VIEW PLANNING WITH COMBINED VIEW AND TRAVEL COST

by

Pengpeng Wang

B.S., University of Science and Technology of China, 1998

M.A.Sc., Simon Fraser University, 2003

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
in the School
of
Engineering Science

APPROVAL

Name:	Pengpeng Wang
Degree:	Doctor of Philosophy
Title of thesis:	View Planning with Combined View and Travel Cost

Examining Committee: Dr. Rodney Vaughan
Professor, Engineering Science, Chair

Dr. Kamal Gupta, Senior Supervisor
Professor, Engineering Science

Dr. Ramesh Krishnamurti, Supervisor
Professor, Computing Science

Dr. Binay Bhattacharya, Supervisor
Professor, Computing Science

Dr. Thomas Shermer, SFU Examiner
Professor, Computing Science

Dr. Tetsuo Asano, External Examiner,
Professor, School of Information Science
Japan Advanced Institute of Science and Technology

Date Approved: Aug 17, 2007

Abstract

In this thesis, the problem of view planning with combined view and travel costs, denoted by *Traveling VPP*, is addressed. It refers to planning a sequence of viewpoints to fully inspect the objects of interest and a path to realize these viewpoints, while minimizing total cost, including both view cost and traveling cost. The idea of combining the two costs is motivated by robotic applications, especially the remote missions, where the time and energy spent are a critical factor to successfully complete the tasks. Travel cost is the cumulative time and energy consumption due to the robot movements and thus is proportional to the total distance traveled by the robot. View cost corresponds to the image processing, image registration and geometric model construction after each view is taken and is proportional to the number of viewpoint planned.

First, we assume that the viewpoints are given, the geometries of the object, and the graph encoding the robot paths are known (also known as the model-based case). We give an LP based approximation algorithm the solution cost of which is within a certain ratio of the optimal cost. We show that the approximation ratio is in the order of the frequency parameter, defined as the maximum number of viewpoints that see a single surface patch of the object, of the problem. Together with the poly-logarithmic approximation ratio provided by an existing LP based randomized algorithm (after reducing our problem to the related group Steiner tree problem), the best known approximation ratio to Traveling VPP is the minimum of a constant times frequency and the poly-logarithmic of the input size, which matches (with the approximation order) the existing hardness of approximation result for a closely related problem, the Group Steiner Tree problem. This result parallels that for the well-known set covering problem.

Then we introduce a geometric problem, namely the *Watchman Route Problem with Discrete View Cost*, denoted by GWRP, which refers to planning a continuous robot tour inside a polygon and a number of discrete viewpoints on it with the minimum total cost as the weighted sum of the view and travel cost, such that every point on the polygon (interior) boundary is visible from at least one viewpoint planned. The GWRP generalizes the well-known Watchman Route Problem, which refers to plan the shortest tour such that any polygon boundary point is visible to at least one point on the tour. We propose a novel sampling method to reduce any GWRP instance to a Traveling VPP instance with a bounded number of viewpoints such that the optimal solution to the GWRP is within a constant ratio of the optimal solution cost of the reduced Traveling VPP. Thus combining the approximation algorithm to the Traveling VPP, we have an approximation algorithm to the GWRP. We also implement our approximation algorithm for the GWRP that takes a polygon and a robot start point and produces a GWRP solution. We present some preliminary experimental results that show the power of the algorithm.

Acknowledgments

First, I would like to thank my PhD thesis and also Master's thesis advisor, Dr. Kamal Gupta, for his guidance, help and support over the years. He not only introduced me to the wonderful world of robotics research, but encouraged me to pursue the research directions of approximation algorithm and computational geometry.

I would like to thank my PhD advisor, Dr. Ramesh Krishnamurti for his constant support for my thesis. The fruitful discussions with him led to many results in this thesis.

I would like to thank Dr. Binay Bhattacharya and Dr. Thomas Shermer for helpful discussions during my thesis research and for reviewing and commenting on my thesis.

I would like to thank my external examiner, Dr. Tetsuo Asano for taking the time to review my thesis and giving me feedbacks. I would like to thank Dr. Rodney Vaughan for chairing my thesis defense.

I would like to thank John Gonzalez, Hatesh Radia, Dallas Thomas, and of course, our mentor, Dr. Daya Gaur, a.k.a. the 'Gang of Primal-Dual', during Pacific Institute for the Mathematical Sciences (PIMS) Graduate Industrial Mathematics Modeling Camp (GIMMC) 2005. Although just two weeks, the intense and joyful moments we had on approximation algorithm and primal-dual algorithm is going to last in memory for a long long time. Special thanks to Dr. Daya Gaur for his help and support.

I would like to thank friends and colleagues over the years for the enjoyable time I had at Simon Fraser University.

Last, I would like to thank my wife Sharon, without whom this thesis work would never have been possible. Her love and support have helped me through tough times during my thesis completion.

Contents

List of Tables

List of Figures

Nomenclature

A_{IS}, A_{ES}: An inspection action, and an exploration action.

$EE^k(v)$: The excursion edge set of a graph node v at iteration k.

ES: The exploration sensor.

FOV: The sensor field of view.

$G = (\mathcal{V}, E)$: The robot traveling graph.

IS: The inspection sensor.

P, $int(P)$, ∂P: The simple polygon, its interior and its boundary.

$\mathcal{S}$: The surface patch set.

$\mathcal{S}(i)$: The surface patch set of viewpoint i.

$\mathcal{V}$: The viewpoint set.

$\mathcal{V}(j)$: The viewpoint set of surface patch j.

$\mathcal{W}$, $\mathcal{W}_{free}$, $\mathcal{W}_{obs}$: The environment where the robot and the object of interest reside, its free part, and its obstacle part

Acronyms

AGP: The art gallery problem.

ESP: The errand scheduling problem.

GST: The group Steiner tree problem.

GWRP: The watchman route problem with discrete view cost.

SCP: The set covering problem.

SLAM: The simultaneous localization and mapping.

Traveling VPP: The view planning problem with traveling cost.

TPP: The traveling purchaser problem.

TSP: The traveling salesman problem.

VPP: The view planning problem.

WRP: The watchman route problem.

Chapter 1

Introduction

1.1 Introduction

Imagine at a historic site, a robot is asked to autonomously scan the artifacts on the site and build their complete surface representations. The robotic artifact "documentation" or virtual reality environment construction ability automates many tedious tasks done primarily by human thus far. (See Ref. [BA06] and the references therein for some of the few existing works on automating this process.) Please see Ref. [LPC+00] for an interesting work on 3D digitization of historic artifacts in which a group from Stanford University spent nine months in Italy digitalizing the sculptures made by Michelangelo. In Ref. [LPC+00], the planning and moving were done by human. Note that if using autonomous robot sensor systems, not only the time spent in moving the 3D laser scanner and planning the scanning activities can be greatly shortened, but tedious work can be avoided for human operators.

The aforementioned vision-based robotic application motivates the problem considered in this thesis: to plan sensing actions and a robot traveling path in realizing these actions such that the surfaces of the objects of interest are completely "documented"/scanned. For a solution to be feasible, it must satisfy: the *covering constraint* that for any object surface patch, at least one viewpoint where the surface patch is visible from is chosen; and the *connection constraint* that the viewpoints chosen are connected via the path planned. At the same time, for such applications, especially in remote missions, the time and energy spent are a critical factor for the tasks to be successfully completed. Thus, we model this robotic object inspection task as an optimization problem of minimizing the corresponding total cost, a weighted sum of both the view cost and the travel cost. View cost corresponds to the image processing, image registration and geometric model construction after each view is taken and is thus proportional to the number of viewpoints planned [SRR03]. Travel cost is the cumulative time and energy consumption due to the robot movements and we model it as proportional to the length of the total path the robot travels. We call this problem of view planning with combined view and travel cost, as Traveling View Planning Problem, or *Traveling VPP*. Clearly, the solution to this problem has ample applications ranging from surveillance to object inspection.

See Fig. 1.1 for a simple example where the robot-sensor system, a mobile manipulator with a range sensor mounted at the manipulator's end-effector, is required to inspect the surface of a large object. Compared with just the mobile base, the mobile manipulator gives the sensor additional degrees of freedom and maneuverability. This is illustrated in Fig. 1.1, where the robot achieves visibility by extending the manipulator over occluding obstacles. The six robot configurations that realize the planned viewpoints are also shown, and the dotted lines between these configurations denotes the traveling path, including the manipulator movements, of the robot. The dotted triangles that are attached to the robot end-effector are the sensor's field of view (FOV) at different configurations. It is clear that the solution shown in Fig. 1.1 is a successful plan that satisfies both the covering and the connection constraints. Its total cost includes the total view cost, proportional to the number of viewpoints planned (six in this case), and the travel cost, proportional to the length of the path traveled by the robot.

In this thesis, we consider two cases of the problem, namely a discrete version, where the set of possible discrete viewpoints are given as the problem input, and the geometric version in a 2D scene where the viewpoints can be any points in a polygon (possibly with holes). The latter version corresponds to the case where a mobile robot, usually modeled as a point, is asked to inspect a 2D environment.

