D can move some small distance. If a_A has not reached the forward projection of s_D, then a_A lies between the back projection of s_B and the forward projection of s_D. Agent a_D can move a small distance since $\overline{a_A\, s_D}$ slopes the wrong way to be crossed by x-restricted a_D (Lemma 30) and τ_A and has a steeper gradient than the first segment of τ_D and so cannot be crossed by it.

Finally, we need to show that none of the bounding agents have reached their target points. If p was to the left of the forward projection of s_B, then a_A and a_C lie at back projections and hence cannot be at their targets (Lemma 42). Agents a_B and a_D have moved distances which are small enough not to have reached their targets. If p was to the right of the forward projection of s_B, then a_A will only move as far as the back projection of a_B or the forward projection of s_D, whichever comes first. The back projection is ruled out by Lemma 42. What if τ_D consisted of a single segment which ends at t_A? This cannot happen since it would require a segment in τ_B with a back projection on or below T, which is impossible for x-restricted paths.

We have moved the agents into a convex arrangement with no bounding path in its interior, so we are done.

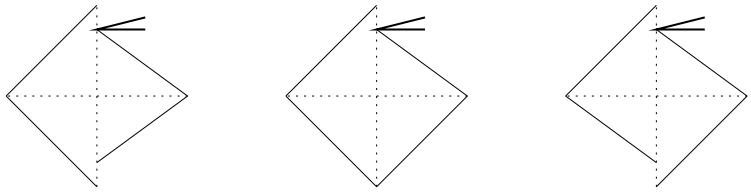
□

Lemma 47 *Cases represented by row three of Figure 7.4 can be started using x-restricted paths.*

Proof: The three possible arrangements (up to isomorphism) are shown in Figure 7.11. Figure 7.11(a) can be solved using the procedure for the first row given in Lemma 45. Figure 7.11(b) is a subtype of Figure 7.11(a). Figure 7.11(c) can be solved using the procedure given in Lemma 46.

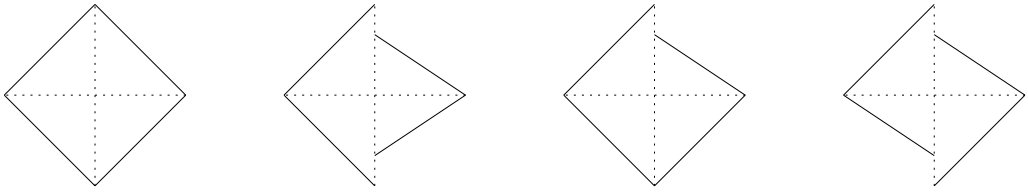
□

Note that instances such as those from row three can be started using shortest paths since the subproblem involving $Ag_A \cup Ag_C$ has a solution(Lemma 23). This does not imply that



(a) (b) (c)

Figure 7.11: Variants on row three of Figure 7.4.



(a) (b) (c) (d)

Figure 7.12: Variants on row four of Figure 7.4.

they always have an ideal solution since that would require finishing to also be possible using shortest paths.

Lemma 48 *Cases represented by row four of Figure 7.4 can be started using x-restricted paths.*

Proof: The variants (up to isomorphism) are shown in Figure 7.12. Figures 7.12(a) and 7.12(b) can be handled by the procedure for row one (Lemma 45). Figures 7.12(c) and 7.12(d) can be handled by row three (Lemma 47).

□

Theorem 6 *For a type-D MVPP-n there exists a set of paths which has a schedule, provided that Starting and Finishing tasks do not interfere with each other.*

Proof: The existence of x-restricted paths which work for the Starting and Finishing tasks comes from Lemmas 45, 46, 47 and 48. The Connecting task follows from Lemma 44.

□

Theorem 7 *Any type-D MVPP-n where the Starting and Finishing tasks do not interfere or require simultaneous movement has an ideal solution.*

Proof: The ability to start and finish using x-restricted paths is given in Lemmas 45, 47, 48. The Connecting task follows from Lemma 44. Since shortest Euclidean paths are x-restricted, an ideal solution can be produced using those lemmas.
□

7.2 Summary of Crossing Case

Where shortest paths are not used we assume that the replacement paths have the following properties (from Section 5.1.4):

- The paths permit the agents to start;
- The paths permit the agents to finish;
- None of the paths have disconnected intersections with any other paths;
- The paths must be x-restricted. This requirement may be relaxed to apply only to the section between the start and finishing tasks if visibility during those tasks can be guaranteed by other means.

Figure 7.13 summarises the different cases.

Sections 4.2.1 and 4.2.2 give algorithms for constructing bounding paths. The complexity of these algorithms depends on whether or not shortest paths are used.

7.3 Discussion

Our inability to guarantee the existence of an approximately ideal solution in a number of types all stem from the simultaneous start problem. Even if the inequalities in Chapter 4 have no solution, varying the positions of the start points must produce a solution. Can a reasonable approximation to the ideal be produced without moving the start points too far?

We showed how the Starting and Finishing tasks could interfere in type-C instances and how to produce schedules when that interference occurs in Chapter 6. However, we have not discussed the types of interference which can occur in type-D instances. This raises questions

Type	Vision	Quality	Copes with interference	Reference
Type-A	connected	approximate	-	Lemma 25
	connected-non-triv	ideal	-	Theorem 5
	complete-non-triv	ideal	-	Theorem 5
	complete	-	-	Theorem 4
Type-B	connected	ideal	-	Lemma 26
	complete	ideal	-	Lemma 27
Type-C	connected	ideal	-	Lemma 29
	complete-nosim	ideal	yes	Lemma 38
	complete-sim	-	yes	Lemma 39
Type-D	connected	ideal	-	Lemma 40
	complete-nosim	ideal	no	Theorem 7
	complete-sim	-	no	Theorem 6
Type-E	complete	ideal	-	Lemma 28

Figure 7.13: Summary of results for MVPP-n instances where S and T cross. Cases labelled "non-triv" mean that at one of the initial (and final) bounding agents has non-trivial visibility of the other. Cases labelled "sim" and "no-sim" indicate whether a simultaneous start (or finish) is required. The quality column contains "-" when solutions exist but we cannot make any guarantees about how close to the ideal they are. The interference column contains "-" when interference is not an issue for that type of instance.

about what types of interference can occur in type-D instances and if ideal solutions exist under interference.

Chapter 8

Varying speeds and Non-Collinear agents

This chapter presents extensions to material to the point-agent case material presented in previous chapters.

Section 8.1 discusses allowing the agents to move at different speeds. In previous chapters the continuous case, the start and target points were arranged in lines (S and T). In Section 8.2 we consider instances where start and target points are not arranged in lines. We do require, however, that the convex hulls of the start points and the target points be disjoint. Additional restrictions will be explained in the discussion.

8.1 Speed Ratios Other Than 1:1 in the Two-Agent Case

In Chapter 3 we required that the agents moved either at a constant speed v or remain stationary, with v common to both agents. Under those conditions, some MVPP instances did not admit schedules using shortest paths. Mitchell and Wynters [MW90b] showed that varying agent speeds allowed shortest paths to be used in all instances. So the question becomes whether there is a smaller relaxation of our assumptions which allows ideal solutions in all cases. In this section we show that allowing the agents to choose two speed ratios is sufficient to allow an ideal solution for any MVPP. That is, agent a_1 moves at speed 1 and agent a_2 moves at either k_1 or k_2. We still assume that acceleration is instantaneous.

Assuming that the instance is oriented like Figure 3.13, we can use similar reasoning to Lemma 8. The positions of the agents after moving are

$$a_1 = (d\sin\alpha, -D_1 + d\cos\alpha),$$
$$a_2 = (-kd\sin\beta, D_2 - kd\cos\beta).$$

The intersection between the sightline and S occurs at

$$\lambda = \sin\alpha/(k\sin\beta + \sin\alpha).$$

This gives a y coordinate for the intersection point of

$$y = \frac{1}{k\sin\beta + \sin\alpha}(D_2\sin\alpha - D_1 k\sin\beta + kd\sin(\alpha - \beta)).$$

Taking the limit as $d \to 0^+$ gives

$$(D_2\sin\alpha - D_1 k\sin\beta)/(k\sin\beta + sin\alpha).$$

Equating with the mid point between the two obstacles $(C_r/2)$ gives

$$k = (2D_2 - C_r)\sin\alpha/(C_r + D_1)\sin\beta.$$

This ensures that for sufficiently small $d > 0$ there is a k such that the intersection of the sightline between the agents and S lies between the obstacles.

So, two speed ratios chosen for a specific problem are enough to allow any MVPP where S and T cross to start and finish using shortest paths. Since the algorithms for solving two agent instances where S and T cross do not rely on moving agents together (apart from special case starting and finishing) the actual value of the speed of individual agents does not matter for the Connecting task. So either of the two chosen ratios can be used for the Connecting task. Since instances where the sightlines do not cross are already known to always admit ideal solutions, that covers all instances.

8.2 Non-Overlapping Arrangements of Point-Agents

In this section we examine n-agent instances where the start points are not necessarily collinear (the target points might also not be collinear). Specifically, we require that a line segment S (in the interior of the polygon) of minimal length can be drawn such that at least

one $p \in \{s_1, \ldots, s_n\}$ lies on S and that all paths Π_i intersect S. Further, the remainder of the start points lie either on ${}^{ext}S$ or on one side of it and all points t_i must lie on the other side of ${}^{ext}S$. To avoid confusion with the endpoints of the overall instance, the point where Π_i first intersects S will be denoted e_i. Symmetric statements define T. Finally, we require that ${}^{ext}S$ and ${}^{ext}T$ do not intersect.

Note that if all paths Π_i meet at a point which is also a start point, then there is no unique chord through S. In such cases, any chord meeting the other requirements will do.

Definition 36 *The* start hull *is the convex hull of the points* $\{s_1, \ldots, s_n\}$. *Similarly the* target hull *is the convex hull of the points* $\{t_1, \ldots, t_n\}$.

We also require that S and T do not intersect and that the start and target hulls do not intersect. If these restrictions are met, then the material in this section applies.

Definition 37 *The* entry *of the start hull (or target hull) is the minimal unbroken sequence of segments on the hull boundary such that all paths enter the hull through those segments. See Figure 8.1.*

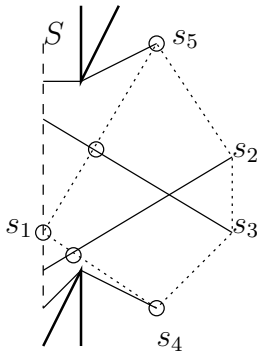


Figure 8.1: Circles indicate where the shortest paths leave the start hull (shown dotted).

We will address both the complete visibility and connected visibility constraint versions of the problem. Theorem 8 addresses the complete case, then the rest of the section covers the connected constraint.

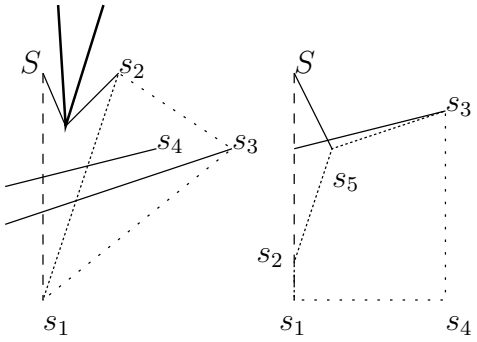


Figure 8.2: Two examples where the entry (dense dotted) touches S at one end. Paths are shown as unbroken lines, hulls as dotted lines and the polygon boundary with thick lines.

Theorem 8 *Consider an MVPP-n such that the start and target hulls are free of obstacles and do not intersect and S and T do not cross. In such a case, there exists an ideal solution under the complete constraint.*

Proof: We decompose the instance into three subproblems: before S, between S and T and after T. We know an ideal schedule exists for the middle subproblem from Chapter 4. We will only show how to produce schedules for the first subproblem since the last subproblem is symmetrical. For convenience we rotate the polygon until $\overline{S}$ lies vertically and all start points are on or to the right of ${}^{ext}S$.

We show that a schedule exists for moving agents from their start positions to S in two stages: first moving agents from their start positions to the entry of the convex hull, and second, moving agents from the entry to positions on S.

The complete constraint is in use so the start hull is free of obstacles, and agents can be moved directly to the entry, which deals with the first point. For the second point consider the following cases:

1. The entry to the start hull is a subset of S;
2. The entry to the start hull touches S at one end;
3. None of the above.

The first case has a trivial (empty) schedule since no moves are required. For Case 2 see Figure 8.2. Without loss of generality assume the entry touches at the south end of S.

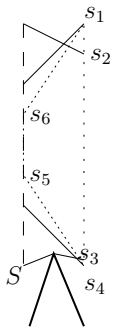


Figure 8.3: The entry of the start hull which is neither a subset of S nor does it contain either end of S. S is shown as a dashed line, paths as thin lines, the start hull is shown as a sparse dotted line and the entry as a dense dotted line.

Denote as a_1 an agent which is furthest North in S. A schedule for agents not already on S can be constructed by finding the upper (in the figure) bounding path then moving the bounding agent(s) along it, pausing at vertices to allow a change of bounding agent and to allow the other agents to move to the sightline between the current bounding agent and a_1. As can be seen in Figure 8.2(b), there may be stationary agents below a_1. However, the segment from them to any point in the entry runs entirely through the start hull and so is a sightline. By the triangle lemma, these agents retain visibility with all the moving agents.

For Case 3, see Figure 8.3. Here the agents end up arranged in a line but do not necessarily start that way. Choose agents at the extremes of the entry to be bounding agents. A schedule can be constructed finding upper and lower bounding paths and walking bounding agents along them, pausing at vertices to allow changes of bounding agents and other agents to move to the sightlines between them. Any agents already in front of the sweep can be ignored until the sweep reaches them.

Concatenating the schedules for the three subproblems gives the complete schedule. Note that there may be some movement along S and T between stages to meet the clipping assumption, but since there can be no visibility problems in such cases we do not explicitly address these moves.

□

The bounding path can be constructed using the methods from Chapter 4. This is not exactly the same situation as given in Chapter 4 since the start points might not be collinear,

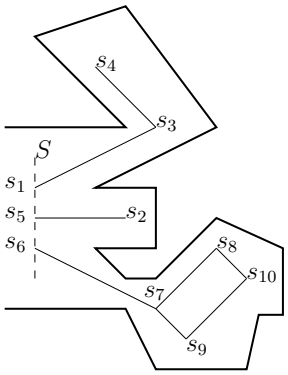


Figure 8.4: An example of an MVPP-n with non-collinear endpoints under the connected constraint. Thick lines show polygon boundaries, thin lines show visibility.

but such "floating" start points will never represent the best turn to make and so false edges to the start points will not enter the bounding path.

Now we move on to the connected constraint. In Chapter 4 even under the connected constraint the agents start and end in a state which obeys the complete constraint. In this case, however, the connected constraint allows instances with significantly less visibility than the empty convex hull required by the complete constraint. See Figure 8.4 for an example.

Definition 38 *The* start graph *for an MVPP-n contains a node for each s_i with edges between nodes which are mutually visible. A* valid path *through the start graph is a sequence $\{z_1, \ldots, z_k\}$ such that z_1 lies on S and for all $1 < i < k$ z_i is connected to z_{i-1} and z_i cannot see any nodes earlier in the sequence than z_{i-1}. Further, valid paths are not permitted to self-intersect. Node z_{i-1} is said to be a* parent *of z_i and a node with no children is termed a* leaf. *Note that this definition does not permit cycles in the path. We will use ζ to denote the path formed by the points in $\{z_1, \ldots, z_k\}$.*

The function $d_S(v : node)$ equals the number of edges in ζ from v to z_1 (which is defined to have value 0).

The function $d_S(\cdot)$ also actually depends on the path chosen through the graph. However as will become clear, potential ambiguity does not present a problem. Essentially, edges are removed from the start graph until each node has a unique valid path. This reduced start graph has a unique definition of $d_S(\cdot)$ for each node.