We use the same name *Traveling VPP* for the discrete version. Traveling VPP uses a graph connecting the input viewpoints to encode the robot traveling paths for realizing the corresponding sensing actions. The use of graph makes Traveling VPP general enough for such complex robot model as the many degree-of-freedom (dof) mobile manipulator shown in Fig. 1.1, where the 1D graph structure called roadmap is generally considered as the only data structure to solve the robot motion planning problem practically [Lat91,KSLO96]. In addition, we consider a special case of Traveling VPP, in which a point robot equipped with a omnidirectional range sensor with a certain range D, is asked to inspect object

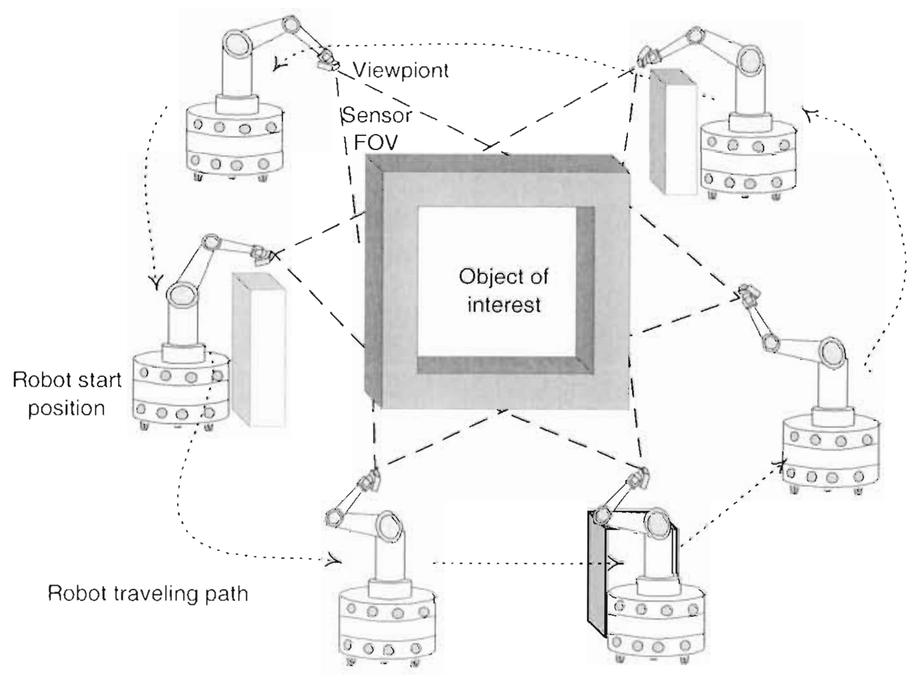


Figure 1.1: A Traveling VPP instance. It shows 6 planned sensor viewpoints that totally cover the surface of the object of interest, and the robot traveling path to realize them.

surfaces in a two- or three-dimensional environment. We call this special case *Metric View Planning Problem with Travel Cost and Visibility Range*, or *Metric TVPP* in short. Note that unlike the general Traveling VPP, for Metric TVPP, the robot travel distance between two viewpoints is the shortest path length between them. Note that the shortest path lengths are governed by a metric, i.e., they satisfy the triangular inequality.

Rather than calling the geometric version "Traveling VPP in a polygon", we use a well-known problem in the computational geometry literature, the *Watchman Route Problem* or WRP in short [CN91], which asks for the shortest path inside a polygon (possibly with holes) such that every point on the polygon boundary is visible from at least one point on the path, and call the geometric version the *Watchman Route Problem with Discrete View Cost*, or *Generalized Watchman Route Problem* (GWRP). GWRP is a true generalization to the WRP: while the WRP for simple polygons (without holes) is in **P** (solvable in polynomial time), GWRP is **NP-hard**. The latter follows from the fact that when travel cost is ignored, GWRP reduces to the well-known **NP-hard** Art Gallery Problem (AGP).

In this paper, we consider a nontrivial restricted version of the GWRP, called the *Whole Edge Covering GWRP*, or *WEC-GWRP*, in which any polygon edge is required to be entirely visible from at least one planned viewpoint. The restriction arises naturally in inspection tasks in robotic applications, where the "map" given to the robot is often a discretized boundary representation and during inspection tasks each small discretized boundary piece is considered as inspected from one planned viewpoint if and only if all the points on it are visible. Thus, by regarding each piece as a polygon edge, we have a whole edge covering instance. The same restriction is also used in the terrain guarding problem [Eid02]. WEC-GWRP has the same NP-hardness and inapproximability as GWRP. (Please see Appendix A for a quick recap of the inapproximability concept and L-reduction method to prove inapproximability result.) It is because that the reductions used for establishing the NP-hardness for Point AGP, for polygons without holes [LL86] and polygons with holes [OS83] respectively, construct *whole edge covering* Point AGP instances from an arbitrary 3-Satisfiability instance. In addition, the inapproximability result for Point AGP, i.e., Point AGP is log-inapproximable for polygons with holes, follows from the reduction from an arbitrary Set Covering Problem instance to a *whole edge covering* Point AGP instance. Formal definitions of Traveling VPP and WEC-GWRP are given in Chapters 2 and 4 respectively.

Both Traveling VPP and WEC-GWRP combine and generalize some well-known NP-hard problems and are immediately NP-hard. We adopt in this thesis approximation algorithms [Vaz01] that are fast (runs in time of the order of a polynomial of the input size) and guarantee worst-case performance as our methodology. To measure the quality of an approximation algorithm, we use the approximation ratio [Vaz01]. For our minimization problem, it is defined as the upper bound on the ratio between the algorithmic solution cost (its objective function value) and the optimal solution cost.

In the following, we survey some related works to Traveling VPP and GWRP respectively. These works span the fields of algorithmic robotics, combinatorial optimization, approximation algorithm, and computational geometry. They are organized in the following fashion. First, we discuss those closely related to the case when the travel cost is ignored. Second, we discuss those closely related to the case when the planned viewpoints are given and the view cost is ignored. Last, we discuss those related to the general problem, i.e., combining both view and travel.

1.2 Related Work to Traveling VPP

1.2.1 View Planning Problem and Set Covering Problem

Without considering travel cost, Traveling VPP reduces to that of finding the minimum number of viewpoints to cover all the surfaces of the object, and thus is closely related to the view planning problem (VPP) considered in the computer vision research area and the set covering problem (SCP) considered in the combinatorial optimization research area.

The View Planning Problem is defined formally as: given a set of viewpoints and a set of surface patches, plan the minimum number of viewpoints such that the object surfaces are totally viewed [SRR03].

In such vision applications as digitizing object geometries, usually a sensor positioning system is used in a well-controlled and limited workspace. Hence the view cost that corresponds to the image processing, image registration and geometric model construction after each view is taken dominates the effort in moving the sensor and thus realizing different viewpoints. Thus, the aim is to optimize only the number of planned viewpoints. Please see [SRR03] for a detailed survey on the VPP.

The assumption of VPP that view cost is dominant limits its possible applications. For instance, in the hazardous environment or in a large workspace, autonomous robot-sensor systems are more appropriate and the travel cost cannot be ignored. Note that the combined travel and view cost is a major consideration of the Traveling VPP.

The Set Covering Problem is defined formally as: given a universe of elements and some subsets of the universe, determine the minimum number of such subsets, the union of which is the universe [Vaz01].

The SCP is a well-know NP-complete problem and plays a central role in the combinatorial optimization area. The best known approximation algorithms for SCP proceed greedily according to the (amortized) covering cost, and have an approximation ratio of either the frequency constant, defined as the maximum number of given subsets an element belongs to, or the logarithm of the number of elements [Vaz01].

Scott *et al.* formulated the VPP into an integer linear program (ILP) [SRR00, SRR01]. Also, they claimed that VPP is isomorphic to the set covering problem below, but no concrete construction was shown. (In Chapter 2, we give reductions in both directions between VPP and SCP, thus establishing their equivalence.)

1.2.2 Metric TSP and Steiner tree problem

For the Traveling VPP, if the viewpoints to be visited are pre-determined, the problem reduces to the metric traveling salesman problem (Metric TSP). The Metric TSP is to plan a tour to visit specified vertices on a complete graph with metric. The well-known and straightforward approximation framework is to approximate the tour by solving first a shortest tree connecting these vertices and constructing a tour from the tree [Vaz01]. The construction can be either to first double the edges on the tree, and construct a tour by taking shortcuts (having a 2 approximation ratio); or to use Christofides' algorithm (having a 1.5 approximation ratio) [PS82]. Christofides' algorithm first constructs the minimum spanning tree T on the original graph; second, it computes the minimum cost perfect matching between vertices of T with odd degrees; third, it adds edges corresponding to this matching to T to make it Eulerian; last, it computes a tour on the resulting Eulerian graph [Chr76].

Planning the shortest connecting tree corresponds to another NP-complete problem, the Steiner tree problem.[1] The Steiner tree problem of planning the shortest tree connecting a given subset of all the vertices on a graph admits constant approximation ratio algorithms. These approximation algorithms generally exploit the underlying metrics of the problem, for instance, some algorithms use greedy techniques that utilize the metric information to first optimally solve the minimum spanning tree (MST) on the shortest path graph. Note that this metric information is not available for a general SCP, while it exists implicitly in the Metric TVPP due to its finite range constraint.

1.2.3 Attempts to combine both travel and view

There is some existing work on combining the view and travel cost in the robotics literature, but not in a unified and global fashion. For example, Fekete *et al.* [FKN04] and Isler *et al.* [IKD03] considered a local version of the robot exploration problem, "to look around a corner", i.e., to detect an object hidden behind a corner while minimizing the sum of the robot travel distance and the sensor scan time. The problem is considerably simpler since the goal is local, i.e., the objective is not to cover all the object surfaces.

Danner and Kavraki [DK02] considered the combined problem, however, in a "weak

[1] Please note the difference between the Steiner tree problem (connecting a subset of the vertices) and the minimum spanning tree problem (connecting all the vertices). Although MST can be solved exactly in polynomial time, Steiner tree is NP-complete.

sense", since no view cost is considered, thus corresponding to a special case of *Traveling VPP*. They proposed to solve the problem by a decoupled two-level approach, i.e., to plan the minimum number of viewpoints without considering robot travel cost first and then to solve (approximately) the Metric TSP using the shortest path graph. This two-level decoupled approach would work well for cases where the coverage of the views considered do not overlap (they become the only choice for a view plan.), or those with large coverage overlap are close to each other (they correspond to similar travel cost). In Chapter 3, we also show that for the Metric TVPP, this two-level approach achieves an approximation ratio of $O(\log m)$, m being the number of surface patches, of the same order as the inapproximability result of the problem. Intuitively, the sensor range constraint of the Metric TVPP implicitly couples the traveling and view components: in order to cover a surface patch, the robot has to travel to at least within the sensor range of it. Thus the two-level approach is no longer decoupled.

However, this is not true for a general Traveling VPP setting. For example, as shown in Fig. 1.2, even assuming that at each level the respective optimization subproblem, obtaining the minimum number of viewpoints and the shortest path tour respectively, is solved optimally, this two-level decoupled approach provides no performance bound (with respect to the optimal solution cost), and can perform arbitrarily poorly. This is easily achieved by pulling the leftmost viewpoint arbitrarily farther from the rightmost ones. This issue occurs because the planned viewpoints at the first level are too far apart for the robot to realize a plan efficiently since no travel cost is considered at the first stage.

These related works mentioned above do not address the two types of cost of different natures, thus are not applicable to solving the Traveling VPP.

1.2.4 Connected facility location problem

In an interesting work [SK04], the problem of connected facility location is addressed, which, given a set of *facilities* and a set of *clients* both residing in a metric space, asks for a set of *open* facilities connected by a Steiner tree and the *service* assignments between these open facilities and all the clients, such that the total cost, the sum of both the total service assignment cost and the tree cost, is minimized. Using the metric heuristics in their algorithm, the authors give a greedy algorithm with constant approximation ratio. By regarding the clients as the surface patches in the Traveling VPP and regarding the facilities as the viewpoints, the connected facility location problem is related to the Traveling VPP. However,

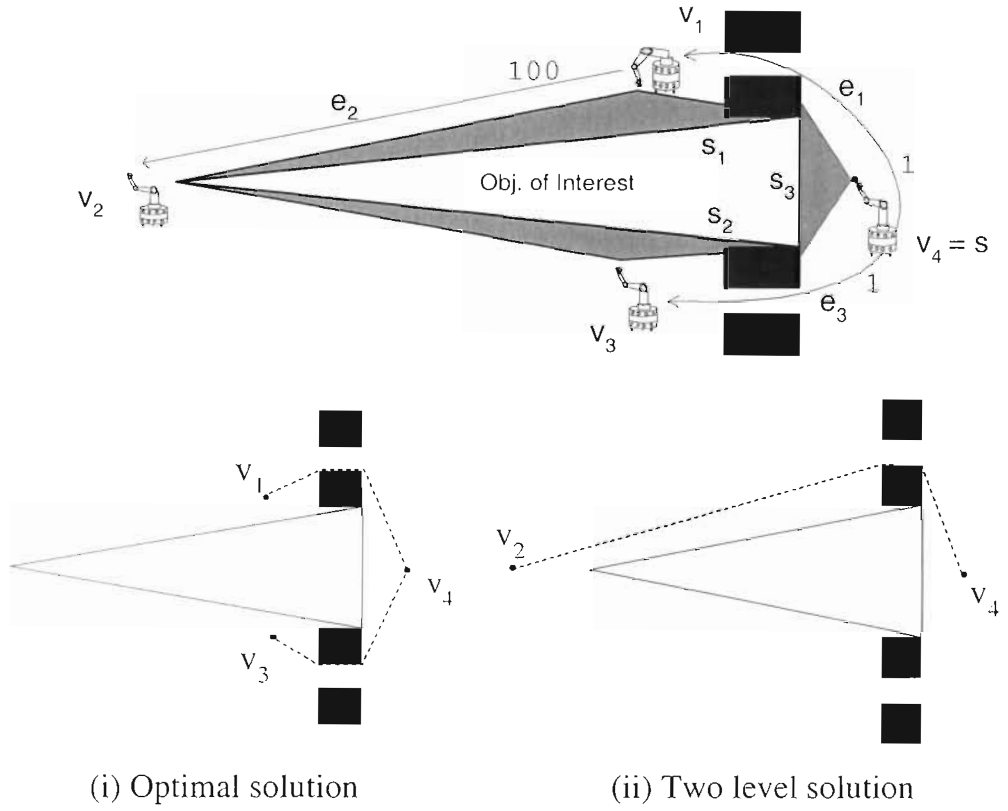


Figure 1.2: A planar example shows the arbitrarily poor performance of the two-level decoupled approach. The object to inspect, the triangle, has three surface patches, s_1, s_2 and s_3; the four possible sensing positions, or viewpoints, are v_1, v_2, v_3 and v_4, all shown in the top figure. v_4 coincides with the robot start position s. The shaded sensing triangles show the covering relations: viewpoints v_1, v_3 and v_4 cover the surface patches s_1, s_2 and s_3 respectively, while v_2 (sensing triangle of which is not shown) covers both s_1 and s_2. The line segments connecting the views, e_1, e_2 and e_3, denote the robot's traveling path; the numbers on each segment are the respective travel cost (distances). (The distances are not drawn to scale.) We assume the view and travel cost are equally weighted in the objective function, i.e. the unit view cost (cost for each view) is the same as the unit travel cost (cost for unit travel distance). Thus the total cost is the sum of the number viewpoints planned and the total distance of the path planned. The dashed lines shown in the two bottom figures are the planned paths connecting the planned viewpoints. The optimal solution is to take three views at s, v_1 and v_2 using the dashed line segments as the traveling path. The solution given by the decoupled cost can be made arbitrarily poor by pulling v_2 farther to the left.

the visibility relation between viewpoints and surface patches does not assume a metric, and the heuristic by Swamy and Kumar [SK04] is not applicable to the Traveling VPP.

1.2.5 Group Steiner tree problem, traveling purchaser problem, and errand scheduling problem

The problems of group Steiner tree (GST), traveling purchaser (TPP), and errand scheduling (ESP) are closely related to Traveling VPP.

The Group Steiner Tree Problem is defined formally as: given a graph $G = (V, E)$, where the vertex set V is divided into k distinct groups, $g_1, g_2, \ldots, g_k$, construct the minimum cost Steiner tree to connect at least one vertex from each group.

The GST generalizes both the SCP and the Stener tree problem where the best known approximation ratios are $O(\log n)$ and $O(1)$ respectively. Chekuri *et al.* [CEK06] and Garg *et al.* [GKR00] used a greedy algorithm and an LP-based randomized rounding algorithm respectively to achieve the best known poly-log approximation ratio, $O(\log |V| \log\log |V| \cdot \log k \log N)$, where $|V|$, k and N are the number of graph nodes, the number of groups and the maximum cardinality of the groups respectively. As an interesting robotic application, Saha *et al.* [SRLSA06] used the approximation GST algorithm proposed by Chekuri *et al.* [CEK06] to solve the problem of planning tours for the robot arm to achieve at least one configuration from each distinct group of configurations that achieve the same end-effector pose.

It was open whether the gap between the best known SCP and GST approximation ratios could be closed until recently, Halperin and Krauthgamer show the poly-log inapproximality hardness result for GST, i.e., the optimal solution to GST cannot be approximated by any polynomial algorithm within the $O(\log^{2-\epsilon} k)$ ratio, for any $\epsilon > 0$ [HK03]. By considering each group in any GST instance as the viewpoint set of a surface patch in the Traveling VPP, GST is reduced to a special case of the Traveling VPP where the viewpoint sets that cover different surface patches are exclusive, i.e., they do not share common viewpoints. Thus, the Traveling VPP is certainly a harder problem than GST and can not be approximated within $\log^{2-\epsilon} |\mathcal{S}|$. Please see Chapter 2 for the reduction from Traveling VPP to GST, thus showing their equivalence. It also implies that the Traveling VPP is not approximable within the same poly-log ratio.