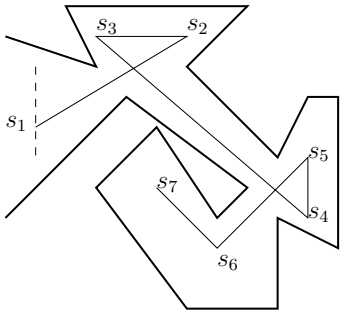


Figure 8.5: This arrangement is not permitted because the path from s_7 to S self-intersects.

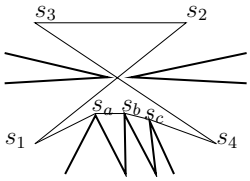


Figure 8.6: This arrangement is acceptable even though edges intersect, since there is an alternative longer path for s_4 via s_a

It turns out that the connected visibility constraint is harder than the complete constraint. We impose an additional constraint on connected visibility instances. Each leaf node in the start graph must have a valid path through the start graph in which does not self-intersect. For example, we do not permit arrangements like Figure 8.5. This does not mean the start graph must be planar, only that there must be non-self-intersecting paths for each node. See Figure 8.6.

As with the complete constraint, we focus on the task of moving agents to S. To avoid confusion with the target point of the overall instance, let e_i denote the point where Π_i first reaches S. The basic approach is to iteratively reduce the maximum $d_S(\cdot)$ value for the start graph until all nodes are visible to a node on S at which point the triangle lemma allows all the agents to move directly to S.

Some care needs to be taken with labelling here because there are two paths associated with each start point s_i. The first is the Euclidean shortest path Π_i from s_i to e_i. The second is a path ζ through the start graph from s_i to a start point which lies on S. When we wish to refer to the shortest path for a node z_k in the start graph we will refer to that path as Π_k.

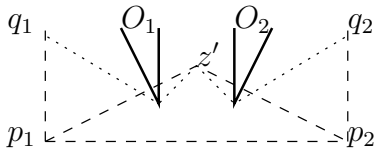


Figure 8.7: Illustration that "shortest path" (shown dotted) would need to be non-convex in order to have a point not visible to p_1 and p_2. Sightlines are shown dashed.

Lemma 49 *Consider two mutually visible points p_1, p_2 in a simple polygon, Π the shortest Euclidean path from q_1 to q_2 so that $\overline{p_1\,q_1}$ and $\overline{p_2\,q_2}$ are non-crossing sightlines. In this case $\exists z \in \Pi$ such that $\overline{p_1\,z}$ and $\overline{p_2\,z}$ are sightlines.*

Proof: First we show that no point on the path can be invisible to both p_1 and p_2 by contradiction. Note that if q_1 and q_2 are on opposite sides of ${}^{ext}\overline{p_1\,p_2}$, then Π must cut the extended segment $\overline{p_1\,p_2}$ and hence be visible to both points. This leaves q_1 and q_2 on the same side. Suppose there is a point z' on the path which is invisible to both p_1 and p_2. There must be obstacles O_1 and O_2 blocking vision. See Figure 8.7. However, since obstacles can only affect Π on one side, Π must be convex, so no point on Π can move between two obstacles. So by contradiction, all points on the path must be visible to at least one of p_1 or p_2.

Now we show that there is a point visible to both p_1 and p_2. From the triangle lemma we know that the part of Π which is visible to a point is continuous and closed. Suppose that the sections of Π visible to p_1 and p_2 do not intersect. Since Π is a connected continuous subset of $\mathbb{R}^2$, this means there must be a point on the path between them which is not part of either set. But that is a contradiction.

Note that non-crossing intersections are permitted since the intersection point must be on the path which gives the required point.

□

Note that Lemma 49 is also applicable when q_1, q_2 are the endpoints of a connected subset of a shortest path. This is because a connected subset of a Euclidean shortest path is also a Euclidean shortest path between its endpoints.

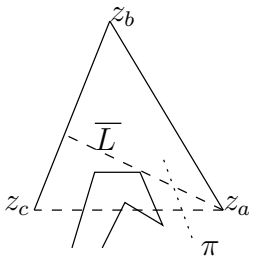


Figure 8.8: Illustration of Lemma 50

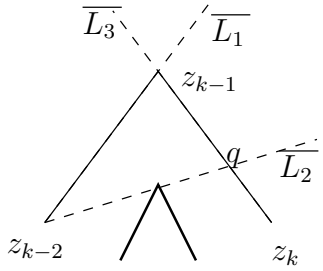


Figure 8.9: Definitions for Lemma 51.

Lemma 50 *Consider a triangle z_a, z_b, z_c where $\overline{z_a\, z_b}$ and $\overline{z_b\, z_c}$ are sightlines and $\overline{z_c\, z_a}$ is broken by an obstacle. If π is a polyline which*

1. *does not intersect $\overline{z_a\, z_b}$ or $\overline{z_b\, z_c}$,*
2. *leaves the triangle z_a, z_b, z_c between z_a and the obstacle,*
3. *does not intersect the exterior of the polygon,*

then π must contain a vertex which is visible to z_a.

Proof: Refer to Figure 8.8 for an illustration. We will assume the instance is oriented as shown in this figure. Let $\overline{L}$ be the chord through z_a which makes the largest possible angle with $\overline{z_a\, z_b}$ without being blocked by the obstacle. If π starts in the region above $\overline{L}$ within the triangle, then this endpoint will do. If π starts elsewhere, then it must turn to get below $\overline{L}$ so there must be a vertex or π on or above $\overline{L}$. Such a vertex must be visible to z_a, so we have our vertex.

□

Lemma 51 *For a valid path ζ (with two or more segments) from z_k to z_1, there is a point p on Π_k (from z_k to a point on S) which is visible to z_{k-2}.*

Proof: We proceed by contradiction. Suppose no point on Π_k is visible to z_{k-2}. Refer to Figure 8.9 for illustration of the following definitions. Let $\overline{L_1}$ be the chord through $\overline{z_{k-1}\, z_{k-2}}$. Let q be the point on $\overline{z_{k-1}\, z_k}$ closest to z_k which is visible to z_{k-2}. Let $\overline{L_2}$ be the chord through $\overline{z_{k-2}\, q}$. Note that $\overline{L_2}$ must touch at least one obstacle (otherwise $q = z_k$ and we have a contradiction). Let $\overline{L_3}$ be the chord through $\overline{z_{k-1}\, z_k}$.

Since all points on $\overline{L_2}$ are visible to z_{k-2}, if Π_k intersects $\overline{L_2}$, then any point in the intersection would do as the point claimed by the lemma. First consider $k = 3$, that is, $z_{k-2} = z_1 \in S$. This means that no part of S can be in the region below $\overline{L_2}$ containing z_k. The same obstacles which prevent $\overline{L_2}$ from sloping any lower, will prevent S from doing so.

So assume that $k > 3$ and Π_k remains below $\overline{L_2}$. So where is ${}^{ext}S$? We know that all start points (and hence all z_i's) must lie on one side of ${}^{ext}S$ so at least some points on ${}^{ext}S$ must lie below $\overline{L_2}$. We also want to know how ζ gets to S without vertices which are not parent and child seeing each other. Since ζ does not self-intersect it must either:

1. enter the triangle z_k, z_{k-1}, z_{k-2}, or

2. move above $\overline{z_{k-1}\, z_{k-2}}$.

In the first case, the only way out of the triangle for ζ(short of backing out the way it came), is to cross $\overline{z_k\, z_{k-2}}$ on the z_k side of the obstacles which cut $\overline{z_k\, z_{k-2}}$. However, from Lemma 50 we know that there must be a vertex on ζ which is visible to z_k. But this leads to a contradiction because such a vertex must be of lower $d_S(\cdot)$ than z_{k-1}, hence z_{k-1} could not be a parent of z_k.

In the second case (ζ moves above $\overline{z_{k-1}\, z_{k-2}}$) there are two possibilities, either z_1 lies on the left of $\overline{L_3}$ or it lies on the right. Point z_1 cannot lie on ${}^{ext}S$ because then z_k could see z_1 (which contradicts z_{k-1} being z_k's parent).

If z_1 lies to the left of $\overline{L_3}$, then we have a situation as shown in Figure 8.10. This arrangement would allow Π_k to reach a point on S without crossing $\overline{L_2}$. However, we need to consider how ζ gets from z_{k-1} to z_1. If ζ crosses $\overline{L_3}$, then there will be a point z_j ($1 < j < k-2$) in the triangle formed by $\overline{L_3}$, S and $\overline{L_2}$ (ζ cannot cross S). This triangle is clear of obstacles and z_{k-1} lies on its boundary, so z_j would be visible to z_{k-1}. This is a contradiction because it would mean z_{k-1} was not a child of z_{k-2}. If ζ does not cross $\overline{L_3}$,

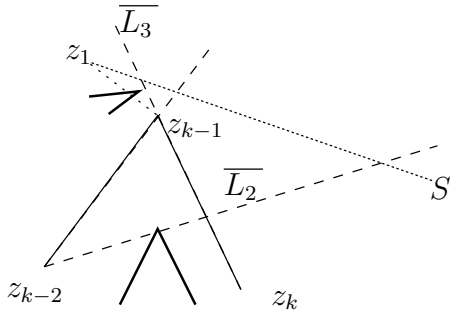


Figure 8.10: Point z_1 left of $\overline{L_3}$. The dense dotted line is ${}^{ext}S$ while the sparse dots show the blocked segment $\overline{z_{k-1}\, z_1}$

then consider the triangle bounded by $\overline{L_3}$, S and $\overline{z_{k-1}\, z_1}$. Since ζ must travel through the triangle to reach z_1, Lemma 50 means that there must be a z_j $(1 < j < k-2)$ visible to z_{k-1} which contradicts z_{k-1} being a child of z_{k-2}.

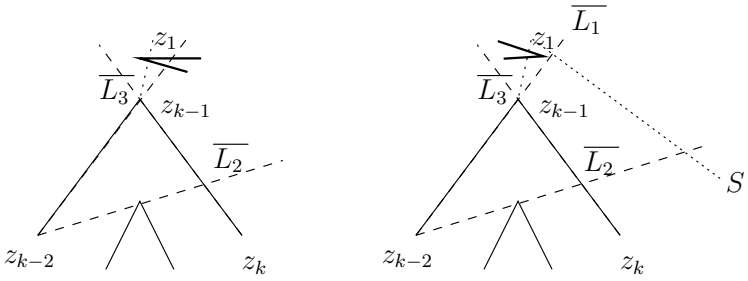
If z_1 is to the right of $\overline{L_3}$, then there are two subcases. Because z_{k-1} cannot see z_1, either $\overline{z_{k-1}\, z_1}$ is cut only from the left or it is cut from the right (possibly as well as the left). Figure 8.11(a) (page 166) shows a cut from the right and Figure 8.11(b) shows a cut from the left. An obstacle cutting from the right can be ruled out because S must pass through z_1 and also enter the region below $\overline{L_2\, w}$hich includes z_k. This would require S to turn anti-clockwise around the obstacle, which is a contradiction.

The only case left is $\overline{z_{k-1}\, z_1}$ being cut from the left (Figure 8.11(b)). If ζ cuts $\overline{L_1}$, then it has a vertex z_j $(1 < j < k-2)$ in the quadrilateral bounded by $\overline{L_1}$, $\overline{L_3}$, S and $\overline{L_2}$. If ζ does not cut $\overline{L_1}$, then Lemma 50 gives us a z_j $(1 < j < k-2)$ visible to z_{k-1}. In both cases this is a contradiction, since z_{k-1} would not be a child of z_{k-2}.

So there exists a point p on Π_k which is visible to z_{k-2}.

□

So Lemma 51 ensures that z_{k-2} can see a point on Π_k but it does not ensure that $\overline{z_{k-2}\, p}$ does not cross $\overline{z_{k-1}\, z_k}$. In fact that can happen. Lemma 53 will show how to untangle that arrangement if it does occur, but we still need to ensure that $\overline{z_{k-2}\, p}$ does not intersect any of the other segments in ζ. We need this to preserve the invariant for the new instance.



(a) Segment $\overline{z_{k-1}\, z_1}$ cut from the right.

(b) Segment $\overline{z_{k-1}\, z_1}$ cut from the left.

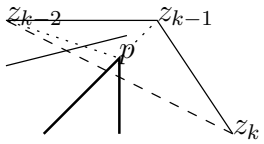
Figure 8.11: Point z_1 to the right of $\overline{L_3}$. Dotted lines show the blocked segment $\overline{z_{k-1}\, z_1}$.

Lemma 52 *Consider a valid (graph) path ζ from z_k to z_1; and the Euclidean shortest path Π_k from z_k to e_k with $p \in \Pi_k$ being the closest point to z_k on Π_k which is visible to z_{k-2}. The path through the start graph from p via z_{k-2} to z_1 does not self-intersect.*

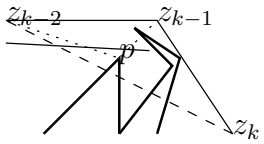
Proof: This proof works by contradiction. We will show that either p is not the closest point to z_k, or that the parent–child links in ζ are incorrect (that is, a node in ζ can see another node which is not its direct child or parent). First consider the case where $\overline{z_{k-2}\, p}$ does not cross $\overline{z_{k-1}\, z_k}$. See Figure 8.12(a). For ζ to cross $\overline{z_{k-2}\, p}$ (and not self-intersect) it must have a vertex z_x inside the region defined by z_{k-2}, z_{k-1}, p and z_k. Since p must be visible to both z_{k-2} and z_{k-1}, vertex z_x cannot lie in the triangle z_{k-2}, z_{k-1}, p. So z_x must be in the triangle z_k, z_{k-1}, p. But z_x is required to be invisible to both z_k and z_{k-1} (otherwise at least one parent–child relation will be invalid) so there must be an obstacle blocking sight. See Figure 8.12(b). Path Π_k would need to travel around this obstacle before it reached p so there must be an earlier point on Π_k which is visible to z_{k-2}. This gives the required contradiction.

Alternatively we could have the arrangement shown in Figure 8.13. We know that ζ cannot have an endpoint in the triangle z_{k-2}, z_{k-1}, p since then it would be visible to z_{k-1}. This situation does not work anyway because Π_k (being shortest) must cut $\overline{z_{k-2}\, z_{k-1}}$ in order to reach that position. Again this gives a contradiction because p is not the earliest candidate. This reasoning also rules out the possibility of p being to the right of $\overline{z_k\, z_{k-1}}$ and above $\overline{z_{k-1}\, z_{k-2}}$ (as shown in Figure 8.13).

Finally we consider the case where $\overline{z_{k-2}\, p}$ crosses $\overline{z_{k-1}\, z_k}$. Let z' be the vertex immediately before ζ cuts $\overline{z_{k-2}\, p}$ for the last time (that is z' has the highest index). See Figure 8.14. Now to



(a) The thin line shows the only way a later segment in ζ could cut $\overline{z_{k-2}\,p}$.



(b) For ζ to cross $\overline{p\,z_{k-1}}$ as well there must be another obstacle.

Figure 8.12: See Lemma 52. Segment $\overline{z_{k-2}\,p}$ does not cross $\overline{z_k\,z_{k-1}}$.

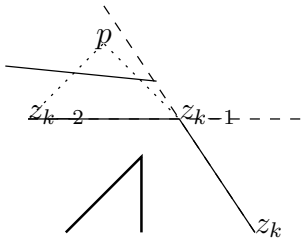


Figure 8.13: An arrangement where Π_k must cut $\overline{z_{k-2}\,z_{k-1}}$.

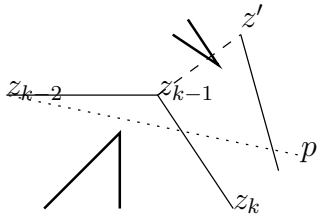


Figure 8.14: Path ζ cuts $\overline{z_{k-2}\,p}$.