The Traveling purchaser problem (TPP) is defined formally as: given a set of warehouses, $\mathcal{W}$, connected by a graph $G = (\mathcal{W}, E), E \subseteq 2^{\mathcal{W} \times \mathcal{W}}$, and a set of products $\mathcal{P}$ with *requirements and the prices of buying a product at a warehouse,* $d_{w,p}, w \in \mathcal{W}, p \in \mathcal{P}$*. the (unlimited capacitated) Traveling Purchaser problem (TPP) asks for certain warehouses for each product and the tour connection between the planned warehouses with the minimum total cost. the sum of the product purchase cost and the tour cost.*

The Traveling VPP is a special case of TPP where the prices are uniform for all the product and warehouse pairs. In Chapter 2, we also show that TPP can be reduced to Traveling VPP. This again shows the equivalence of the two problems.

The equivalence between GST, TPP and Traveling VPP also implies that the approximation result we develop for Traveling VPP also applies to GST and TPP.

The errand scheduling problem (ESP) is defined formally as: given a graph $G = (V, E)$ *with metrics (the edge weights of which satisfy the triangular inequality) and a set of* errands. *where each vertex is associated with a subset of these errands. plan a short tour such that the union of the subsets associated with the vertices visited is the whole set [Sla97].*

Slavik [Sla97] gives an algorithm with an approximation ratio of $3\rho/2$, where ρ is the maximum number of nodes an errand is associated with. Traveling VPP generalizes ESP in the following senses: the graph in Traveling VPP does not assume metrics; there is no view cost in ESP; and there is no distinction in ESP of viewpoint and Steiner node on the graph. In Traveling VPP, even if some viewpoints are traversed in the solution, the robot does not need to take a view of them. They are simply for travel use and do not incur view cost, hence termed as Steiner nodes [Vaz01]. Thus the results by Slavik [Sla97] do not apply to Traveling VPP. Simple reductions from Traveling VPP to ESP, for example, adding to the travel cost of each edge $e = (u, v)$ the view cost of both viewpoints u and v, do not work, since this construction requires that the robot take a view at every graph node it visits.

1.3 Related Work to GWRP

1.3.1 Art Gallery Problem

Ignoring the travel cost, GWRP reduces to the art gallery problem (AGP) considered in the computational geometry research. In the similar fashion that Traveling VPP generalizes VPP, the GWRP generalizes the AGP. We refer to [O'R87,She92,Urr00] for detailed surveys on works for AGP in the last century. Here we only mention those closely related to this thesis work, and some recent developments.

The AGP (point guard) is defined formally as: given a polygon, plan the minimum number of guards, points in the polygon (including polygon interior and boundary), such that every point on the polygon boundary is visible to at least one guard planned [O'R87].

The AGP has different versions depending upon the restrictions on the guard set. The AGP defined above is often called the *AGP with point guards*, denoted by Point-AGP, where the guards can be any point in the polygon, as opposed to the *AGP with vertex guards*, denoted by Vertex-AGP, where the guards are restricted to be placed on the polygon vertices. All these versions are NP-hard, even for simple polygons (without holes) [O'R87]. This motivates the efforts to design approximation algorithm. To the best of our knowledge the best existing approximation algorithm for Vertex-AGP has the approximation ratio of $O(\log n)$, n being the number of polygon vertices [Gho87]. There is no existing approximation algorithm for Point-AGP.

There are two major recent developments on AGP. On the complexity analysis, the Point-AGP and Vertex-AGP are shown to be APX-hard, i.e., there exists a constant $\epsilon > 0$ such that no polynomial algorithm can approximate the optimal solution within a ratio of $(1 + \epsilon)$, for simple polygons and log-inapproximable for polygons with holes [ESW01]. Note that the inapproximability result for polygon with holes is the same as that for SCP [Fei98]. And indeed the reduction used to establish the inapproximability for Point-AGP is from the SCP. On the other hand, recent efforts are made on designing approximation algorithms for some simpler versions of Point-AGP. Nilsson [Nil05] designed a constant approximation algorithm for Point-AGP for monotone polygons. A simple polygon $\mathcal{P}$ is *monotone* if there exists a line l such that the intersection of $\mathcal{P}$ and any line l' perpendicular to l is either an empty set, a single point, or a single line segment [dBvKOS00]. Constant approximation algorithms were given for 1.5D terrain guarding problem [BMKM05,Kin06]. A 1.5D terrain is a polygonal curve that is monotone w.r.t. the horizontal axis, i.e. its intersection with any vertical line is either an empty set or a single point [BMKM05]. The terrain guarding problem refers to finding the minimum number of points from a terrain such that all the points of the terrain are visible. Note that it is open whether the two above-mentioned guarding problems are in **P**. As also mentioned by Ben-Moshe *et al.* [BMKM05], their efforts to obtain an approximation algorithm for the Point-AGP have failed.

There are also related works on how to practically solve Point-AGP. Efrat and Har-Peled assume that a dense grid laid on the polygon is available and viewpoints are restricted to be grid vertices [EHP02]. Gonzalez-Banos and Latombe propose to first randomly sample the interior of the polygon; and by restricting the possible guard positions to be these samples, solve the corresponding SCP by regarding each sample as the set of the decomposed polygon boundary it covers [GBL01]. These approaches have an immediate drawback. As shown in Fig. 1.3, if the polygon has a small kernel (the set of viewpoints that sees the whole polygon boundary), taking a view at a single point in the kernel becomes the optimal solution, both methods are likely to fail: the discretization method has to make the resolution very fine; and sampling a point in the kernel becomes a rare event for the randomized sampling method.

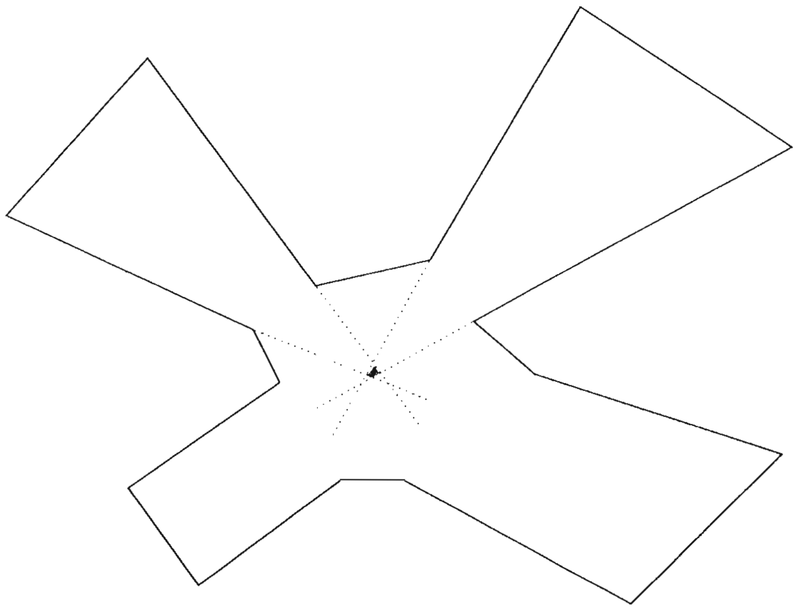


Figure 1.3: A polygon with a small kernel, the shaded region.

1.3.2 Watchman Route Problem

Ignoring the view cost, the GWRP is equivalent to the watchman route problem (WRP).

The Watchman Route Problem is formally defined as: given a polygon, plan a continuous tour in the polygon with minimum length, such that every point on the polygon interior boundary is visible to at least one point on the tour planned [CN91].

It is interesting enough that although the covering constraint is inherent in the WRP, for simple polygons, it is actually polynomially solvable [CN91, Tan04]. However, for polygons with holes (even with a bounded number), the WRP is NP-complete. Clearly, the Traveling VPP and the GWRP that consider the view cost are generalizations to the WRP. Also, unlike the WRP being restricted in 2D environment, *Traveling VPP* uses a graph to encode the traveling, thus is applicable to more general cases, for example robots with nontrivial geometries and kinematics. Note also that GWRP is not a trivial extension to the WRP in the following sense. First, unlike the WRP, the GWRP is a clear generalization to both the Point-AGP and the Metric TSP, hence is as least as hard. Second, the best WRP solution may incur an unbounded cost for the corresponding GWRP solution, i.e., infinite number of viewpoints are needed on the tour to cover the whole polygon boundary, [GB04].

1.3.3 Shortest Path Problem

Here we give a quick preview of the approach we take for GWRP. We apply a *sampling algorithm* that computes a finite number of viewpoints inside the polygon to reduce the GWRP instance to a Traveling VPP instance and apply the Traveling VPP solver to the *induced* Traveling VPP instance. This sampling algorithm utilizes the metric structure of the polygon and is related to approximation algorithms for shortest path problems, such as the unweighted shortest path problem in 3D [Pap85, MMP87] and the weighted shortest path problem[2] in 2D [MP91, SR06, AMS05]. Unlike the unweighted shortest path problem in 2D where the shortest path is encoded in the visibility graph consisting of only the polygon vertices and the start and end points of the path, for both the above-mentioned problems, the shortest path may pass through the polygon or polyhedron edges at points between the vertices and these points are hard or impractical to compute. From our point of view, these works can be categorized into those that use the principle of optimality and approximate the globally optimal path, for example, [MMP87, MP91]; and those that do not and thus approximate all the locally optimal paths. Due to the view cost and visibility constraints, only techniques of the second category can be used for WEC-GWRP. To our knowledge, there are two such works as discussed below.

Papadimitriou [Pap85] designed an approximation algorithm for the Euclidean shortest

[2]That is, a plane partitioned into regions with different weights is given and the length of the path is the weighted summation of the lengths of its parts in different region.

path problem in 3D, i.e., to plan the shortest path between a start point and a goal point in a 3D space while avoiding polyhedral obstacles in the space. The algorithm approximates (within $(1+\epsilon)$ ratio) every locally optimal path by discretizing each edge of the polyhedron obstacle according to the shortest distance from any point on that edge to the start point. This sampling step is applicable to WEC-GWRP: first, apply the same algorithm to discretize every visibility segment; and then solve the Traveling VPP instance using these computed viewpoints. Since the distance from a visibility edge where a planned viewpoint in the optimal WEC-GWRP solution resides to the start position (after being weighted by w_p) is at most the optimal cost of WEC-GWRP, the error introduced by the discretization is still bounded w.r.t. the optimal cost of WEC-GWRP. The number of computed viewpoints is $O(\frac{n^8 \log L}{\epsilon})$, where L is the largest coordinate of any point in $\mathcal{P}$.

Aleksandrov *et al.* [AMS05] designed an approximation algorithm for the problem of planning the shortest path on weighted (triangulated) polyhedral surfaces. The algorithm discretizes each triangular face of the input polyhedral surface $\mathcal{P}$ such that every locally shortest path can be approximated within a ratio of $(1+\epsilon)$ using these discretized points. The number of such computed points is bounded by $O(C(\mathcal{P})\frac{n'}{\sqrt{\epsilon}}\log_2\frac{2}{\epsilon})$, where n' is the number of triangulated faces of $\mathcal{P}$. $C(\mathcal{P})$ captures the geometric characteristic of $\mathcal{P}$ and is proportional to the average reciprocal of the sine values of all the triangular face angles. This sampling step is also applicable to WEC-GWRP: first, triangulate all the visibility cells; second, discretize each triangle using their algorithm; last, solve the corresponding Traveling VPP instance. Since the number of triangles is $O(n^4)$, the number of computed viewpoints is $O(C(\mathcal{P})\frac{n^4}{\sqrt{\epsilon}}\log_2\frac{2}{\epsilon})$.

Note that for both of the above-mentioned approximation algorithms, the number of computed viewpoints depend critically on some geometric parameters, the largest coordinate of any point in the polygon [Pap85], and the quality of the polygon triangulation [AMS05] (Triangulation with sharp angles result in large $C(\mathcal{P})$ values.) respectively, as opposed to our sampling algorithm. To better illustrate, in Section 4.5, we give a simple example in which the triangulation-based algorithm [AMS05] requires substantially more sampling viewpoints than our sampling algorithm to achieve the same approximation ratio.

1.4 Overview of Results

The emphasis of this thesis is to design good algorithms for Traveling VPP and the GWRP. Approximation algorithms are fast and guarantee the algorithmic performances even in the worst case scenarios. In the following, we discuss the methodology adopted and the results for the Traveling VPP and the GWRP respectively.

1.4.1 Traveling VPP and Metric TVPP

The fact that the decoupled approach by Danner and Kavraki, [DK02], cannot solve the Traveling VPP satisfactorily motivates us to model the problem in a unified formulation that combines the view and travel cost in a single objective function to minimize. In Chapter 2, we give an ILP formulation for the Traveling VPP. We show via a reduction to the group Steiner Tree problem (GST) [HK03] that Traveling VPP is log-square inapproximable. We design a rounding algorithm, called *Round and Connect*, that takes an optimal solution to the relaxed linear program (LP) and outputs an integral solution. We also show that the approximation ratio of our rounding algorithm is a constant times F, the frequency of the Traveling VPP, i.e., the maximum number viewpoints that cover a single surface patch. Together with the poly-logarithmic approximation algorithm given in [CEK06, GKR00] for the GST, our algorithm can approximate the Traveling VPP within the ratio of either constant times frequency or the poly-log of the input size, whichever is smaller. This result parallels the well-known approximation ratio result for SCP, i.e., the best approximation ratio for SCP is either the frequency or the logarithm of the number of elements, whichever is smaller.

In Appendix B and Chapter 2, we also discuss several ways of solving the relaxed LP, including the column generation method and give an alternative LP formulation using multi-commodity network flows, the size of which is a polynomial of the input size.

In Chapter 3, we consider a restricted version of the Traveling VPP, the Metric TVPP. We propose a greedy two-step algorithm that first solves the SCP component and then solves the Metric TSP to get a tour connecting the planned viewpoints at the first step. We show the approximation ratio for this algorithm is in the order of the logarithm of the number of surface patches.

1.4.2 WEC-GWRP

Even if seemingly very close to the Traveling VPP, to actually apply the approximation algorithm designed for the Traveling VPP, we will have to deal with the different viewpoint spaces respectively for the two problems, i.e., unlike the Traveling VPP where the discrete viewpoint set is given in advance, for the WEC-GWRP, we have to deal with the continuous polygon interior and an infinite number of possible viewpoints. Although one can discretize, up to a certain resolution, or randomly sample the polygon interior and then call the Traveling VPP solver, as mentioned above, there are two clear shortcomings with this approach: first, the resulting algorithm running time depends critically on the size of the polygon and is no longer truly polynomial (the running time also depends on the number of samples, not an input to the problem); and as we discussed earlier, some critical viewpoints may be missed during the discretization and this results in unbounded approximation ratio.

This motivates the approach we take in this thesis: to propose a novel method that only samples a polynomial number of viewpoints "adequately" (not missing any critical points up to a constant approximation ratio); and then to call the Traveling VPP solver for an approximate solution for these samples. The resulting algorithm then has the approximation ratio as the product of the two parts, i.e., the constant as the result of the sampling algorithm and the minimum of the view frequency parameter and the poly-logarithm of the number of polygon edges.

1.4.3 Experiments

As the first step towards implementations on real robot-sensor systems, we developed a preliminary implementation of the algorithm for simulated 2D polygonal environments as well as real environment maps generated by robot-sensor systems. For the latter, we first compute the polygon approximations. We implement the viewpoint sampling algorithm mentioned above; after calling an LP solver and getting the LP optimal solution, we implement the algorithm Round and Connect for Traveling VPP.

1.5 Contributions

- To the best of our knowledge, Traveling VPP is the first work in robotics that tries to optimize both view and travel cost in a unified and global fashion (as opposed to the decoupled two-level approach [DK02] and the local optimization of looking around a corner [FKN04, IKD03]), and the algorithms developed give the best known approximation ratio. in the order of the view frequency or a polynomial of $\log n$ whichever is smaller.

- WEC-GWRP is also the first and the only work in the computational geometry area that combines AGP and the shortest path problem. We also give the first approximation algorithm for this problem. It has the approximation ratio of the smaller of the order of the view frequency or $O(\log n)$. We believe that the major component of the GWRP solver, the sampling algorithm, is a general technique and can also be used for other shortest route problems. As opposed to the sampling techniques in the shortest path literature, [Pap85, AMS05], the number of computed viewpoints by our sampling algorithm is bounded by a polynomial of the number of polygon vertices and does not depend on any geometric parameter of the polygon.

1.6 Thesis Organization

In the rest of the thesis, we cover the problem formulation and algorithms for Traveling VPP, Metric TVPP and GWRP in Chapters 2. 3. and 4 respectively. The details of the simulator and the simulation results are given in Chapter 5. We also outline some future work directions in Chapter 6, including formulating and proposing possible approaches for unknown version of Traveling VPP.

Chapter 2

Traveling VPP

In this chapter, we model Traveling VPP as a combinatorial optimization problem, give its integer linear program (ILP) formulation, analyze its inapproximability, and give a linear program (LP) relaxation based approximation algorithm.

2.1 Notations and ILP formulation

2.1.1 Traveling VPP: abstraction to set covering on a graph

Traveling VPP is formally defined as: given a set of viewpoints, a set of surface patches, a graph connecting the viewpoints with edge costs, the unit view cost and the unit travel cost, plan a subset of the viewpoints and a subset of the edges, with the minimum total cost as the summation of the view cost (unit view cost times the number of planned viewpoints) and the travel cost (unit travel cost times the total edge cost), such that every surface patch is covered by at least one planned viewpoint.