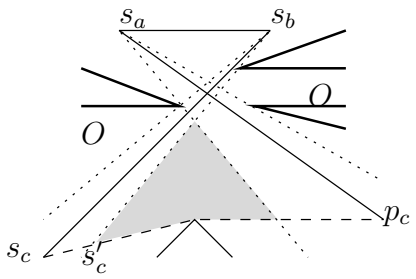


Figure 8.15: An instance as described in the statement of Lemma 53. Thick lines are obstacles, thin lines are the links in the start graph (and also $\overline{s_a\, p_c}$), dashed lines show Π_c and dotted lines show the limits of sight for s_a and s_b.

avoid contradiction z' must not be visible to z_{k-1} so there must be an obstacle between them. However, since ζ does not cut $\overline{z_{k-2}\, p}$ again and cannot cut itself, it must cross the segment $\overline{z_{k-1}\, z'}$ between z_{k-1} and an obstacle. But in order to make such a turn it will eventually need a vertex which is visible to z_{k-1}, and when that happens we have our contradiction. □

Lemma 53 *Consider consecutive nodes s_a, s_b, s_c in ζ such that s_c is a leaf and s_a sees a point $p_c \in \Pi_c$ and $\overline{s_a\, p_c}$ crosses $\overline{s_b\, s_c}$. Further, s_b has no children which are not leaves with crossing sightlines. In this case, the current instance can be transformed into a new instance with lower total $d_s(\cdot)$.*

Proof: Refer to Figure 8.15 for an example. In this method we wish to relocate s_b which is not a leaf, so we need to take some care dealing with its children. Note that s_b could be the parent of multiple nodes as well as s_c so we must account for these other children as well. If those children use an alternative path through the start graph there is no problem, but if they rely on s_b for connectivity, then we must also move those children as well. If any of the children of s_b are not leaves then they should be processed before s_c. To avoid potential confusion (since we are discussing relocating start points which ordinarily would not move), s_i denotes the start position of a_i in the current instance. Agent a_i will move and its ending position will be the new value for s_i in the new instance.

Some care needs to be taken when considering whether ζ in the new instance self-intersects or not. It is possible that when agents move they become visible to nodes which are not in ζ with lower $d_S(\cdots)$ values. This would mean that the agent would attach to a completely

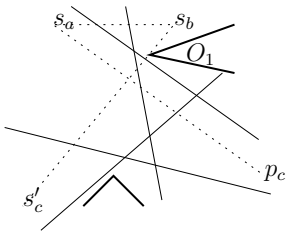


Figure 8.16: Some invalid positions for ${}^{ext}S$ (shown as thin lines).

different part of the graph. To avoid this complication we will keep nodes connected to the same graph path back to S throughout successive applications of this algorithm. That is, nodes not on the original ζ are ignored for path-finding purposes.

The following analysis applies to all children of s_b. Just substitute the point (and its path) for s_c and Π_c. We know that s_a cannot see s_c since then s_a would be a parent of s_c (and s_b could not be a parent) so we have a contradiction. If s_b also sees p_c, then a_c (the agent starting at s_c) can safely be moved to p_c without losing visibility. This makes s_a a parent of a_c and reduces $d_S(s_c)$ in the next instance. Before proceeding any further all children of s_b which can be safely moved in this way should be moved.

If there are any remaining children of s_b they must be such that they leave s_b's sight before they enter s_a's. Again, we use a_c as a representative of all such children. Move a_c and all its siblings to the edge of s_b's sight. Refer to Figure 8.15 where s'_c indicates the new position for a_c. The shaded region is invisible to both s_a and s_b. So from this point on we assume that s_b cannot see p_c.

Let O_1 be the obstacle inducing the next vertex on Π_b. Let L be ${}^{ext}\overline{s_b\, s'_c}$. Note that O_1 must lie on L. If not, s_b would see further along Π_c than s'_c. We know that e_c (the end of Π_c on S) must be on the right of L and below O_1 since $p_c \in \Pi_c$ is in that region. Since S cannot cut Π_c, and all s_i must be on at most one side of ${}^{ext}S$, we know that ${}^{ext}S$ cannot cross L. Doing so would mean that there would be s_i on both sides of ${}^{ext}S$. See Figure 8.16 for examples. Note that it is not possible to position ${}^{ext}S$ "under" all of the start points because doing so would require cutting the obstacle which blocks sight between s_c and p_c or requiring Π_c to make an upward movement after p_c (meaning Π_c cannot be shortest). This means that Π_b must travel to (and then around) obstacle O_1.

Since S cannot cut L, for all children s_x of s_b, the path Π_x must intersect L. Move all children of s_b to L. Now move a_b to O_1. Visibility with its children is preserved because this

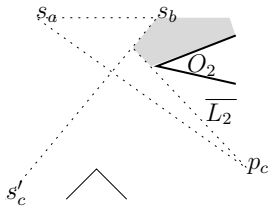


Figure 8.17: Used in Lemma 53 in the argument that a_b cannot move out of sight of s_a before being visible to p_c. The shaded area is not visible to p_c.

movement is along the current sightline. If this does not provide greater visibility of Π_c then move a_b to its next segment. Keep moving a_b until it gains greater visibility of s'_c. When a_b can see further along Π_c, move a_c and its siblings to the new edge of a_b's sight and resume moving a_b. This also applies to the other children as well, that is, when a_b gains greater visibility of Π_x, then a_x should be moved to the edge of a_b's vision (or to p_x, whichever comes first). If a_b becomes visible to a node in ζ with $d_S(\cdot)$ lower than s_a's then we need to stop and restart because we have reduced $d_S(\cdot)$ for s_b and all its children. So at the end of this step, either the nodes which were children of a_b are now children of s_a (and hence have lower $d_S(\cdot)$) or they are still children of a_b but a_b has lower $d_S(\cdot)$ (and hence so do all its children).

There are two possible objections to moving a_b which must be dealt with:

1. Could a_b lose sight of any of its children before they attach to another node?

2. Could a_b move out of sight of s_a before the above stopping conditions are met?

Objection 1 can not hold since when a_b moves it moves along the sightline with its children. The children only move between old and new sightlines with a_b so visibility won't be lost due to their movement.

For Objection 2 see Figure 8.17. Note that a_b can stop moving when it sees p_c since at that point s_a and a_b have a point on Π_c in common and so a_c can be safely moved to be a child of s_a. So a_b would need to move to a position on Π_b out of sight of both s_a and p_c. Let $\overline{L_2}$ be a segment from p_c to the segment $\overline{a_b\, s'_c}$ which touches obstacle O_2 (Note that O_2 must be the last obstacle blocking $\overline{p_c\, a_b}$). We know there must be at least one obstacle or p_c would be able to see a_b and we would be done. Path Π_b cannot intersect $\overline{L_2}$ or a_b would be able to see p_c when a_b moved there. This means that e_b (the end of Π_b on S) must lie in the shaded region shown in the figure. But this means there is no possible position for

${}^{ext}S$ which does not divide the s_i's. So Π_b intersects $\overline{L_2}$ meaning we can apply Lemma 49 to show there must be a point which both a_b and s_a can see simultaneously. Moving a_b to this point means that a_c can move to p_c without becoming disconnected and thus $d_S(s_c)$ can be reduced. The siblings of s_c can be treated similarly.

Once all this is done we will have an instance with lower total $d_S(\cdot)$. However, we also need to ensure that the new ζ we have produced does not self-intersect. That is, does $\overline{s_a\, p_c}$ cut any other edges in ζ? Since a_b is removed from the path (see above) we only need consider intersections with $\overline{s_a\, p_c}$. Lemma 52 assures us that such intersections do not occur.

□

Theorem 9 *With a non-collinear MVPP-n as described in this section where every start (target) point has a valid path there is a solution under the connected constraint.*

Proof: We will only address the task of moving the agents to S since Chapter 4 ensures the middle part of the solution exists and the final part is symmetric with the first. Given that we apply the clipping assumption, some movement within S and T may be needed before combining the schedules.

Denote as s_1 a start point which lies on S.

If $\max\{d_S(s_i)\} = 1$, then each node v with $d_S(v) = 1$ must be visible to some node v' on S. Using the triangle lemma on v' ensures that v can be moved along its path to its target on S without visibility problems.

For $m = \max\{d_S(s_i)\} > 1$ we proceed by induction on m. Suppose there are schedules for all instances $m = k$, then consider an instance with $m = k + 1$. Let w be the number of nodes v such that $d_S(v) = k + 1$. Consider one such v and z_m (a parent of v).

Applying Lemma 51 we know there is a point p_{m-1} on Π_v which is visible to z_{m-1}. If the sightlines $\overline{z_m\, v}$ and $\overline{z_{m-1}\, p_{m-1}}$ do not cross, then Lemma 49 tells us that there must be a point on Π_v visible to z_m and z_{m-1}. Agent a_v can be moved along Π_v to the earliest such point. This reduces $d_S(v)$ by one. Such a move is safe because v was a leaf, so no other nodes were dependent on it to remain connected. Further, the triangle lemma assures us that it remains visible to its parent throughout the journey. Finally, to preserve the invariant we need to ensure that this new position does not cause the path back to S to self-intersect. Lemma 52 ensures this.

If $\overline{z_m\, v}$ and $\overline{z_{m-1}\, p_{m-1}}$ do cross then we can employ Lemma 53 to reduce the $d_S(\cdot)$ for at least one node. The preconditions for Lemma 53 require that z_m has no children with non-crossing sightlines. This can be ensured by using Lemma 53 only after any children with non-crossing sightlines have already been dealt with.

If this process is repeated at most w times, then the instance can be restarted as an instance of size $m = k$. So by induction the result is established.

□

8.3 Summary

Section 8.1 showed that in the two-agent case, two chosen speed ratios is sufficient to permit ideal solutions for all MVPP instances.

Section 8.2 considers n-agent instances where the convex hulls of the start and target points are disjoint. We defined the start graph and the $d_S(\cdot)$ function. Theorem 8 proves the existence of an ideal solution under the complete visibility constraint provided the start graph can be reduced to a forest where no path self-intersects. It does this by inductively reducing the maximum $d_S(\cdot)$ value for the start graph. This is done using Lemma 53 if intermediate sightlines would cross, and using Lemmas 49, 51 and 52 otherwise.

8.4 Discussion

Section 8.1 showed two speed ratios are sufficient for the two-agent case. Can this be applied to the n-agent case? That is, are two sets of speeds for each agent sufficient to allow ideal solutions to all MVPP-n instances?

Section 8.2 does not cover any instance where ζ self-intersects. Can such an instance be solved? Section 8.2 was also restricted to instances where the start and target hulls do not overlap. How can schedules be produced for other types of instances?

Part II

Non-Point Agents

Chapter 9

Body-Agents

In this chapter we consider what happens if two point-agents are replaced with disks of non-zero area. This is important because the centre point of the agent cannot travel along the boundary of the polygon anymore. We will assume that the agents can move equally well in any direction (or alternatively that they can rotate on the spot) so there is no explicit rotation operation. Note that we still permit the agents to move through other agents. The proofs we give for the existence of schedules are constructive, and hence could be used to produce algorithms.

Definition 39 *A* body-agent *is represented by a circular disk with non-zero radius. Two such agents are visible if the line segment joining their centres does not intersect the exterior of the environment.*

Definition 40 *The* exclusion zone *of a polygon for an agent is the subset of the interior of the polygon which the centre of the agent may not enter.*

Since our agents are circular and we do not need to think about rotation, finding shortest paths for body-agents can be reduced to finding shortest paths for the centre point of the agents without entering the exclusion zone. A typical approach used here is to calculate the Minkowski sum[1] of the polygon boundary and the body of the agent. In this case the result is all points in the polygon closer than r to the boundary, but this has the drawback that the boundary of the exclusion zone may not be a polyline. Refer to Figure 9.1. Schwarz and

[1] A discussion of Minkowski sums can be found in the "Handbook of Discrete and Computational Geometry" [GO97].

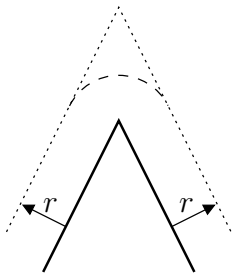


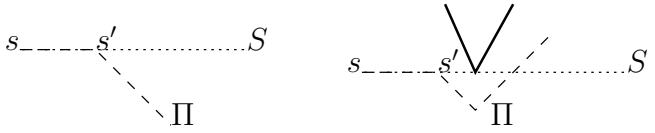
Figure 9.1: Exclusion zone around an obstacle. The dotted lines show the wide boundary version while the dashed arc shows the minimal zone.

Sharir's discussion [SS83] separates the lines induced by edges on the polygon boundary from the arcs induced by reflex vertices on the boundary. For discussion of minimal paths using a combination of line segments and circle arcs, see work on *Dubins Curves* [Lav06].

One possible approach to getting a polyline boundary is to construct the exclusion zone by offsetting the boundaries of the polygon by r and calculating new intersections. For example the dotted boundary in Figure 9.1. This gives a polyline boundary to the exclusion zone at the cost of "shortest" paths not being truly minimal [Pri]. Any path which travels around an obstacle can be shortened by "slicing off" a corner of the exclusion zone. In this section we are not strictly specifying the form of the exclusion zone except to say it must be constructable from the wide boundary form by "slicing off" a finite number of corners. This restriction guarantees that the zone is connected as well as being convex in the neighbourhood of a lone obstacle. We will use the notation Π_i to denote the shortest path under the chosen exclusion zone (even if a different zone would lead to a shorter path).

In this thesis, we assume that the exclusion zone given in input has a polygonal boundary. That is, any transformations needed to produce the exclusion zone are performed before the polygon is processed by our algorithms. As such, we make no claims about the relationship between the set of instances solvable under polygonal exclusion zones versus those solvable under more general curves.

Note that the exclusion zone is an open set (some other authors have chosen to treat the exclusion zone as a closed set).



(a) (b)

Figure 9.2: Instances where s would be moved to s' to meet the clipping assumption. Paths are shown dashed, solid lines are polygon boundary. As shown in (b), Π may intersect S again after s'.

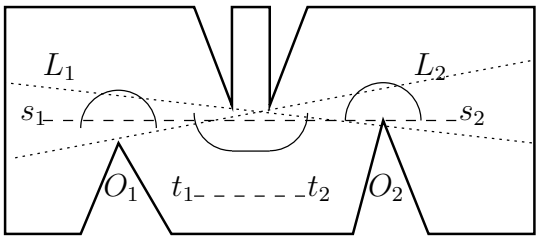


Figure 9.3: A body MVPP which has no solution. The minimum possible exclusion zone is shown as a thin line, sightlines are shown dashed.

Definition 41 *A* body MVPP *is a two-agent MVPP (see Definition 7) with body-agents (with a common radius) substituted for point-agents. The centre points of the agents may not enter the exclusion zone but agents are allowed to intersect each other.*

As in previous continuous versions we employ a form of the clipping assumption to recast instances in which shortest paths travel along S or T. Some care needs to be taken here, but we require that s_i be the last point in Π_i before it leaves S for the first time. We do not use the last contact with S in this case because unlike the previous types, the shortest path between two points on S might not be along S. A symmetric definition gives t_i. See Figure 9.2 for examples.

Body MVPPs differ from normal MVPPs in that there exist body MVPP instances which have no solution (ideal or otherwise).

Lemma 54 *There exists a body MVPP instance which has no solution.*

Proof: Refer to Figure 9.3. Consider the line labelled L_2. Since L_2 passes through the exclusion zone around the obstacle O_2, the agent a_2 must cross L_2 in order to reach t_2. But

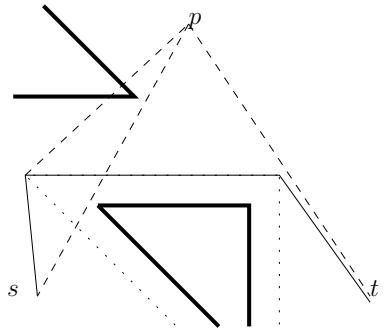


Figure 9.4: A case where the triangle lemma is false for a body MVPP. Thin solid lines show the shortest path, dotted shows the exclusion zone around the boundary. Thick solid lines show the boundary. Dashed lines show sightlines (or attempts at them.)

there is no position for a_1 which preserves visibility, while a_2 does this unless a_1 has already crossed S. A symmetric argument for L_1 means that neither agent can cross S until the other agent does, hence there is no solution.

□

The absence of a solution, in this case, is caused by the sightline passing through the exclusion zone, meaning the body-agent cannot follow the sightline. This emphasises the point that lines of sight and paths of motion are not necessarily equivalent (a point which also applies to the discrete version in Chapter 10).

The remainder of this section addresses the existence of ideal solutions under some conditions. A further complication introduced by body-agents is that the triangle lemma is no longer true. See Figure 9.4 for a counter example. We can, however, prove a more limited form in Lemma 55. Lemma 56 establishes some properties of shortest paths which travel on opposite directions. Theorem 10 establishes the existence of ideal solutions under certain conditions.

Lemma 55 *Consider a body MVPP with a shortest path* Π *such that the endpoints of the path,* s *and* t*, are both visible to a point* p *and no point on* $\overline{p\,s}$ *or* $\overline{p\,t}$ *lies in the exclusion zone. In this case all points on* Π *are visible to* p*.*

Proof: Path Π cannot cut either of the sightlines since no obstacles can get close enough to the inside of the sightlines to push a path through them. Similarly no obstacles on the outside can distort the shortest path. Thus we have the same situation as in the proof of

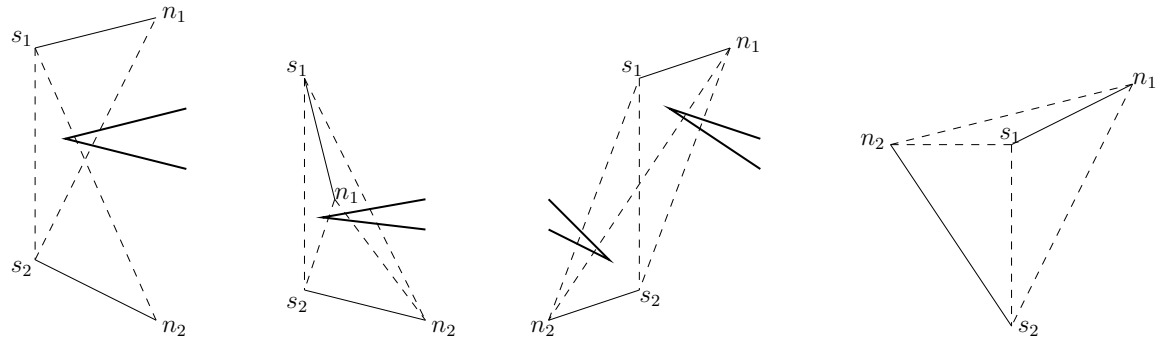


Figure 9.5: Possible arrangements of next steps (assuming both of the agents have them).

the triangle lemma, that is a shortest path pushed on at most one side and so Π must be convex. For example, a path with obstacles only touching below if the instance is oriented as in Figure 9.4.

By the same reasoning as the triangle lemma no obstacle can block vision between any point of Π and p.

□

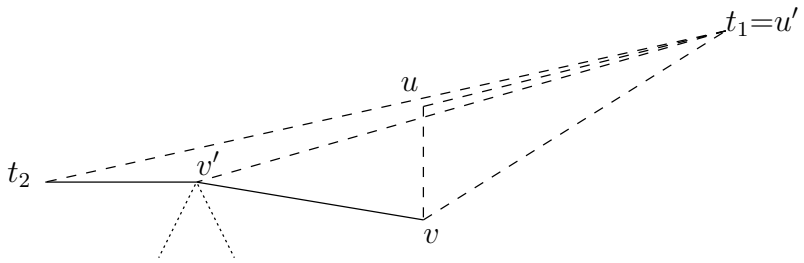


Figure 9.6: An example instance where the sightline $\overline{u'\,v'}$ cuts $\overline{u\,v}$ but T does not. Dense dotted lines indicate the exclusion zone.

Figure 9.5 shows some of the possible arrangements of sightlines and obstructions if neither agent has reached its target. We have omitted arrangements where one path's successor lies on the sightline between the current points. In such cases, we find a solution by moving such an agent to its successor and begin again. We wish to show that there will always be at least one visibility preserving movement, but first we will examine Figure 9.5(c) and Figure 9.5(d) in more detail. In these instances, the next segments of each shortest path lie on opposite

sides of ${}^{ext}S$. They are differentiated by whether or not $\overline{u'\,v'}$ lies inside the region bounded by $\overline{u\,u'}$, $\overline{u'\,v}$, $\overline{v\,v'}$, $\overline{v'\,u}$.

It is tempting to assume that in such cases, where intermediate sightlines cross, S and T must also cross but this is not the case. See Figure 9.6 for a counter-example.

For this lemma denote the second vertex of Π_1 and Π_2 as n_1, n_2 respectively.

Lemma 56 *Consider a body MVPP in which no point on S or T is in the exclusion zone; where S and T do not cross and each shortest path has at least one segment. If n_1 and n_2 lie on opposite sides of ${}^{ext}S$, then any legal positions for t_1 and t_2 must have the following properties:*

1. *Point t_1 lies on the same side of ${}^{ext}S$ as n_1;*

   AND

2. *Point t_2 lies on the same side of ${}^{ext}S$ as n_2;*

   AND

3. *T must cross ${}^{ext}S$ but not S itself.*

Proof: Notice that Property 3 says cross, not cut. The only difference between crossing (Definition 13) and cutting (Definition 22), is that cutting allows the intersection to occur at the end of the segment. However, both ends of ${}^{ext}S$ will lie on the polygon boundary, and will therefore be inside the exclusion zone. Since we are not considering cases where T passes through the exclusion zone, T cannot intersect ${}^{ext}S$ at either of its endpoints.

Talking about sides of ${}^{ext}S$ makes sense in this context because, as we will show, t_1 and t_2 cannot lie *on* ${}^{ext}S$.

Since ${}^{ext}S$ partitions the polygon, T must intersect ${}^{ext}S$ in some way. So for an arrangement to lack Property 3, $T \cap {}^{ext}S$ equals either

- T (that is, T is a subset of ${}^{ext}S$), OR
- a single point (an end of ${}^{ext}S$ or T),

No intermediate possibilities (for example, more than one point from T but not all of T is in the intersection) are not allowed. In order for more than a single point (but less than the whole of T) to be in the intersection, T would need to extend past an endpoint of ${}^{ext}S$.

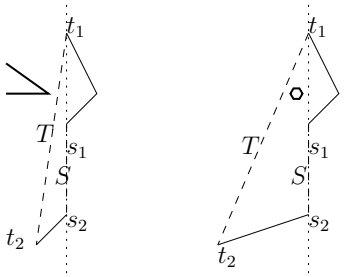


Figure 9.7: Arrangements where Π_1 and Π_2 start in opposite directions but t_1 lies on ${}^{ext}S$. Dotted lines show ${}^{ext}S$, thick lines show polygon boundary.

But those endpoints lie in the exclusion zone where T cannot go. So the only ways two line segments can intersect are the ones listed above. Note that if T intersects S, then since no part of S or T lies in the exclusion zone, at least one of Π_1, Π_2 has zero segments (by the clipping assumption) which contradicts the lemma preconditions.

Now we prove Property 3 by contradiction. We address the possibilities listed above individually, beginning with T being a subset of ${}^{ext}S$. Part of ${}^{ext}S$ between S and T must lie in the exclusion zone or else the shortest paths would have moved along the line. In order to get around the exclusion zone both shortest paths would need to be on the same side of ${}^{ext}S$, which contradicts the requirement that they start by moving in opposite directions.

What about T intersecting ${}^{ext}S$ at an endpoint? Since the agents cannot reach the endpoints of ${}^{ext}S$, the only available endpoints are t_1, t_2. The only way this could happen is if the paths enclose an obstacle or if T passes through the exclusion zone both of which give a contradiction. See Figure 9.7 for an illustration. So by contradiction the instance must have Property 3.

Now we consider the other two properties. Since the preconditions do not permit S and T to cross, an arrangement which has Property 3 but lacks one of the other properties must in fact lack both of them.

Suppose we have such an instance, that is, t_1 lies on the same side of ${}^{ext}S$ as n_2 and vice versa. This means that both paths must cut ${}^{ext}S$ to reach their targets. Why then did they start by heading in the opposite direction? For a shortest path to do this, there must have been an obstacle. Such an obstacle would need to be in R, that is enclosed by paths and sightlines, so we have a contradiction (since we are in a genus zero polygon).

So we have all the required properties.

□

Theorem 10 *A body MVPP in which no point on S or T is in the exclusion zone and where S and T do not cross, has a solution.*

Proof: In this proof we follow the general approach of the two-agent case (Theorem 1 page 56) but we need to ensure that the intermediate sightlines do not enter the exclusion zone.

We need to show that for any instance, a step is possible and that step produces a new instance meeting the same conditions.

Now we show that progress is always possible. If one of the agents has reached its target, then the other agent can move directly to its target (visibility is guaranteed by Lemma 55). For the rest of this proof we assume that neither agent has reached its target.

Note that neither of the paths can cut the segment S after moving off ${}^{ext}S$. We know this because there are no obstacles close enough to the segment to distort the path, so if the path was to travel to a point on the segment, then it should have gone straight there.

Further note that neither path may cut T. To do so means that the path is unable to move directly along T, meaning part of T must be in the exclusion zone which is not allowed by the lemma's preconditions.

First we will deal with the case where $\overline{s_1\, n_1}$ and $\overline{s_2\, n_2}$ cross. Here we move both agents to the intersection point. The paths should be checked to determine if they intersect like this. From now on we will assume that those segments do not cross.

Figure 9.5 shows possible relative positions for n_1 and n_2 (the next segments of Π_1 and Π_2). However, it does not show arrangements where one of them lies on ${}^{ext}S$. In such cases, the move should be made in preference to any other possibilities since no part of the paths can lie in the exclusion zone and we are assured that no part of the new S will do so either. This step also helps preserve the clipping assumption.

We will deal with each of the cases in Figure 9.5 separately. First Figure 9.5(a) and Figure 9.5(b) (these figures have n_1 and n_2 on the right hand side of ${}^{ext}S$). We further decompose this case by the positions of t_1 and t_2 relative to ${}^{ext}S$ (we assume the instance is oriented so that $\overline{s_1\, s_2}$ is vertical; other orientations will follow by symmetry).

1. Both targets are left of ${}^{ext}S$ —

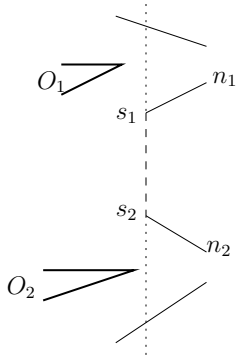


Figure 9.8: Referred to in Item 1. Both t_1 and t_2 are to the left of ${}^{ext}S$ and above and below the obstacles respectively.

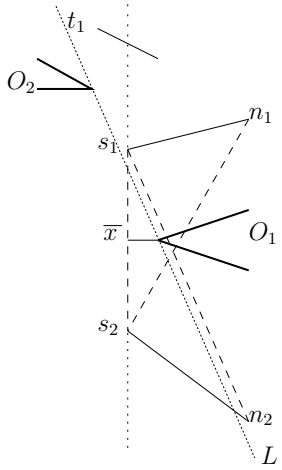
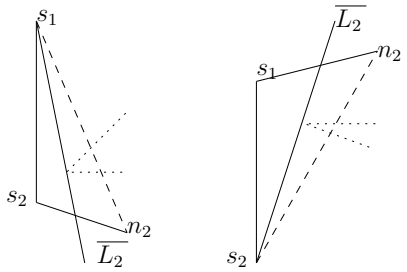


Figure 9.9: Referred to in Item 2, t_1 is left of ${}^{ext}S$ and t_2 is right.

This means that both paths must cross ${}^{ext}S$ somewhere other than between s_1 and s_2, since no point in S lies in the exclusion zone. Since the agents have moved off ${}^{ext}S$, there must have been an obstacle to necessitate this, such as in Figure 9.8 (the obstacles are labelled as O_1 and O_2). The presence of obstacles means that the paths cannot move closer to each other on the left side of ${}^{ext}S$ because they cannot get past the obstacles. So T must run somewhere between ${}^{ext}S$ and a line through the two obstacles. This would place parts of T in the exclusion zone, which is a contradiction.

This case is not possible.

2. One target is left of ${}^{ext}S$. The other is to the right —



(a) (b)

Figure 9.10: The exclusion zone (shown dotted) cutting a sightline in Item 2.

Without loss of generality we assume that t_1 is to the left. Refer to Figure 9.9. Although the figures here illustrate the arrangement shown in Figure 9.5(a), the same reasoning applies to Figure 9.5(b).

If any of the sightlines are not blocked then we are done, so assume there is at least one obstacle which cuts $\overline{s_1\, n_2}$. Choose a vertex O_1 which is closest to ${}^{ext}S$ from one of these obstacles. Draw a segment $\overline{x}$ from the vertex to a point on S (not s_1 or s_2). If $\overline{x}$ is cut by another obstacle choose a point on S closer to s_2 and try again. Note that n_2 must lie below $\overline{x}$ and to the right of ${}^{ext}\overline{x}$ (otherwise O_1 could not block sight between s_1 and n_2.

By the same reasoning as that in Item 1, there must be an obstacle close to ${}^{ext}S$ to distort Π_1 to the right. Label the first such obstacle O_2 (all this is shown in Figure 9.9).

Draw a line through O_1 and O_2 (labelled L). The target t_1 must be on the right of this line and t_2 must be on the left (otherwise they cannot possibly be visible to each other). Note that O_1 blocks sight between s_1 and n_2, and the path Π_2 must cross L. This means that T must cross Π_2 as well. The only way this could happen for a shortest path is if part of T lay in the exclusion zone which is a contradiction. So there must be at least one move which preserves visibility.

So there must be at least one possible visibility-preserving movement. Choose the possibility which makes the smallest absolute angle with S. Now we need to show that the new S which results is clear of the exclusion zone. Let us suppose it is not, for the purposes of contradiction.

If a_2 moves to n_2, then $\overline{s_1\, n_2}$ passes through the exclusion zone. See Figure 9.10(a). Consider the line L_2 through s_1 and the edge of the exclusion zone which cuts $\overline{s_1\, n_2}$. To have T not pass through the exclusion zone, t_2 must be to the left of L_2. This leads to a contradiction because then Π_2 would cut T.

If a_1 moves to n_2, then $\overline{s_2\, n_1}$ passes through the exclusion zone. See Figure 9.10(b). Drawing a line through s_2 and the exclusion zone cutting $\overline{s_2\, n_1}$ leads to a contradiction by the same reasoning as above.

So, there is a move which preserves the constraints.

3. Both targets are right of ${}^{ext}S$ —

   Following the notation from Item 2, in order for all sightlines to be blocked or run through the exclusion zone, we have a $\overline{x}$. For Π_1, Π_2 to be shortest they cannot intersect $\overline{x}$. So, T must cut $\overline{x}$ since T cannot intersect the exclusion zone. This means that T cuts Π_1 or Π_2 which gives us our contradiction (by the same reasoning as in Item 2).

   This case is not possible.

4. One of the targets is on ${}^{ext}S$ —

   Without loss of generality assume t_1 lies on ${}^{ext}S$. The target t_1 must lie above s_1, otherwise Π_1 would run along S and would have been clipped. Further, there must be a part of the exclusion zone on ${}^{ext}S$ between s_1 and t_1 otherwise a_1 would have already advanced.

   All this means that if t_2 is on the left of ${}^{ext}S$, then T must pass through the exclusion zone, which is a contradiction. If t_2 lies on the right of ${}^{ext}S$ we also arrive at a contradiction by the reasoning as in Item 3.

   This case is not possible.

5. Both targets lie on ${}^{ext}S$ — that is, $T \subseteq {}^{ext}S$.

   If T intersects S, then no obstacle can prevent the agents moving directly along ${}^{ext}S$ to their targets. If T does not intersect, then either the paths travel along ${}^{ext}S$ (in which case we are done) or they are forced off ${}^{ext}S$ by an obstacle. For the rest of this part we assume that the paths are forced off ${}^{ext}S$. See Figure 9.11 for an example. Without loss of generality assume that the instance is oriented as in the figure.

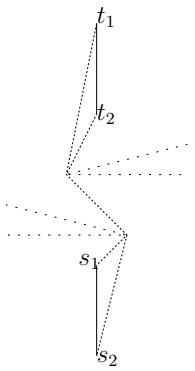


Figure 9.11: T lies on the chord through S. Dense dotted lines show Π_1 and Π_2, sparse dotted lines show exclusion zones (not obstacles). In this figure s_1 can see t_2.