In this following, we formulate Traveling VPP as a combinatorial optimization problem combining two well-known problems, namely the set covering problem (SCP) and the traveling salesman problem (TSP).

By considering each object surface patch as an element in a subset that a viewpoint can cover [SRR01], an arbitrary VPP instance is immediately an SCP instance. Since the VPP has an inherent geometric structure, one might hope that VPP is a simpler version of SCP. In Chapter 3, we give a reduction from SCP to a special case of the Metric TVPP, a restricted case of Traveling VPP, in which the view cost is dominant, thus equivalent to the

VPP [1]. This implies the equivalence of the SCP and VPP.

On the other hand, assuming the planned viewpoints are given (or the VPP as a sub-problem is already solved), Traveling VPP is reduced to the Metric Traveling Salesman Problem (Metric TSP) by constructing the shortest path graph on the planned viewpoints [DK02]. The shortest path graph is defined as the complete graph between the planned viewpoints in which the edge cost between two viewpoints is the length of the shortest path the robot needs to travel to realize them. Since the two problems, SCP (without metrics) and Metric TSP (with metrics), have fundamentally different structures and solving techniques, Traveling VPP is an interesting and non-trivial generalization.

Thus, in the abstract sense, Traveling VPP can also be termed as "Set Covering on a Graph".

2.1.2 Integer program formulation of Traveling VPP

We now give an integer linear program (ILP) formulation for the Traveling VPP.

In our unified formulation of traveling VPP, we make assumptions to abstract out and focus on certain key ingredients, in particular the interplay between SCP and Metric TSP. We assume that the surface patches to be inspected are given, the viewpoints from which a surface can be inspected are also given, and that a graph which encodes the possible robot movements connecting the viewpoints and the robot start position, is also given. We assume binary covering relationship, i.e., a viewpoint either covers a surface patch or does not cover it. These assumptions are based on a realistic scenario and algorithms from the literature can be used to derive these quantities. We discuss these realistic issues with an eye towards implementation in Section 2.8.

Our formulation of Traveling VPP is to choose a subset of the viewpoints and a (Steiner) tree on the graph to connect them, under the (covering) constraint that every surface patch is covered by at least one planned viewpoint and the (connection) constraint that every planned viewpoint is connected to the robot start position via the planned (Steiner) tree. The objective is to minimize the total cost of the plan, defined as the sum of the view costs and the tree cost (the sum of all edge costs in the Steiner tree).

The reason for using a (rooted) Steiner tree instead of a tour is that we are able to

[1] In [SRR01], the authors claimed the VPP is isomorphic to the SCP but did not give a concrete reduction from an arbitrary SCP instance to a VPP instance.

combine the (rooted) Steiner tree formulation with covering constraints. It is not clear how to combine a tour (or path) constraints (especially the subtour elimination constraints [GP02]) with the covering constraints. Moreover, Metric TSP can be easily approximated using the solution to the Steiner tree problem [Vaz01].

We denote the set of all viewpoints by $\mathcal{V}$ and index them by i. We denote the set of surface patches by $\mathcal{S}$ and index them by j. We use the notation $i \in \mathcal{V}$ and $j \in \mathcal{S}$ to imply the "ith viewpoint" and the "jth surface patch", respectively. For $i \in \mathcal{V}$, let $\mathcal{S}(i)$ denote the subset of the surface patches that viewpoint i covers; and for $j \in \mathcal{S}$, let $\mathcal{V}(j)$ denote the subset of viewpoints that cover surface patch j. The robot movements are restricted to the graph $G = (\mathcal{V}, E)$, where the node set $\mathcal{V}$ is the set of all viewpoints and s, the starting position of the robot. In case the robot start position does not correspond to a viewpoint, we simply assign the empty set as the set of surface patches it sees. The edge e between two views v_{i_1} and v_{i_2} represents the path from v_{i_1} to v_{i_2}. We use c_e to denote the cost (length) of edge e. We also use $T \subset \mathcal{V} : s \notin T$ to denote a cut or subset of the graph that does not include the robot start position. We use $\delta(T)$ to denote the set of edges that "crosses" T, having one end inside T and the other outside T, i.e., $e =< v_1, v_2 > \in \delta(T) \Longleftrightarrow v_1 \in T \wedge v_2 \notin T$ OR $v_2 \in T \wedge v_1 \notin T$. We use w_v, the unit view cost or cost per viewpoint, and w_p, the unit path cost or cost per unit traveling distance. Furthermore, we use F to denote the view frequency, defined as the maximum number of viewpoints that cover a single surface patch, i.e., $F = \max_{j \in \mathcal{S}} |\mathcal{V}(j)|$, where $|\mathcal{A}|$ denotes the cardinality of a discrete set $\mathcal{A}$.

We define a binary variable, y_i, as the indicator whether to take a view at viewpoint i, corresponding to $y_i = 1$, or not, corresponding to $y_i = 0$; we define the binary variable, z_e, as the indicator whether to include the edge e in the robot traveling path, corresponding to $z_e = 1$, or not, corresponding to $z_e = 0$. Thus, the ILP formulation for the traveling VPP is given as:

$$\begin{aligned}
&\underline{\textbf{Traveling VPP (ILP):}} && && (2.1)\\
&\quad\min && w_v \sum_{i\in\mathcal{V}} y_i + w_p \sum_{e\in E} c_e z_e &&\\
&\text{Subject to:} \quad \forall j \in \mathcal{S}\ , && \sum_{i\in\mathcal{V}(j)} y_i \geq 1 && (2.2)\\
&\forall i \in \mathcal{V}, \forall T \subset \mathcal{V} : i \in T \wedge s \notin T, && \sum_{e\in\delta(T)} z_e \geq y_i && (2.3)\\
& && y_i, z_e \in \{0,1\},\ i \in \mathcal{V}, e \in E &&
\end{aligned}$$

The coverage constraints, (2.2), require that for each surface at least one view is chosen from its viewpoint set. The connection constraints, (2.3), require that for each planned view i, i.e., $y_i = 1$, and for every cut T of the vertex set that separates i from the robot start position s, at least one edge that crosses T must be chosen to connect the cut. Such connection constraints are used in the standard (rooted) Steiner tree problem ILP formulation [GW92], and essentially express the notion that each selected node must be reachable from the start node. Note that the above ILP formulation (2.1) is not the most compact one, since there are a large number of constraints corresponding to the cuts in the graph. In the following, we will also give a polynomial-sized formulation (especially useful to solve the corresponding relaxed LP). Nonetheless, this formulation gives us a lot of intuition, since it works directly with the edge assignments, and is handy when we analyze the algorithmic performance.

2.2 Hardness Analysis of Traveling VPP

As the generalization to both SCP and Metric TSP, Traveling VPP is immediately seen to be an NP-hard problem. The hardness of approximation, i.e., the best approximation ratio by any polynomial algorithm, is of great importance to approximation algorithm design. Please see the appendix for a brief recap of the hardness of approximation theory. In this section, we use the result in [HK03] to show the hardness of approximation for Traveling VPP via reductions to the group Steiner tree problem (GST).

GST is defined as follows. Given a graph $G = (V, E)$, where the vertex set V is divided into k distinct groups, $g_1, g_2, \ldots, g_k$, construct the minimum cost Steiner tree to connect at least one vertex from each group. The GST generalizes both the SCP and the Steiner tree problem where the best known approximation ratios are $O(\log n)$ and $O(1)$, respectively. In [GKR00, CEK06], the authors used an LP-based randomized rounding algorithm and a greedy algorithm respectively to achieve the best known poly-log approximation ratio for the GST, $O(\log|V|\log\log|V|\cdot\log k\log N)$, where $|V|$, k and N are the number of graph nodes, the number of groups and the maximum cardinality of the groups respectively.

We now show that the Traveling VPP is not approximable within the same poly-log ratio via the reduction given in [GKR00]. By considering each group in any GST instance as the viewpoint set of a surface patch in the Traveling VPP, GST is reduced to a special case of the Traveling VPP where the viewpoint sets that cover different surface patches are exclusive, i.e., they do not share common viewpoints. Thus, the Traveling VPP is certainly at least as hard as GST and cannot be approximated within $\log^{2-\epsilon}|\mathcal{S}|$, following the result of [HK03], where Halperin and Krauthgamer show that the optimal solution to GST cannot be approximated by any polynomial algorithm within the $O(\log^{2-\epsilon}k)$ ratio, for any $\epsilon > 0$. The question remains whether Traveling VPP is harder to approximate. We show in the following that the inapproximalities of Traveling VPP and GST are of the same order by showing a reduction from Traveling VPP to GST in two steps.