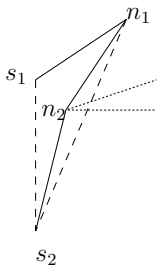


Figure 9.12: From Item 5 — Point s_2 sees n_1 but the sightline passes through the exclusion zone.

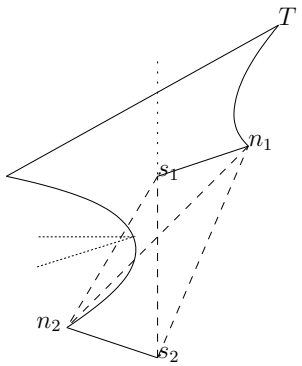


Figure 9.13: An instance of Figure 9.5(c) with T above S.

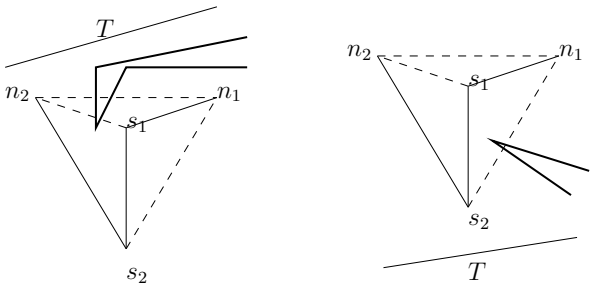
> First consider $\overline{n_1\, s_2}$. If it is not blocked or in the exclusion zone, then we are done. If this is not the case, then we have the situation shown in Figure 9.12. Since the polygon is genus zero, Π_2 must move around the exclusion zone. This constrains the position of n_2 to be closer to s_1. Any part of the exclusion zone or obstacle that would intersect $\overline{s_1\, n_2}$ would mean that Π_2 was not shortest since it would need to divert around it later.
>
> So at least one of $\overline{s_2\, n_1}$ and $\overline{s_1\, n_2}$ must be clear of obstacles and does not enter the exclusion zone.

So only Items 2 and 5 have possible moves. This concludes the proof for Figure 9.5(a), and for Figure 9.5(b).

Now we consider Figure 9.5(c). From Lemma 56 we know that if this problem instance is well formed, then T must cut ${}^{ext}S$ above or below S and t_2 must be on the same side of ${}^{ext}S$ as n_2.

First assume T is above S as in Figure 9.13. If $\overline{s_1\, n_2}$ passed through the exclusion zone, the path Π_2 would need to travel back closer to S in order to get around the zone and reach its target. This means, however, that Π_2 cannot be shortest since no obstacles could force this behaviour without being in the region bounded by Π_1, Π_2, S and T. So there must be at least one sightline $\overline{s_1\, n_2}$ or $\overline{n_1\, s_2}$ which is not blocked and which does not enter the exclusion zone. If T were below S, then a symmetrical argument can be applied using Π_1.

Now we deal with Figure 9.5(d): first the case where T is above S (see Figure 9.14(a)). If $\overline{s_1\, n_2}$ cuts the exclusion zone (or an obstacle), then at least one of the paths must not be shortest in order to move around the exclusion zone. If $\overline{s_1\, n_2}$ does not cut the exclusion zone then we have our new sightline.



(a) (b)

Figure 9.14: Instances of Figure 9.5(d) with movement sightlines blocked. Heavy lines indicate polygon boundary.

Now we look at the case where T is below S (see Figure 9.14(b)). Note that if $\overline{s_1\, n_2}$ cannot be cut from below since that would require cutting Π_2 as well. If $\overline{n_1\, s_2}$ cuts the exclusion zone, then Π_1 must not be shortest since it needs to travel closer to S to get around the exclusion zone. If $\overline{n_1\, s_2}$ does not cut the exclusion zone, then it will do as our new sightline.

This addresses all the cases shown in Figure 9.5. All instances can be addressed by one of them directly or by symmetry. In each case, the new S does not enter the exclusion zone. As mentioned above, if an instance is formed where one of the agents' next moves lies along S, that move should be performed immediately. This preserves the clipping assumption so each new instance meets the lemma preconditions.

We can apply this method inductively until one or both the agents reaches its target. In the case of one arriving before the other, Lemma 55 can be applied to move the other agent directly to its target.

□

Further work on this question could give an algorithm for constructing schedules and address agents of different radii, non-circular agents and instances where the start and target sightlines cross. Another possible avenue is to prevent the agents from colliding while moving. This is likely to be challenging though since Peng and Akella [PA05] consider arbitrary paths for just the non-colliding aspect and they indicate that the optimal solution (in terms of travel time) is NP-hard.

9.1 Summary

Lemma 54 showed there is an instance which has no solution. Lemmas 55 and 56 support the proof of Theorem 10. Lemma 55 is a restricted version of the triangle lemma to allow an agent to be moved to its target if the other agent has already reached its target. Lemma 56 shows the relationship between the first segments of each path and the position of the target points. Theorem 10 shows that S and T not crossing and not entering the exclusion zone is a sufficient condition for the existence of an ideal solution.

9.2 Discussion

We have shown how to produce an ideal solution for body MVPPs under some conditions. A number of questions remain. Is there a simple test to determine whether a body MVPP admits an ideal solution in the general case? For those cases which are solvable, how can schedules be produced? What impact does adding more agents to a body MVPP have on the existence of solutions?

Chapter 10

Discrete Case

In this chapter we consider a discrete form of MVPP. Note that unlike the rest of this thesis, in this chapter, we assume the RAM model of computation. As discussed in Chapter 2, discrete environments can be used where an exact representation is either infeasible or just not needed. In this case, the environment (and paths) for the problem will be composed of discrete cells. Using a different measure of path length means that there may be more than one path of minimal length between two cells. This changes the question from whether the shortest paths admit a solution; to whether there is a pair of shortest paths which admit a solution

After some definitions we will detail some differences between this form and the MVPPs from Chapter 3. We will show how to determine if a given instance has an ideal solution and how to produce the solution. If an ideal solution is not possible, then the method can be simply extended to non-ideal solutions. Next we will describe some efficiency improvements in certain problem instances. Finally we identify a set of instances which can be solved more efficiently.

Throughout this chapter, Λ will denote the number of cells in the interior of the polygon. To distinguish cells from points, we write cells like this c_x. For example, the starting cell for agent a_2 is written c_{s_2}. Cells are closed sets so a point on a boundary could belong to more than one cell.

Definition 42 *In this chapter, a* path *is a sequence of cells such that each cell is a neighbour of the one preceding it. The length of a path is the number of cells it contains.*[1]

For example, in an 8-connected set, a path from a cell to its North-West neighbour would have length 2.

Definition 43 *A subset R of a grid of cells is* 4-connected *if there is a path between any two cells in R consisting of movement between neighbours (cells in R directly* North, East, South *or* West *of the current cell).*

A similar definition with neighbours North, North-East, East, South-East, South, South-West, West, North-West gives 8-connected *sets. The directions may be abbreviated to letters.*

Definition 44 *A* simple discrete polygon *consists of a 4-connected collection of* cells *forming the interior bounded by a 4-connected collection of cells forming the exterior. Points on the boundary of two cells are considered to belong to both cells.*

This 4-connected restriction (in Definition 44) applies even if 8-connected movement is in use. Note that this definition does not imply that a simple discrete polygon must be genus zero. We will, however, highlight some situations where savings can be made if a polygon is known to be genus zero.

Definition 45 *Two cells are* visible *if the line segment joining their centres does not leave the interior of the polygon.*

This definition is somewhat subtle. If the line runs along the boundary, then Definitions 44 and 45 would have those points being part of both the interior cells and the exterior cells, hence the wording here.

[1] The initial definitions in this chapter are similar to those in Agoston [Ago04] but there are differences. For example, Agoston has the length of a discrete path as *number of cells*−1. Also the border cells are part of the navigable area, whereas in this treatment, border cells are obstacles.

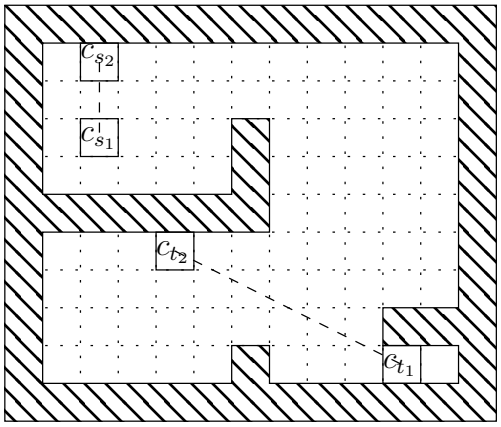


Figure 10.1: A DMVPP

Definition 46 *A* discrete mutually visible path problem *DMVPP consists of a simple discrete polygon and four cells c_{s_1}, c_{s_2}, c_{t_1}, c_{t_2} within its interior. A solution to a DMVPP consists of paths $c_{s_1} \rightarrow c_{t_1}$ and $c_{s_2} \rightarrow c_{t_2}$ and a schedule for moving agents (a_1, a_2 respectively) along these paths so that visibility is preserved. Agents are assumed to always be at the centre of a cell. As in the previous chapter, S and T will denote the sightline between the start and target points respectively. We say that a DMVPP is genus zero, if the underlying polygon is genus zero.*

An example of a DMVPP is shown in Figure 10.1.

As in Chapter 3 we prefer solutions using paths of minimal length. In this chapter we allow agents either 4-connected or 8-connected movement.

Definition 47 *An* ideal solution *to a DMVPP is a solution which uses minimal discrete paths under the prevailing movement constraint. In this chapter we will denote minimal paths as μ.*

In this section we will use $\|\mu_i\|$ for the length of μ_i (since the cardinality equals the length) and reserve $|\mu_i|$ for the Euclidean norm.

10.1 Contrasts With MVPPs

The first major difference is that there may not be a unique shortest path between two cells. For this reason we will use "minimal" to describe paths with the smallest possible length,

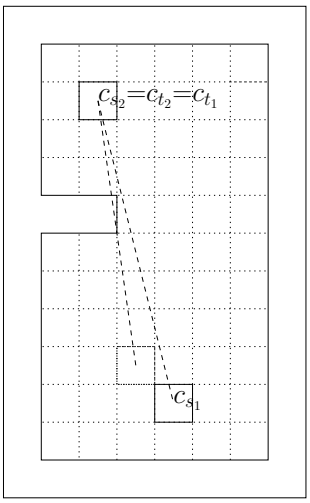


Figure 10.2: In this instance, $c_{t_1} = c_{t_2} = c_{s_2}$ so a_2 does not need to move. There are a number of minimal paths available to a_1, but, any which pass through the dotted cell will lose visibility with a_2.

rather than "shortest" as used in other chapters. Sometimes, some minimal paths will admit solutions while others do not. See Figure 10.2. This example applies to both 4-connected and 8-connected movement. In this case, there is a minimal path through the dotted cell but visibility is lost. On the other hand a path moving North for a few cells will work fine. Thus the question arises, which minimal path should be used?

In other cases, instances exist for which there are solutions but none which use a pair of minimal paths. For example, see Figure 10.3(page 195). In that case neither of the agents can start on their minimal paths but a non-minimal solution is shown in Figure 10.4.

Finally there are instances which admit no solutions at all. One such case is shown in Figure 10.5. Under 4-connected movement, neither agent can move from their start point at all.

An additional complication is that the triangle lemma is not true in the general case.

In the next section we examine a method to find a solution (if one exists) using minimal paths.

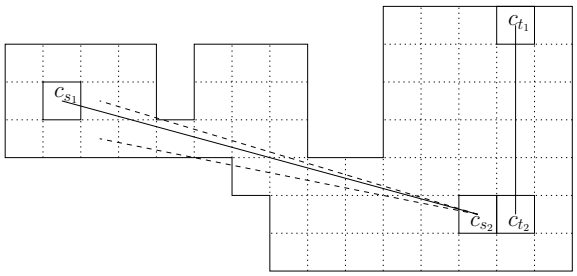


Figure 10.3: An instance with no minimal solution. Any path for agent a_1 must pass through one of the cells to East or South-East of s_1. However, as the dotted lines show, there is no sightline between those cells and s_2. They are also invisible to t_2 which is the only other cell on a_2's minimal path.

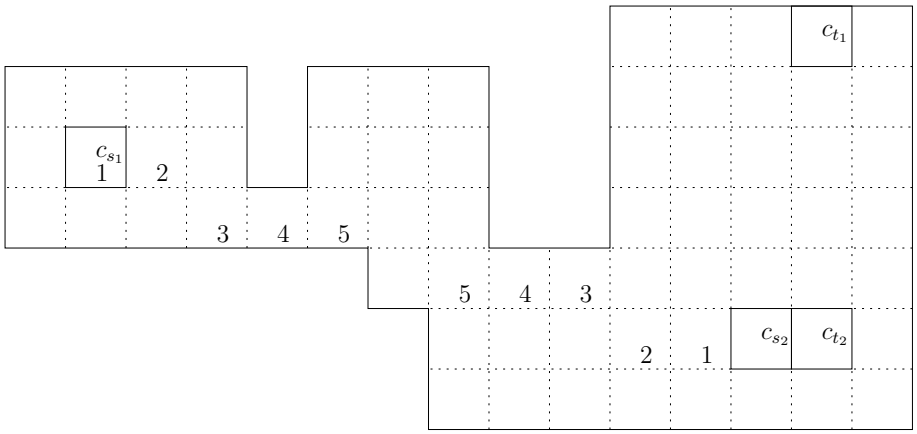


Figure 10.4: The beginning of a solution to Figure 10.3. The positions for both agents in the first five steps are shown. For example, in the first step, a_1 remains still while a_2 moves left.

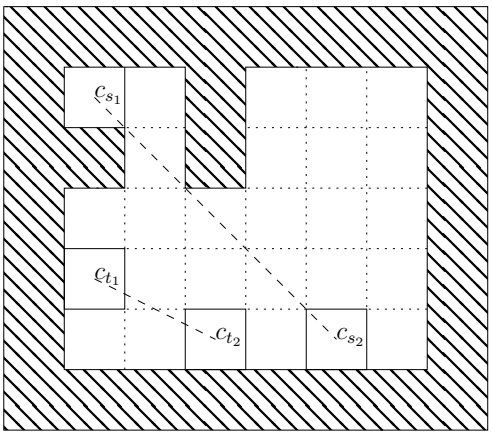


Figure 10.5: A DMVPP with no solution in 4-connected.

10.2 Solution for Minimal Paths

Definition 48 *The minimal path region $R(c_s, c_t)$ is the union of all minimal paths from c_s to c_t.*

Definition 49 *The distance $d(c_s, c_t)$ is the minimal path length from c_s to c_t. Where the target is unambiguous it can be omitted, thus $d(c_s)$.*

Note, since the polygon is simple there must be at least one path between any two cells in the interior.

Definition 50 *The* minimal state graph *for a DMVPP is a graph $G(V, E)$ where the nodes of the graph are ordered pairs (c_{v_1}, c_{v_2}) where $c_{v_1} \in R(c_{s_1}, c_{t_1})$, $c_{v_2} \in R(c_{s_2}, c_{t_2})$ and where c_{v_1} and c_{v_2} are mutually visible. So each node represents an arrangement of the two agents where they can see each other. There is a directed edge from $U = (c_{u_1}, c_{u_2})$ to $V = (c_{v_1}, c_{v_2})$ if moving a_1 from c_{u_1} to c_{v_1} and a_2 from c_{u_2} to c_{v_2} are legal moves and $d(c_{v_1}, c_{t_1}) + d(c_{v_2}, c_{t_2}) < d(c_{u_1}, c_{t_1}) + d(c_{u_2}, c_{t_2})$. That is, an edge represents progress towards the target state (c_{t_1}, c_{t_2}).*

The steps in computing a solution are:

1. Computing the minimal path regions
2. Constructing the minimal state graph
3. Searching the graph

Actually, the state graph can be constructed implicitly as part of the search step. In this way an implementation may be fortunate and not have to construct the entire state graph.

Although not affecting the correctness of the algorithm for the sake of efficiency we make certain assumptions about the data-structures used to store the graph. We assume that there is a fast (constant time) access to the following:

- The cell at specified coordinates
- The neighbours of a given cell
- Properties associated with a given cell
- Interior/exterior status of a cell — Is the cell in the interior of the region?

Fast map structures such as hash-tables should give these properties. If this is not desirable for some reason, other structures can be used because the computational costs of our algorithms will be expressed in terms of the above operations. We will use the notation *cell1.prop* to refer to property "prop" of cell "cell1".

10.2.1 Computing Minimal Path Regions

This step involves computing not only the cells which lie on a minimal path but their minimal distance from the target cell. One way to compute this is by *flooding*, that is a breadth first search of the interior starting from the destination. Each cell is marked with the number of the first iteration in which the cell is encountered. This serves as the distance from the target. The minimal path region can then be computed by travelling down the gradient from start to target. For example, see LaValle [Lav06].

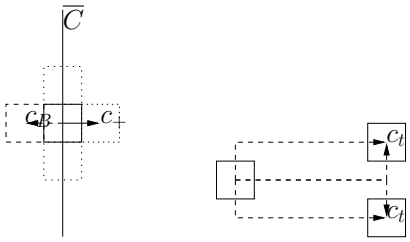
In the worst case, the time complexity is $O(\Lambda + I(\|R(c_s, c_t)\|)$ where $I(r)$ is the cost of initialising the structure to store the region and perform r successive inserts into it. The size of the region $R(c_s, c_t)$ is bounded below by $d(c_s, c_t)$ and above by $d(c_s, c_t)^2$.

The cost to compute the minimal path region can be reduced if extra information is known and the polygon is known to be genus zero. If a minimal length path (μ) from c_s to c_t is given, then the cells in the region (and their distances to target) can be calculated in $O(I(\|R(c_s, c_t)\|)$ time. Alternatively, if the Euclidean minimal path between the cell centres is given, then the region can also be calculated in $O(I(\|R(c_s, c_t)\|)$ time. This is because a discrete path can be computed in time linear in its length by following the Euclidean minimal path.

For the remainder of this section, we will assume the environment is genus zero. First, we discuss computing the minimal path region from a minimal discrete path.

Definition 51 *A cell c_+ is a* successor *of cell c on a minimal path if c_+ is a neighbour of c and $d(c_+) < d(c)$.*

For a cell $c_i \neq c_t$ in a minimal path we know the direction of the next cell on the path. In order to construct the minimal path region, we need to search for alternative minimal paths through c_i. However, we wish to limit the number of directions to be tested. In the following lemma, we show that in a genus zero environment, no minimal path through a cell can be more than 90° on either side of any other minimal path. That is, the turn to reach



(a) (b)

Figure 10.6: From Lemma 57 (4-connected movement). These figures are oriented so that α is horizontal.

any successor cannot be greater than 90° from the turn to reach the successor on the known path.

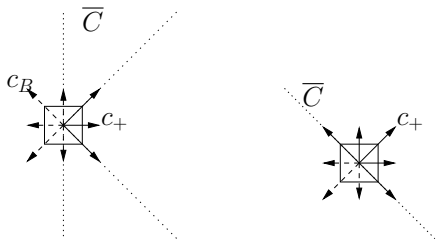
Lemma 57 *For a minimal path μ in a genus zero simple discrete polygon, if μ leaves the current cell at angle α, then no cells at angles outside $\alpha \pm 90°$ can be successors on any minimal path.*

Proof: Let c_+ be the successor in direction α and $\overline{C}$ be the chord through the current cell at 90° to α. First consider the 4-connected case. We proceed by contradiction. Suppose there is a successor c_B in direction $\alpha + 180°$ (this is the only angle which falls outside the range). Rotate the instance so that the step in μ to c_t is horizontal and refer to Figure 10.6(a). The target cell cannot lie on the same side of $\overline{C}$ as c_B since that would mean that any minimal path through c_+ would need to cross $\overline{C}$. In that case, a shorter route would be to travel along $\overline{C}$, this would mean c_+ was not on a minimal path, which is a contradiction.

This same reasoning shows that c_B cannot be on a minimal path since any such path would need to turn back to reach $\overline{C}$ again. So we have our contradiction and c_B cannot be a successor.

To see that the full range is necessary see Figure 10.6(b). In this case, the target cell could be in either of the indicated positions.

Now the 8-connected case. Some care is needed here because we will consider two separate subcases. We will give different treatment depending on whether the direction of the next step is a multiple of 90° (that is: N,S,E,W) or not. Again we use contradiction. Suppose c_B is a successor but is outside the range $\alpha \pm 90°$. This needs different treatment depending on whether or not α is a multiple of 90°.



(a) (b)

Figure 10.7: Possible directions (for 8-connected movement) in the range are shown unbroken. Broken lines indicate impossible directions.

First, suppose α is a multiple of 90°. See Figure 10.7(a), where α is shown as a move to the East. Even though cells to North and South are within the range they are not possible successors. The target cell cannot lie on either of the chords at $\alpha \pm 45°$ since there is only one minimal path in such cases (along the diagonal). So c_+ would not be a successor. Any "minimal" path including $\alpha \pm 90°$ would need to cross a diagonal in order to reach the same partition as c_+. Hence those cells cannot be minimal either. The possibility that the target cell does not lie between the two diagonal chords is excluded because it would render c_+ a non-successor. Note that crossing a diagonal chord does not necessarily mean including a cell from the chord. The crossing path could move between the cells of the chord.

Figure 10.7(b) illustrates a case where α is not a multiple of 90°. Without loss of generality assume the instance is oriented as shown in the figure. The target cell cannot lie on the chord $\overline{C}$ since in that case there would always be a shorter path than any path involving c_+ by moving directly along the chord. Similarly the target cannot lie below $\overline{C}$, since any path involving c_+ would need to cross the chord to reach it. This means that the target cell must be above $\overline{C}$ and so by the same reasoning, no cell below the chord can be a successor.

An instance where the full range must be checked is shown in Figure 10.8 (page 200).

□

We now present an algorithm to produce the minimal path region from a minimal discrete path μ in a genus zero environment. This operation has some resemblance to *fill* algorithms in computer graphics (in particular those which grow from a seed point) [Ago04, LH87]. However, in our case, there is no fast test for a cell being on the border of the region, so methods which rely on walking in one direction until the border is encountered are not applicable.

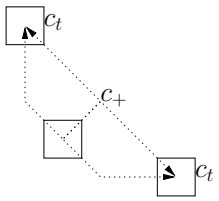


Figure 10.8: The target cell could be in either of the indicated locations. Possible minimal paths from the current cell to targets are shown dotted.

In our algorithm, cells in the interior of the polygon belong to one of three classes *good, bad* and *unmarked*. Initially, cells are assumed to be unmarked. The algorithm will store, for each cell, the minimal distance to the target as well as a link to the previous cell in a minimal path back to the start cell (termed the parent pointer). This algorithm uses two stacks labelled S and T. Stack S holds cells which might be part of the minimal path region. The algorithm walks the current border of the region pushing cells onto stack T. If at least one new minimal length path can be found using cells in T, then those cells will be added to S for later processing. If a cell in T does not form part of a minimal path, then it is marked as bad. So the region grows outwards from the original path. At the end of the algorithm all cells in the minimal path region will be marked as good.

First, the cells in μ are marked as good, their distance is calculated and parent pointers set to their predecessors in the path. The cells are then pushed onto the stack c_s first. That is, c_t is on top of S and c_s is on the bottom.

While S is not empty, pop cell c. There are three possibilities:

- Cell c is marked good.

  Push possible successors (based on Lemma 57) which are unmarked onto S. The order in which cells are pushed is significant here, cells which make smaller angles to known paths are pushed later. The parent for the added cells attribute is set to c.

- Cell c is marked bad. In this case no action is taken.

- Cell c is not marked (that is, it is unknown).

  At this point stack T will be empty. Beginning with c, the algorithm walks the border of the good cells pushing them onto T (for an example of a border following algorithm

see Agoston [Ago04]). The direction to travel from c can be determined from the parent of c. Each cell's distance is set based on the distance value of c's parent.

While T is not empty, pop cell c_p.

- Cell c_p has a neighbour c_N such that c_N is marked good and c_p.distance = c_N.distance+1.

  In this case, pop each element from T, mark it as good and push it onto S.

- Cell c_p has no such neighbour.

  In this case there is no path through c_p which can reach the target in minimal length. Mark c_p as bad. The loop continues since there may be a cell earlier in the path which can connect to a minimal path.

Lemma 58 *The algorithm given above marks all cells in the minimal path region but no others as good.*

Proof: We need to show that if a cell is marked good then it is in the region and the reverse, that all cells in the region are so marked.

If a cell is marked good, the algorithm has found a minimal path to the target which runs through it (the path is encoded in the parent pointers). So cells marked as good must be in the minimal path region.

Now we consider the implication in the other direction. We will do this by contradiction. We assume that there must be at least one *missing* cell, that is, a cell in the region which is not marked good. Let c_B be a missing cell such that there are no other missing cells closer to c_s. Since c_s is marked good in the first step of the algorithm, c_B cannot be c_s. If c_B did not have a good predecessor, then either c_B is not in the region or there is a missing cell which is closer to c_s.

So we have a c_B which has a good predecessor. But all marked cells have their possible good neighbours added to S so c_B must have been added to S. To not be marked good, walking the border of known good cells does not find any place to connect with the other minimal paths. So if there is a minimal path through c_B it would need to turn further out. The only way this could yield a minimal path would be if the region looped back and was not genus zero. Since the polygon is genus zero the minimal path region must be genus zero as well.

So we have our contradiction, hence all cells in the region are marked.

□

Now the complexity of this algorithm.

Each cell is added to each of the stacks at most once. Cells are only added to S when they are adjacent to cells known to be in the region. Hence the total number of cells processed will be the number of cells in $R(c_s, c_t)$ plus the number of cells in the boundary of the region. Denote the number of cells in the region as A.

Each prospective cell must be tested to determine if it lies in the interior of the polygon. As previously stated, though, we assume that there is a function already prepared for this purpose so it can be performed in constant time.

The other operations performed on the cells are property lookups (and tests to determine if a cell is marked good or bad). Here we consider two possibilities. First, there is constant time access to each cell in the interior using its coordinates, and each cell stores the properties required. This assumes that only one copy of the algorithm runs at a time and that there is some mechanism for distinguishing properties from different runs. A counter or timestamp would fulfil this requirement. Under these conditions the complexity will be $O(A)$ time. The space used to execute the algorithm will be $O(A)$ but it should be noted that this assumes a structure storing the cells already exists.

An alternative is to rely only on polygon interior tests and store the rest of the information separately. Under these conditions we require a datastructure with fast membership tests and property lookups. A hashtable of the appropriate size with a good hash function would do for this. The difficulty here is that A is unlikely to be known ahead of time. Even if it were known, finding a good hash function may not be easy.

Since we do not know A ahead of time, the best we can do is to use a table of size $O(\|\mu\|^2)$. In that case we could use $h(x, y) = (x \bmod \|\mu\|) \cdot \|\mu\| + (y \bmod \|\mu\|)$ as a hash function. Since each cell maps to a unique location this is a "good" hash function [GBY91]. This gives time and space complexity of $O(\|\mu\|^2)$.

Our paper describing some of this material [FEC07] claims $O(A)$ time using a hash function but it fails to account for the time required to initialise the storage required. However, the same algorithm can be used with a different data-structure to give $O(A)$ time. The cells (and their properties) of the region can be stored in a graph indexed by a dictionary on cell coordinates. Since cells are only added adjacent to existing cells, all the neighbours of

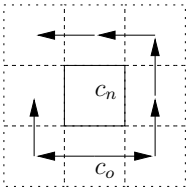


Figure 10.9: All the neighbours of cell c_n can be reached by following neighbours around the cell starting from (for example) c_o.

the new cell can be linked to it in constant time by following the links around the cell (See Figure 10.9).

10.2.2 Constructing the State Graph

The graph can be constructed explicitly or "on demand" as exploration is carried out in the next section.

To construct the graph explicitly, consider each pair of cells in $(c_1, c_2) \in R(c_{s_1}, c_{t_1}) \times R(c_{s_2}, c_{t_2})$. If c_1 sees c_2 add the pair to the graph as a node. For each node in the graph consider each of its possible neighbour states. If the neighbour state is in the graph and has a lower total distance than the node, add an edge to it. If transitions count 1 where a single agent moves and 2 where both move, then the count for a minimal path through the state graph will equal $\|\Pi_1\| + \|\Pi_2\|$.

So the graph can be constructed in time proportional to the time required to perform $\|R(c_{s_1}, c_{t_1})\| \cdot \|R(c_{s_2}, c_{t_2})\|$ visibility tests (once the regions are known).

10.2.3 Searching the Graph

If the minimal state graph has already been constructed, then it can be searched using breadth-first search. A path to the target state (c_{t_1}, c_{t_2}) represents a schedule. Each state transition is a move in the schedule. We can determine if such a path exists, and produce it, in time linear in the number of nodes in the minimal state graph.

If the graph is not constructed yet, add the state (c_{s_1}, c_{s_2}) to the graph. Perform breadth-first search from this node. Whenever a node is processed, its neighbour states can be tested for visibility and if appropriate, added to the graph.

This has the same overall complexity as the previous version but could lead to faster running times in practice since it is possible that not all states in the graph are required in order to find a path.

10.2.4 Complexity

In summary, given the constant time tests listed at the beginning of Section 10.2, we have:

- A minimal path can be found in $O\left(\min(\Lambda, \|\mu\|^2)\right)$ time in the worst case. If a Euclidean minimal path is known and the polygon is genus zero this can be done in $O(\|\mu\|)$ time.
- If the polygon has genus greater than zero, then the shortest path region R can be computed in $O\left(\min(\Lambda, \|\mu\|^2) + I(\|R\|)\right)$ time (where $I(x)$ is the cost of performing x inserts into the region structure, including initialisation). If the polygon is genus zero then regions can be computed from paths in $O(I(\|R\|))$ time.
- The state graph can be constructed in $O(\|R(c_{s_1}, c_{t_1})\| \cdot \|R(c_{s_2}, c_{t_2})\|)$ visibility tests.
- A path can be found through the graph in time linear in the size of the minimal state graph.

If no ideal solution exists then a non-ideal solution may be required. In that case the approach given here still works, but the state graph will be larger (up to $O(\Lambda^2)$).

Next we consider a subset of instances which can be solved more efficiently.

10.3 Instances With Aligned Endpoints

In this section we consider a subset of 4-connected instances.

Definition 52 *Two cells are* aligned *if they are mutually visible and share either a horizontal or a vertical coordinate. An* aligned instance *is a 4-connected genus zero DMVPP such that:*

- *The cells c_{s_1} and c_{s_2} are aligned,*
- *The cells c_{t_1} and c_{t_2} are aligned,*
- *The sightlines S and T do not cross.*

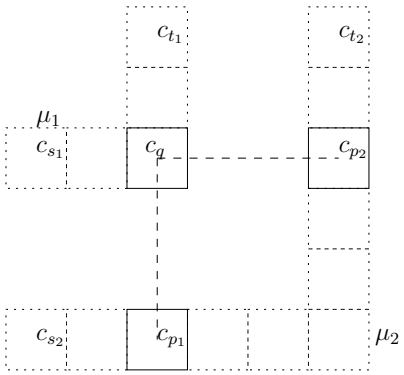


Figure 10.10: Point q is aligned with two points on μ_2, p_1 and p_2.

Definition 53 *A cell q on μ_1 is a complete corner if there are cells p_1, p_2 on μ_2 such that q is vertically aligned with p_1 and horizontally aligned with p_2. See Figure 10.10 for an example.*

Our paper on this topic [FEC07] gave a linear time algorithm for producing schedules for aligned instances. That version relies on having knowledge of all complete corners in linear time. That paper does not explain how this can be achieved. Here we give an algorithm which does not rely on knowledge of corners ahead of time. The algorithm described here can derive such information.