We first show how to reduce an arbitrary Traveling VPP instance to a Traveling VPP instance with 0 view cost. Given an arbitrary Traveling VPP instance with unit view cost and unit traveling cost w_v and w_p respectively, we add a new viewpoint i', for each original viewpoint i, with an identical surface patch set, let the surface patch set for i be empty, and connect i' to i via an edge with cost of $\frac{w_v}{w_p}$. It is easy to see an optimal solution to the reduced (0 view cost) version of Traveling VPP corresponds to an optimal solution to the original Traveling VPP instance since view costs are encoded in the edge costs of the reduced Traveling VPP instance. (Since the surface patch set of the original viewpoint is empty, the new solution has to go to the new copy of the viewpoint, thus incurring the traveling cost $\frac{w_v}{w_p}$ which is equivalent to adding $\frac{w_v}{w_p}\cdot w_p = w_v$ in the objective function.) The size of the resulting instance has $2|\mathcal{V}|$ number of viewpoints and $|\mathcal{V}|+|\mathcal{S}|$ number of edges.

We now show how to construct a GST instance from a Traveling VPP instance with 0 view cost. The idea is to duplicate each viewpoint many times (equal to the number of surface patches it covers) to make the resulting viewpoint sets distinct. Consider such a Traveling VPP instance, i.e., the viewpoint set $\mathcal{V}$, the surface patch set $\mathcal{S}$, the viewpoint set $\mathcal{V}(j)\subseteq\mathcal{V}$ for surface patch $j\in\mathcal{S}$, and the graph G that connects $\mathcal{V}$. We first construct a

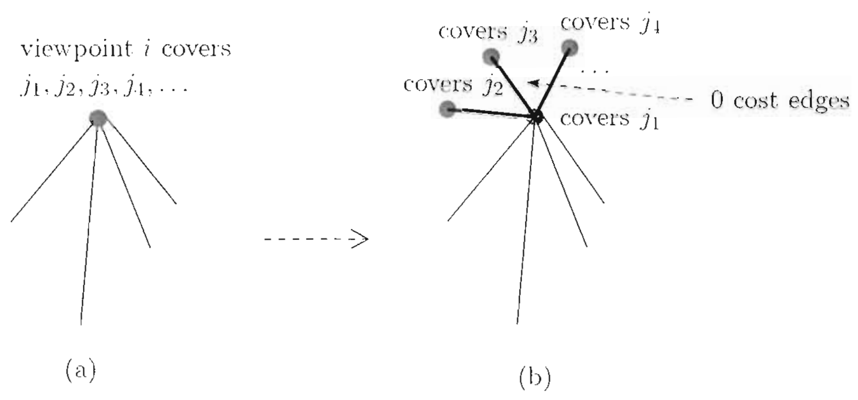


Figure 2.1: Construction of an GST instance from Traveling VPP with 0 view cost.

group g_j for each surface patch j and construct a vertex for each pair of a surface patch j and one viewpoint from its viewpoint set, i.e., (j, i), $i \in \mathcal{V}(j), j \in \mathcal{S}$. We modify the graph G of Traveling VPP accordingly by first constructing a tree with 0-cost edges between the vertices corresponding to the same viewpoint, i.e., $\{(j, i) : j \in \mathcal{S}(i)\}$ (picking an arbitrary vertex (j, i) as the tree root) and then placing the tree root at the node i on G. See Fig. 2.1. Thus, we have a GST instance on the graph over vertices in the form (j, i) and groups corresponding to the surface patches j. And it is easy to see that an optimal GST solution that picks vertices (j, i) and the Steiner tree between them correspond to an optimal solution to Traveling VPP of picking viewpoints i and the resulting Steiner tree connection by collapsing the 0-cost edges. The above GST instance construction produces $O(|\mathcal{V}||\mathcal{S}|)$ number of vertices and $O(E + |\mathcal{V}||\mathcal{S}|)$ number of edges.

By combining the two reduction steps above, an arbitrary Traveling VPP instance with viewpoint set $\mathcal{V}$ and surface patch set $\mathcal{S}$ is reduced to a GST instance with a graph having $O(|\mathcal{V}||\mathcal{S}|)$ vertices, $O(|\mathcal{V}||\mathcal{S}|)$ edges, and $|\mathcal{S}|$ groups. As a result, the Traveling VPP is inapproximable within $O(\log^{2-\epsilon} |\mathcal{S}|)$ ratio of the optimal using any polynomial algorithm. Also the best known approximation algorithms mentioned at the beginning of this section can be applied to the Traveling VPP (after the reductions given above) and the approximation ratio is $O(\log |\mathcal{V}| \log\log |\mathcal{V}| \cdot \log |\mathcal{S}| \log F)$, where F is the view frequency.

2.3 LP based Algorithms for Traveling VPP (ILP)

In this section, we first give the LP relaxation for the ILP formulation given in Section 2.1.2, introduce a rounding algorithm to get an integral solution from the LP solution, give the approximation ratio analysis, and then discuss how to solve the LP.

2.3.1 LP Relaxation for Traveling VPP

By relaxing the binary integral variables, y_i and z_e, to be positive reals, we have the relaxed linear program (LP) formulation given as:

$$
\begin{array}{rlr}
\underline{\textbf{LP Relaxation:}} \quad \min & w_v \sum_{i \in \mathcal{V}} y_i + w_p \sum_{e \in E} c_e z_e & \\
\text{Subject to:} \quad \forall j \in \mathcal{S}\ : & \sum_{i \in \mathcal{V}(j)} y_i \geq 1 & (2.4) \\
\forall i \in \mathcal{V}, \forall T \subset \mathcal{V} : i \in T \wedge s \notin T : & \sum_{e \in \delta(T)} z_e \geq y_i & (2.5) \\
& y_i, z_e \geq 0, \ i \in \mathcal{V}, e \in E &
\end{array}
$$

We call the optimal (fractional) solution and the corresponding cost the *LP-optimal solution* and *LP-optimal value* respectively. The LP-optimal solution corresponds to the fractional *LP-optimal viewpoint assignments* and the fractional *LP-optimal edge assignments*. We call the optimal (integral) solution and corresponding cost to the original ILP the *ILP-optimal solution* and *ILp-optimal value*, respectively. The ILP-optimal solution correspond to the integral *ILP-optimal viewpoint assignments* and the integral *ILP-optimal edge assignments*.

2.3.2 Rounding Algorithm

Let y_i^* and z_e^* denote the LP-optimal viewpoint assignments and the LP-optimal edge assignments respectively, and let OPT^* denote the LP-optimal value, i.e., $OPT^* = w_v \sum_{j \in \mathcal{V}} y_i^* + w_p \sum_{e \in E} c_e z_e^*$. Let y_i', z_e' denote the integral solution given by the algorithm *Round and Connect* given below, and let $cost_{Alg}$ denote the corresponding cost, i.e., $cost_{Alg} = w_v \sum_{j \in \mathcal{V}} y_i' +$

$w_p \sum_{e \in E} c_e z'_e$. Throughout this thesis, we use the superscript $*$ to denote the LP-optimal solution/cost to the corresponding problem instance; and use superscript $'$ to denote a feasible ILP solution/cost. The algorithm *Round and Connect* is given below:

```
Algorithm Round and Connect: (take LP-optimal $y_i^*, z_e^*$ as input and output $y'_i, z'_e$)

  Step 1. Initialize.
  Set viewpoint choice set $\mathcal{V}^c$ to include all the viewpoints, i.e., $\mathcal{V}^c \leftarrow \mathcal{V}$; the viewpoint
  solution set $\mathcal{V}'$ to be empty, i.e., $\mathcal{V}' \leftarrow \emptyset$; the uncovered surface patch set $\mathcal{S}^u$ to include all
  surface patches, i.e., $\mathcal{S}^u \leftarrow \mathcal{S}$
  Step 2. Round.
  While set $\mathcal{S}^u$ is not empty
    Select the viewpoint $i_{max} \in \mathcal{V}^c$ that covers some uncovered surface patch(es) and
    has the largest LP-optimal viewpoint assignment, i.e., $i_{max} = \arg\max_{i \in \mathcal{V}^c:\ \mathcal{S}(i) \cap \mathcal{S}^u \neq \emptyset} y_i^*$,
    and add it to $\mathcal{V}'$, i.e., $\mathcal{V}' \leftarrow \mathcal{V}' \cup \{i_{max}\}$
    Delete the surface patch(es) that $i_{max}$ covers from the uncovered surface patch set,
    i.e., $\mathcal{S}^u \leftarrow \mathcal{S}^u \setminus \mathcal{S}(i_{max})$; and delete $i_{max}$ from the viewpoint choice set, i.e.,
    $\mathcal{V}^c \leftarrow \mathcal{V}^c \setminus \{i_{max}\}$
  End while
  Output $\mathcal{V}'$, i.e., set $y'_i = 1$ for $i \in \mathcal{V}'$, and set $y'_i = 0$ for $i \notin \mathcal{V}'$.
  Step 3. Connect.
  Get the optimal solution to the Steiner tree problem to connect $\mathcal{V}'$. Set $z'_e = 1$ for edges
  in the solution, and 0 otherwise.
```

In the above algorithm, we iteratively choose the viewpoint with the largest (fractional) LP-optimal viewpoint assignment until all the surface patches are covered. We then feed these chosen viewpoints to a Steiner tree algorithm to get the optimal integral solution, in the Connect step. Note that the Steiner tree problem is an NP-complete problem for a general graph. So practically speaking, we can use a constant-ratio approximation algorithm, for example the one in [GW92], and incur an additional bounded performance degradation. It is easy to see that the rounding part of the above algorithm (up to the Connect step) runs in polynomial time, $O(|\mathcal{V}||\mathcal{S}|)$.