Lemma 59 *Consider a complete corner in an aligned instance formed by $c_q \in \mu_1$ and c_{p_1}, $c_{p_2} \in \mu_2$. All cells in μ_2 between c_{p_1} and c_{p_2} are visible to c_q.*

Proof: Refer to Figure 10.10. Since $\overline{c_q\, c_{p_1}}$ and $\overline{c_q\, c_{p_2}}$ are sightlines, the only way vision could be blocked to any point on a minimal path is if μ_2 moved behind an obstacle. See Figure 10.11 (page 206) for an example. The figure also shows why such a path cannot be minimal. Since the environment is genus zero, no obstacles can be in the region bounded by $\overline{c_{p_1}\, c_q}$, $\overline{c_q\, c_{p_2}}$ and μ_2. This means that there is no reason for μ_2 to reverse its direction (moving South after already having moved North in the figure). But a path which reverses direction is the only way for an obstacle to obstruct vision. Hence, all cells between c_{p_1} and c_{p_2} must be visible to c_q or μ_2 is not minimal.
□

Lemma 60 *In an aligned instance, any pair of minimal paths has a schedule.*

Proof: The invariant of this algorithm is that after each step the agents are visible and aligned. We need to show an algorithm which preserves this invariant and can always make

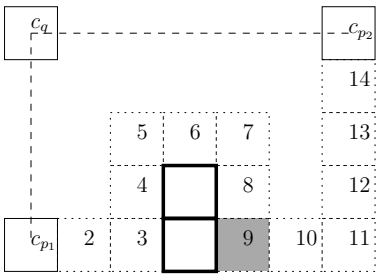


Figure 10.11: A example of a non-minimal path (numbered cells from c_{p_1} to c_{p_2}) moving behind an obstacle. Sight between c_q and the shaded cell on μ_2 is blocked by the boundary(shown by bold lines).

progress. The algorithm works as follows. At the beginning of a step we have the following possibilities. It will test each of the conditions numbered next, one at a time and in the order presented. When one matches, it applies the action and goes back to the top of the list. This list is exhaustive so one of the conditions must match.

1. One of the agents has reached their target.

2. The agents coincide.

3. One or both of the agents can move and retain the current alignment. For example moving horizontally if the agents are currently aligned.

4. Both agents can step in the same direction (orthogonal to the current alignment).

5. Both agents can step in opposite directions (orthogonal to the current alignment). For example, one agent moves North and the other agent moves South while they are horizontally aligned.

In Item 1, without loss of generality, suppose a_1 has reached its target. Since a_1 must be aligned with t_2 we ask how the targets are aligned? If the targets are aligned in the same orientation as the two agents, then the target must lie on the line through a_1 and a_2, so a_2 can move there directly without concerns about visibility. If the alignment of the targets is orthogonal to the current alignment, then a_2 can be moved directly to its target (by Lemma 59). So once one agent reaches its target, the other agent can be moved directly to its target.

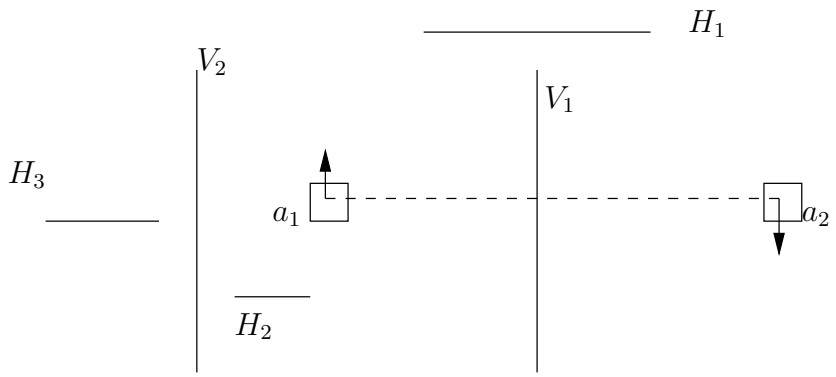


Figure 10.12: Possible positions for T labelled as H_1, H_2, H_3, V_1, V_2, symmetrical possibilities are not shown.

In Item 2, if both agents' next move would leave them aligned, then both moves can be safely made. If the next moves are orthogonal, then move one of the agents only, it does not matter which one. This preserves the invariant and leads to one of the other cases.

Item 3 preserves the invariant since movement is along the sightline.

For Item 4, since the agents move in the same direction we can be sure that they will be aligned. We need to be sure that they will be visible. An obstacle blocking sight for the new positions would force at least one of the paths to move around the obstacle. That is, they would need to move back between the agents' current positions and hence would be non-minimal.

For Item 5, if both agents move, then they will no longer be aligned so some care must be taken. To see what action to take, consider the possible positions of the target sightline T. Since the endpoints are aligned, T must be horizontal or vertical. Refer to Figure 10.12 which shows possible positions relative to the current agent positions.

- H_1: Horizontal T between the current points — This is not possible since one of the agents (a_2 in the figure) has made a non-minimal movement.

- H_2: Horizontal T with at least one endpoint outside both a_1 and a_2 — This is ruled out due to the same problem as H_1.

- H_3: Horizontal T on the same level as the current agents — If there is no block between one of the agents and H_3, then both agents have made non-minimal movements. If there is a block, then one of the agents has still made a non-minimal movement. If that were

not the case, then the paths would enclose the obstacle, which is a contradiction with genus zero.

- V_1: Vertical T between the agents — Seeing that this leads to a contradiction takes some care. We can say immediately that a_2 would need to travel to the South end of V_1 while a_1 travels North (if not they have made non-minimal moves). Further we can say that the agents are not currently at their starting points (since we know S and T do not cross). Next we consider the orientation of S. If S were vertical, then at least one of the agents would have been vertically aligned with V_1 in the past to reach its current position (this means the path of that agent must not be minimal). This leaves a horizontal S. If S lies on the chord through a_1, a_2 then it must lie on one side of V_1. In that case no minimal path can include points on the far side of the chord through V_1. This is infeasible. If S lies above the chord through a_1 and a_2, then it must have its endpoints on opposite sides of the chord through V_1 or else one of the agents cannot reach their current position and remain minimal. But in such a case one of the agents would be required to move back towards the start line. There are two possible reasons for this: either the path is non-minimal (contradiction) or both the endpoints of S were in fact on the same side of V_1 (which reduces to the previous case).

- V_2: Vertical T not between the agents — In this case, one of the paths must pass the other agent's current position (in the figure μ_2 must pass a_1 to reach V_2). We just need to determine which agent is which. This can be determined by checking which agent is furthest (horizontally from T), the closer agent stays fixed while the farther agent moves until it is aligned with a_2 (visibility is preserved by Lemma 59). The horizontal distance test is reliable because, endpoints of T must lie on both sides of the horizontal chord through a_1 and a_2. If this were not the case, then one of the agents would have made a non-minimal move.

So the last case is in fact the only possible one. This algorithm requires $O(\|\mu_1\| + \|\mu_2\|)$ time.

□

Chapter 11

Conclusions and Discussion

A summary of the notation in this thesis is available in Appendix A (page 227). An index of terms can be found on page 215.

Main Lessons from this Research

Lesson 1:

The requirement for a single fixed speed is feasible for many instances. In cases of severely restricted visibility though, it complicates matters significantly. In the two-agent case good solutions are still possible, but this is not necessarily true in cases with more agents. There are algorithms to test for difficult cases efficiently.

Lesson 2:

The crossing shape from the two-agent case is the cause of the majority of problems in MVPP-n instances. Apart from possible interference in type-D instances, it is the only thing to force the use of approximate paths rather than Euclidean shortest paths.

Lesson 3:

Methods for solving MVPPs can be employed in solving MVPP-n's but there is a much larger number of special cases to consider.

Lesson 4:

The algorithm for solving DMVPPs is simpler than for MVPPs and generalises more easily to n-agents. However it is not necessarily more efficient.

Summary of Contributions

In this thesis we describe three types of problem instances for point-agents moving to preserve visibility: MVPPs, MVPP-n's and n-agent MVPPs with non-overlapping hulls. We also describe two problem instances for non-point agents: DMVPPs where the agents occupy cells in a discrete environment and Body MVPPs where the agents are represented by circles.

MVPPs consist of two-agents moving in a genus zero polygon. Each agent can move either at a fixed speed (common to both agents) or remain stationary. MVPP instances are divided into two types crossing (S and T cross) and non-crossing (S and T do not cross). For non-crossing instances, we give an $O(|\Pi_1|+|\Pi_2|)$ time algorithm for producing ideal solutions. We give an $O(\mathcal{V})$ time algorithm to determine whether start (or target points) have non-trivial visibility of each other. If this is the case, the instance admits an ideal solution. If only trivial visibility exists between the endpoints, then we give an algorithm to construct replacement (x-restricted) paths in time linear in the number of segments in the path. These paths can be made to have length arbitrarily close to that of the path they are replacing. Regardless of whether shortest paths or x-restricted paths are in use, a schedule can be produced in $O(\mathcal{V}+\|\pi_1\|+\|\pi_2\|+\mathcal{O})$ time, where $\|\pi_1\|$ and $\|\pi_2\|$ are the number of segments in the paths and $\mathcal{O}$ is the size of the schedule.

MVPP-n's consist of n agents with their start and target points collinear. We give two possible constraints on the graph of pairwise visibility relations. Under the *complete visibility constraint* the graph must be complete. Under the *connected visibility constraint* the graph must be connected (it need not be minimally connected). As with MVPPs, instances are categorised as crossing or non-crossing. In the non-crossing case, bounding paths can be computed in $O(n\log n+(\mathcal{I}+\mathcal{L})\log(\mathcal{I}+\mathcal{L}))$ time or $O(n^2+n\mathcal{L})$ time in the worst case (where $\mathcal{I}$ is the number of segment intersections). Once the bounding paths are known ideal solutions can be computed in $O\left((\|\pi_1\|+\|\pi_2\|)n+\mathcal{L}\right)$ time. We also show that if the agents are allowed to choose two speed ratios, then all MVPPs admit ideal solutions.

For the crossing case under the connected visibility constraint, we show how to produce ideal solutions for all instances except those which require a simultaneous start in Sector A and Sector B. For those instances, replacement paths may be computed in the same way as for MVPPs. This gives a solution which is an arbitrarily accurate approximation to the ideal.

The crossing case under the complete visibility constraint is complicated by two factors. First, cases which require a simultaneous start in Sector A and Sector B might not have solutions which approximate the ideal. Second, the Starting and Finishing tasks could interfere. We show how to determine if the first factor applies in $O(\mathcal{V} + n^2)$ time. Once a set of x-restricted paths which satisfies the Starting and Finishing tasks is known, we show how to compute a schedule using those paths provided there is no interference.

We also examined a class of n-agent instances where the start and target hulls were disjoint and separated by non-crossing chords S and T (these chords must not cross). We show that all such instances admit ideal solutions under the complete visibility constraint. Under the connected visibility constraint, all instances consisting of non-self-intersecting paths through the start and target graphs admit ideal solutions.

Body MVPPs are two-agent instances where the agents are represented by circles. We show that instances exist which have no solution at all. We also show that, for instances where the sightlines do not enter the exclusion zone we show how to produce ideal solutions.

In DMVPPs, agents occupy cells in a grid which may contain holes. We discuss both 4-connected and 8-connected movement restrictions. Once shortest paths are known, the shortest path regions can be computed in $O(\|\mu\|)$ time (for genus zero polygons) or $O\left(\min(\Lambda, \|\mu\|^2) + I(\|R\|)\right)$ time (for polygons with holes). An ideal solution can then be found (if one exists) in time proportional to the time required for $O\left(\|R(c_{s_1}, c_{t_1})\| \cdot \|R(c_{s_2}, c_{t_2})\|\right)$ visibility tests. We give a variation on this algorithm to produce solutions if the polygon is not described in grid form. We also showed that for 4-connected aligned instances, ideal solutions can be produced in time linear in the length of the paths.

Discussion

This research raises the following questions for further investigation.

The simultaneous start problem (from type-A instances) complicates the task of finding solutions which approximate the ideal in type-A, type-C and type-D MVPP-n instances. It would be interesting to know if the inequalities in Chapter 4 always have a solution. If there are instances which do not have a solution, then the positions of the start points could be varied. This leads to the question of how the cost of solutions produced in this way would compare with the ideal.

In Chapter 6 we consider the phenomenon of interference between the Starting and Finishing tasks. In that case we show how to produce ideal solutions when interference occurs. However, we still do not know what types of interference can occur in type-D instances and if there are any, how to produce schedules under those conditions.

From Section 8.1, we know that two speed ratios are sufficient to produce ideal solutions for all two-agent MVPP instances. Is this also true for MVPP-n instances? That is, are two speeds for each agent sufficient to allow ideal solutions for all MVPP-n instances?

We have shown that some body MVPPs admit ideal solutions while others have no solution at all. Is there an efficient test to determine this? For example, is there any other shape apart from the one shown in Figure 9.3 which rules out any solution? Do all solvable body MVPPs admit ideal solutions or are there some which require approximations?

What impact would changing the body shape of the agents have? For non-circular bodies rotation becomes an issue, particularly if the agents cannot turn on the spot. To what extent would this complicate generating schedules?

In Section 8.2 we show that there are ideal solutions where none of the paths through the start graph self intersect. In cases where a ζ does self-intersect, does an ideal solution exist? Can every such ζ be untwisted safely?

We have not investigated (non-collinear) MVPP-n instances where the start and target hulls intersect or instances where S and T cross. What can be said about the existence of solutions in such cases?

We have shown how to produce a solution to a DMVPP based on a continuous Euclidean shortest path. If MVPPs were discretised, how would the path lengths for schedules produced for the DMVPP compare with those for the original MVPP?

In discrete settings, small changes in direction are sometimes required to keep travelling on a specified (continuous) heading. Can solutions be found which minimise the number of turns in the paths?

What impact would limiting visual range have on the problem types described in this thesis? The algorithm for DMVPPs is expressed in terms of visibility tests, if a pair of cells are not mutually visible, then they are not added to the state graph. The reason why they are not visible (blocked by an obstacle vs too far apart) is not relevant to the algorithm. In the continuous case, however, our proofs rely on certain geometric properties of the paths (such as convexity). What extra conditions (if any) need to be placed on the chosen paths to

ensure a schedule exists? Another variation of this type would be to place a lower bound on distance at which agents are visible to each other. This would be significantly more difficult, however, since it could render instances, where the paths coincide, unsolvable.

We have discussed instances which require the visibility graph to be either connected or complete. Other constraints on the graph could be investigated. For example, all nodes could be required to keep a single designated node in sight at all times. We could require that the graph be bi-connected or have some maximal diameter. Another possibility would be to allow the graph to be disconnected a small number of times. This would be particularly applicable to cases where the line of sight is being considered as a model for communication.

The environment and its constraints could also be varied. The exclusion zone could be modified to allow Dubins curves or more general shapes. Could schedules be produced which minimised or maximised the area occupied by the agents? The risks might be higher in having agents spread out (when modelling communication, this might lead to a less resilient network). Alternatively, it might be desirable not to have too many agents in close proximity (due to communication interference or other types of disruption). Could schedules be produced which minimised or maximised the area of the polygon visible to the agents at any point in time? Maximal areas could be applied to guarding problems, whereas minimal areas might be of interest to reduce disruption.

It might also be desirable to be able to impose different constraints on different parts of the environment. For example, a difficult area of the environment might require some minimal number of agents to be present at all times. On the other hand, structural weaknesses or interference might impose an upper limit on the number of agents in a region.

Finally, are there on-line algorithms with constant competitive ratios for any of the MVPP varieties described in this thesis?

Index of Terms

Bibliography

[AdBvdS+99] B. Aronov, M. de Berg, A. F. van der Stappen, P. Švestka, and J. Vleugels. Motion planning for multiple robots. *Discrete and Computational Geometry*, 22(4):505–525, December 1999.

[Ago04] M. K. Agoston. *Computer Graphics and Geometric Modeling: Implementation and Algorithms*. Springer, 2004.

[AGR01] N. M. Amato, M.T. Goodrich, and E.A. Ramos. A randomised algorithm for triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, 26(2):245–265, 2001.

[AHU74] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[Alb03] S. Albers. Online algorithms: a survey. *Mathematical Programming*, 97(1/2):3–26, Jul 2003.

[AMOT90] R. K. Ahuja, K. Mehlhorn, J. B. Orlin, and R. E. Tarjan. Faster algorithms for the shortest path problem. *Journal of the ACM*, 37(2):213–223, 1990.

[Bal99] T. Balch. The impact of diversity on performance in multi-robot foraging. In *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents*, pages 92–99, New York, NY, USA, 1999. ACM Press.

[Bar98] M. Barbehenn. A note on the complexity of dijkstra's algorithm for graphs with weighted vertices. *IEEE transactions on computers*, 47(2):263, 1998.

[BBC+02] P. Bose, A. Brodnik, S. Carlsson, E. D. Demaine, R. Fleischer, A. López-Ortiz, P. Morin, J. I. Munro, and T. Tokuyama. Online routing in convex subdivi-

sions. *International Journal of Computational Geometry and Applications*, 12(4):283–295, 2002.

[BC00] A. Blum and P. Chalasani. An online algorithm for improving performance in navigation. *SIAM Journal on Computing*, 29(6):1907–1938, 2000.

[BDH+04] P. Bose, E. D. Demaine, F. Hurtado, J. Iacono, S. Langerman, and P. Morin. Geodesic ham-sandwich cuts. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 1–9, New York, NY, USA, 2004. ACM Press.

[BK07] J. Backer and D. Kirkpatrick. Finding curvature-constrained paths that avoid polygonal obstacles. In *SCG '07: Proceedings of the twenty-third annual symposium on Computational geometry*, New York, NY, USA, 2007. ACM Press.

[BM04] P. Bose and P. Morin. Online routing in triangulations. *SIAM Journal on computing*, 33(4):937–951, 2004.

[BSZ06] A. Blum, T. Sandholm, and M. Zinkevich. Online algorithms for market clearing. *Journal of the ACM*, 53(5):845–879, Sep 2006.

[CE92] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39(1):1–54, 1992.

[Cha91] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geomrety*, 6:485–524, 1991.

[Chv75] V. Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory (Series B)*, 18:39–41, 1975.

[CJN99] S. Carlsson, H. Jonsson, and B.J. Nilsson. Finding the shortest watchman route in a simple polygon. *Discrete and Computational Geometry*, 22(3):377–402, October 1999.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[CLW+07] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper. Bridging the gap between simulation and reality in urban search and rescue. In G. Lakemeyer,

E. Sklar, D. Sorrenti, and T. Takahashi, editors, *RoboCup 2006: Robot Soccer World Cup X*, Lecture Notes in Computer Science, pages 1–12. Springer Berlin / Heidelberg, 2007.

[CNVW07] S-W. Cheng, H-S. Na, A. Vigneron, and Y. Wang. Querying approximate shortest paths in anisotropic regions. In *SCG '07: Proceedings of the twenty-third annual symposium on Computational geometry*, pages 84–91, New York, NY, USA, 2007. ACM Press.

[Coh96] W. Cohen. Adaptive mapping and navigation by teams of simple robots. *Robotics and Autonomous Systems*, 18(4):411–434, Oct 1996.

[DEH+04] E. D. Demaine, J. Erickson, F. Hurtado, J. Iacono, S. Langerman, H. Meijer, M. Overmars, and S. Whitesides. Separating point sets in polygonal environments. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 10–16, New York, NY, USA, 2004. ACM Press.

[DH04] G. Dósa and Y. He. Better online algorithms for scheduling with machine cost. *SIAM Journal on Computing*, 33(5):1035–1051, 2004.

[DHS95] A. Datta, C. A. Hipke, and S. Schuierer. Competitive searching in polygons – beyond generalised streets. In J. Staples, P. Eades, N. Katoh, and A. Moffat, editors, *Algorithms and Computations, 6th International Symposium, ISAAC '95 Cairns, Australia, December 4-6, 1995 Proceedings*, LNCS, pages 32–41. Springer Berlin / Heidelberg, 1995.

[DI94] A. Datta and C. Icking. Competitive searching in a generalized street. In *SCG '94: Proceedings of the tenth annual symposium on Computational geometry*, pages 175–182, New York, NY, USA, 1994. ACM.

[DMN+06] O. Daescu, J. S. B. Mitchell, S. Ntafos, J. D. Palmer, and C. K. Yap. Approximating minimum-cost polygonal paths of bounded number of links in weighted subdivisions. In *SCG '06: Proceedings of the twenty-second annual symposium on Computational geometry*, pages 483–484, New York, NY, USA, 2006. ACM Press.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1987.

[EGHP+00] A. Efrat, L. J. Guibas, S. Har-Peled, D. C. Lin, J. S. B. Mitchell, and T.M. Murali. Sweeping simple polygons with a chain of guards. In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 927–936, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[EGS07] D. Eppstein, M. T. Goodrich, and N. Sitchinava. Guard placement for efficient point-in-polygon proofs. In *SCG '07: Proceedings of the twenty-third annual symposium on Computational geometry*, pages 27–36, New York, NY, USA, 2007. ACM Press.

[Ent] Blizzard Entertainment. StarCraft campaign editor.

[FEC05] J. Fenwick and V. Estivill-Castro. Optimal paths for mutually visible agents. In X. Deng and D.-Z. Du, editors, *Algorithms and Computation, 16th International Symposium ISAAC*, Lecture notes in Computer Science 3827, pages 869–881. Springer, 2005.

[FEC07] J. Fenwick and V. Estivill-Castro. Mutually visible agents in a discrete environment. In G. Dobbie, editor, *Procedings of the Australasian Computer Science Conference (ACSC2007)*, volume 62 of *Conferences in Research and Practice in Information Technology*, pages 141–150. Australian Computer Society, 2007.

[Fis78] S. Fisk. A short proof of chvátal's watchman theorem. *Journal of Combinatorial Theory (Series B)*, 24:374, 1978.

[FT87] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[Gam] EA Games. Command and Conquer Generals.

[GBY91] G.H. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures*. Addison Wesley, 1991.

[GO97] J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry*. CRC Press, 1997.

[GR03] Y. Gabriely and E. Rimon. Competitive on-line coverage of grid environments by a mobile robot. *Computational Geometry: Theory and Applications*, 24(3):197–224, April 2003.

[GTG06] B. P. Gerkey, S. Thrun, and G. Gordon. Visibility-based pursuit-evasion with limited field of view. *International Journal of Robotics Research*, 25(4):299–315, April 2006.

[Had94] G. Hadley. *Linear Programming*. Narosa, 1994.

[HIKK01] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. The polygon exploration problem. *SIAM Journal on Computing*, 31(2):577–600, 2001.

[HLK06] D Hsu, J-C Latombe, and H. Kurniawati. On probabilistic foundations of probabilistic roadmap planning. *International Journal of Robotics Research*, 25(7):627–643, July 2006.

[How06] A. Howard. Multi-robot simultaneous localisation and mapping using particle filters. *International Journal of Robotics Research*, 25(12):1243–1256, December 2006.

[HPS06] A. Howard, L. Parker, and G. Sukhatme. Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment and detection. *International Journal of Robotics Research*, 25(5-6):431–447, May-June 2006.

[IK92] C. Icking and R. Klein. The two guards problem. *International Journal of Computational Geometry and Applications*, 1992.

[IKK04] V. Isler, S. Kannan, and S. Khanna. Randomized pursuit-evasion with limited visibility. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1060–1069, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[IKKL02] C. Icking, T. Kamphans, R. Klein, and E. Langetepe. On the competitive complexity of navigation tasks. In *Revised Papers from the International Work-*

shop on Sensor Based Intelligent Robots, pages 245–258, London, UK, 2002. Springer-Verlag.

[JQW07] R.-Q. Yang J. Qian, J.-J. Zeng and X.-. Weng. A fire scout robot with accurate returning for urban environment. *Robotica*, 25(3):351–358, May 2007.

[KL94] L. E. Kavarki and J.-C. Latombe. Randomized preprocessing of configuration spaces for fast path planning. In *ICRA 1994*, pages 2138–2145. IEEE, 1994.

[Kle91] R. Klein. Walking an unknown street with bounded detour. In *Foundations of Computer Science, 1991. Proceedings., 32nd Annual Symposium on*, pages 304–313. IEEE Computer Society Press, 1991.

[KP00] E. Koutsoupias and C. H. Padadimitriou. Beyond competitive analysis. *SIAM Journal on Computing*, 30(1):300–317, 2000.

[KZ86] K. Kant and S. W. Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *International Journal of Robotics Research*, 5(3), 1986.

[Lav06] S. M. Lavalle. *Planning Algorithms*. Cambridge University Press, 2006.

[LH87] B. Lind and T. Hrycej. Graphics goodies number 1 – a filling algorithm for arbitrary regions. *SIGGRAPH Comput. Graph.*, 21(5):281–282, 1987.

[LH97] V.J. Lumelsky and K.R. Harinarayan. Decentralized motion planning for multiple mobile robots: The cocktail party model. *Autonomous Robots*, 4(1):121–135, March 1997.

[LS85] D. Leven and M. Sharir. An efficient and simple motion planning algorithm for a ladder moving in two-dimensional space amidst polygonal barriers (extended abstract). In *SCG '85: Proceedings of the first annual symposium on Computational geometry*, pages 221–227, New York, NY, USA, 1985. ACM.

[LSC99] J.-H. Lee, S. Y. Shin, and K.-Y. Chwa. Visibility-based pursuit-evasion in a polygonal room with a door. In *SCG '99: Proceedings of the fifteenth annual symposium on Computational geometry*, pages 281–290, New York, NY, USA, 1999. ACM Press.

[LSST06] G. Lakemeyer, E. Sklar, D. Sorrenti, and T. Takahashi, editors. *Proceedings of the Robocup 2006 Symposium*, 2006. CD Rom version.

[MCTS+07] R. Murrieta-Cid, T.Muppirala, A. Sarmiento, S. Bhattacharya, and S. Hutchinson. Surveillance strategies for a pursuer with finite sensor range. *International Journal of Robotics Research*, 26(3):233–253, 2007.

[Meh84] K. Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching.* Springer-Verlag, 1984.

[MR06] A.I. Mourikis and S.I. Roumeliotis. Prediciting the performance of cooperative simultaneous localisation and mapping (c-slam). *International Journal of Robotics Research*, 25(12):1273–1286, December 2006.

[MW90a] J.S.B. Mitchell and E.L. Wynters. Optimal motion of covisible points among obstacles in the plane. In J. Urrutia, editor, *Proceedings of the Second Canadian Conference in Computational Geometry*, pages 116–119, August 1990.

[MW90b] J.S.B. Mitchell and E.L. Wynters. Optimal motion of covisible points among obstacles in the plane. Technical Report 896, School of Operations Research and Industrial Engineering, Cornell University, 1990.

[Nil98] N. J. Nilsson. *Artificial Intelligence: A new Synthesis.* Morgan Kaufmann Publishers Inc, 1998.

[NTMR04] R. Nair, M. Tambe, S. Marsella, and T. Raines. Automated assistants for analyzing team behaviors. *Autonomous Agents and Multi-Agent Systems*, 8(1):69–111, 2004.

[O'R87] J. O'Rourke. *Art Gallery Theorems and Algorithms.* International Series of Monographs on Computer Science. Oxford University Press, 1987.

[Ove92] M.H. Overmars. A random approach to motion planning. Technical Report RUU-CS-92-32, Department of Information and Computing Sciences, Utrecht University, 1992.

[PA05] J. Peng and S. Akella. Coordinating multiple robots with kinodynamic constraints along specified paths. *International Journal of Robotics Research*, 24(4):295–310, April 2005.

[Pes04] E. Peserico. The lazy adversary conjecture fails. *Theory of Computing Systems*, 37(3):397–403, May/Jun 2004.

[PF05] D. Pellier and H. Fiorino. Coordinated exploration of unknown labyrinthine environments applied to the pursuit evasion problem. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 895–902, New York, NY, USA, 2005. ACM Press.

[PM07] V. Polishchuk and J. S.B. Mitchell. Thick non-crossing paths and minimum-cost flows in polygonal domains. In *SCG '07: Proceedings of the twenty-third annual symposium on Computational geometry*, pages 56–65, New York, NY, USA, 2007. ACM Press.

[PP03] D. T. Pham and D. R. Parhi. Navigation of multiple robots using a neural network and a petri net model. *Robotica*, 2003.

[Pri] Private communication J.R. Sack.

[PS90] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Texts and Monographs in Computer Science. Springer-Verlag, 1990.

[RD01] N. Roy and G. Dudek. Collaborative robot exploration and rendezvous: Algorithms, performance bounds and observations. *Autonomous Robots*, 11(2):117–136, 2001.

[RDM01] I. M. Rekleitis, G. Dudek, and E. E. Milios. Multi-robot collaboration for robust exploration. *Annals of Mathematics and Artificial Intelligence*, 31(1-4):7–40, 2001.

[Rob] What is robocup? http://www.robocup.org/Intro.htm Retrieved - 20070510.

[Rud74] W. Rudin. *Real and Complex Analysis*. McGraw-Hill, 2nd edition, 1974.

[Sch99] S. Schuierer. On-line searching in simple polygons. In Christensen et al., editor, *Sensor Based Intelligent Robots*, LNAI, pages 220–239. Springer-Verlag, 1999.

[SL02] G. Sánchez and J. Latombe. On delaying collision checking in prm planning: Application to multi-robot coordination. *International Journal of Robotics Research*, 21(1):5–26, January 2002.

[SRLSA06] M. Saha, T. Roughgarden, J. Latombe, and G. Sánchez-Ante. Planning tours of robotic arms among partitioned goals. *International Journal of Robotics Research*, 25(3):207–223, March 2006.

[SS83] J.T. Schwarz and M. Sharir. On the piano movers' problem: III. coordinating the motion of several idependent bodies: The special case of circular bodies moving amidst polygonal barriers. *International Journal of Robotics Research*, 2(3):46–75, 1983.

[TC03] J. Tan and G. Clapworthy. Virtual environments for internet-based robots–II: Path planning. *Proceedings of the ACM*, 91(3):389–395, 2003.

[THL98] L. H. Tseng, P. Heffernan, and D.T. Lee. Two-guard walkability of simple polygons. *International Journal of Computational Geometry and Applications*, 8(1):85–116, Feb 1998.

[Tho99] M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46(3):362–394, May 1999.

[VKSM06] G. Varadhan, S. Krishnan, T.V.N. Sriram, and D. Manocha. A simple algorithm for complete motion planning or translating polyhedral robots. *International Journal of Robotics Research*, 25(11):1049–1070, November 2006.

[Wei] E. W. Weisstein. Convex polygon. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/ConvexPolygon.html retrieved 20070509.

[Woo02] M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley and Sons Ltd, 2002.

[YJBB07] D. R. Yoerger, M. Jakaba, A. M. Bradley, and B. Bingham. Techniques for deep sea near bottom survey using an autonomous underwater vehicle. *International Journal of Robotics Research*, 26(1):41–54, Jan 2007.

[ZS06] R. Zolt and A. Stentz. Market-based multirobot coordination for complex tasks. *International Journal of Robotics Research*, 25(1):73–101, Jan 2006.

Appendix A

Notation

Symbol	Meaning
a_i	Agent i.
$\overline{x\,y}$	the line segment between points x and y.
Π_i	A shortest Euclidean path.
$\|X\|$	The cardinality of X. For discrete paths, this is the number of cells in the path. For continuous paths, this is the number of segments in the path.
$\|\Pi_i\|$	The Euclidean length of a path.
π_i	A path in a continuous setting which is not known to be shortest.
τ, τ_k	A bounding path. Subscripts indicate the sector being bounded.
μ_i	A path in a discrete environment.
c_{s_i}	A cell in a discrete environment.
Ag_k	The set of all agents which move through Sector k.
$\mathcal{L}$	The total number of segments in all paths in the sector (or instance).
${}^{ext}S$, ${}^{ext}\Pi_i$	The extension (Definition 5) of a segment or path.
Λ	The number of cells in a discrete polygon.
n	The number of agents in an MVPP-n.
$\mathcal{I}$	The number of intersections between segments.
$\mathcal{V}$	The number of vertices in a polygon.
$I(k)$	The time required to initialise a set structure and insert k items into it.
ζ	A sequence of start points in Chapter 8.