2.4 Approximation ratio for algorithm *Round and Connect*

It is trivial to see that the solution given by algorithm *Round and Connect* is a feasible integral solution. In the following, we analyze the performance of the algorithm using the fact that the LP-optimal value is a *lower bound* on the ILP-optimal value. We first show that the view part of the cost of the solution given by the algorithm is bounded and then bound the total cost using a feasible *hybrid solution* with integral viewpoint assignments and fractional edge assignments.

2.4.1 View cost analysis

In the following, we show in Lemma 2 that the LP-optimal viewpoint assignments of the chosen viewpoints are lower bounded by $\frac{1}{F}$. This follows immediately from Lemma 1, which is a simple observation based on the feasibility of the LP-optimal solution. In Corollary 1, these results are used to bound the view cost part of the algorithmic solution.

Lemma 1. *For any surface patch, there exists a viewpoint that covers it with the corresponding LP-optimal viewpoint assignment greater than $\frac{1}{F}$, i.e., $\forall j \in \mathcal{S}, \exists i \in \mathcal{V}(j) : y_i^* \geq \frac{1}{F}$.*

Proof. We show this by contradiction. Assume that for some surface patch $j \in \mathcal{S}$, all the LP-optimal viewpoint assignments are strictly less than $\frac{1}{F}$, i.e., $y_i^* < \frac{1}{F}, \forall i \in \mathcal{V}(j)$. By recalling that view frequency F is the maximum number of viewpoint that covers any surface patch (i.e., $|\mathcal{V}(j)| \leq F, \forall j \in \mathcal{S}$), we must have,

$$\sum_{i\in\mathcal{V}(j)} y_i^* < \sum_{i\in\mathcal{V}(j)} \frac{1}{F} = |\mathcal{V}(j)| \cdot \frac{1}{F} \leq 1$$

The above implies that for $j \in \mathcal{S}$, the sum of covering viewpoint assignments is strictly less than 1, or in other words, surface j is not covered. This contradicts the feasibility of the LP solution, specifically the constraints (2.4). □

Lemma 2. *The LP-optimal viewpoint assignment for each viewpoint chosen by Algorithm* Round and Connect *is lower bounded by $\frac{1}{F}$, i.e., $y_i^* \geq \frac{1}{F}, \forall i \in \mathcal{V}'$.*

Proof. It is equivalent to show that the above algorithm cannot choose any viewpoint whose LP-optimal viewpoint assignment is less than $\frac{1}{F}$. We show this by contradiction. Assume we choose one viewpoint i with $y_i^* < \frac{1}{F}$. By the *Round and Connect* algorithm, the Round

Step, at the iteration when i is picked, it has the maximum LP-optimal viewpoint assignment among the viewpoints that cover the remaining uncovered surface(s). We arbitrarily choose one uncovered surface patch that i covers. By Lemma 1, there exists another i' for which $y^*_{i'} \geq \frac{1}{F}$. This implies $y^*_{i'} > y^*_i$. i' has not yet been chosen, since otherwise all its covering surface pathes including this *uncovered* one would have been deleted from uncovered surface patch set. This contradicts that i has the largest LP solution, y^*_i, among unchosen viewpoints that cover uncovered surface patch(es). □

Lemma 2 implies that the view cost part of the algorithmic solution is bounded by the view cost of the LP-optimal, as stated in Corollary 1.

Corollary 1. *Algorithm* Round and Connect *gives an integral solution with view cost at most F times the view cost of the LP-optimal solution, i.e., $w_v \sum_{i\in\mathcal{V}} y'_i \leq F \cdot w_v \sum_{i\in\mathcal{V}} y^*_i$.*

Proof. By Lemma 2, we have $Fy^*_i \geq 1$, for all the chosen viewpoint $i \in \mathcal{V}'$. It follows that

$$w_v \sum_{i\in\mathcal{V}} y' = w_v \sum_{i\in\mathcal{V}'} 1 \leq F \cdot w_v \sum_{i\in\mathcal{V}'} y^* \leq F \cdot w_v \sum_{i\in\mathcal{V}} y^*$$

□

2.4.2 Total cost analysis

In the following, after stating in Lemma 3 the half-integrality gap[2] result of the Steiner tree problem [Vaz01], we show that the solution given by the algorithm *Round and Connect* has a total cost at most $2F$ times the LP-optimal value. Since the LP-optimal value is a lower bound on the ILP solution, it follows that the algorithm *Round and Connect* has approximation ratio of $2F$.

Lemma 3. *For the Steiner tree problem, the integrality gap between the IP and its relaxed LP is* 2.

Proof. See Chapter 22 of [Vaz01]. □

Note that the Connect Step of the algorithm *Round and Connect* corresponds to the Steiner tree problem of connecting $\mathcal{V}'$, the ILP-optimal solution to which is z'_e. We use

[2]Integrality gap for an LP is defined to be the worse-case ratio between the cost of the ILP solution and that of its LP relaxation solution [Vaz01].

OPT'_{tree} to denote the corresponding optimal value, i.e., $OPT'_{tree} = \sum_{e \in E} c_e z'_e$. Again, we use OPT^*_{tree} to denote the corresponding relaxed LP-optimal value. Now we are ready to show the approximation ratio of algorithm *Round and Connect*.

Theorem 1. *Algorithm* Round and Connect *has an approximation ratio of* $2F$, *i.e.,* $cost_{Alg} \leq OPT^* \cdot 2F$.

Proof. To prove the approximation ratio result, we utilize an intermediate solution, denoted by y'_i, z^s_e, with integral viewpoint assignments and fractional edge assignments. We emphasize that this solution is only used in the proof and not computed in the algorithm. The viewpoint assignments are the same as in the algorithm output y'_i, and the edge assignments are scaled by F, i.e., $z^s_e = F z^*_e$. The superscript s denotes it is a solution after scaling. We call it the *hybrid solution*, and denote the total cost of this solution $cost^h_{Alg}$. By Corollary 1 and the edge scaling, we have,

$$\begin{aligned} cost^h_{Alg} &= w_r \sum_{i \in \mathcal{V}} y'_i + w_p \sum_{e \in E} c_e z^s_e \\ &\leq F \cdot w_r \sum_{i \in \mathcal{V}} y^*_i + F \cdot w_p \sum_{e \in E} c_e z^*_e \leq F \cdot OPT^*. \end{aligned}$$

Now, we claim that the hybrid solution is a feasible solution to the LP relaxation of Traveling VPP. Since the viewpoint assignments of the hybrid solution is exactly the same as in the solution given by the algorithm *Round and Connect*, all the covering constraints, (2.4), are satisfied by the solution viewpoint set $\mathcal{V}'$. The connection constraints, (2.5), are also satisfied, since

$$\sum_{e \in \delta(T)} z^s_e = \sum_{e \in \delta(T)} F z^*_e \geq F y^*_i \geq y'_i, \quad \forall i \in \mathcal{V}.$$

The first inequality above is due to the feasibility of the LP-optimal solution, and the second is due to Lemma 2.

Since all y'_i are integral, z^s_e is a feasible LP solution to the Steiner tree problem to connect $\mathcal{V}'$. It follows immediately that the connection cost $\sum_{e \in E} c_e z^s_e$ is at least the LP-optimal value to connect $\mathcal{V}'$, i.e.,

$$\sum_{e \in E} c_e z^s_e \geq OPT^*_{tree}.$$

Note that the algorithm *Round and Connect* (the Connect Step) gives an optimal integral Steiner tree solution to connect $\mathcal{V}'$. By the integrality gap result for Steiner trees, Lemma 3, this tree cost is at most twice the LP-optimal value for the Steiner tree problem to connect $\mathcal{V}'$, i.e., $\sum_{e\in E} c_e z'_e = OPT'_{tree} \leq 2 \cdot OPT^*_{tree}$. So we have,

$$\sum_{e\in E} c_e z'_e \leq 2 \cdot OPT^*_{tree} \leq 2 \cdot \sum_{e\in E} c_e z^s_e = 2F \cdot \sum_{e\in E} c_e z^*_e$$

(The last equality above is due to the edge scaling.)

Combined with the view cost part of the algorithmic solution (Corollary 1), we have,

$$\begin{aligned} cost_{Alg} &= w_v \sum_{i\in V} y'_i + w_p \sum_{e\in E} c_e z'_e \\ &\leq F \cdot w_v \sum_{i\in V} y^*_i + 2F \cdot w_p \sum_{e\in E} c_e z^*_e \\ &\leq OPT^* \cdot 2F. \end{aligned}$$

which implies the algorithm *Round and Connect* has approximation ratio of at most $2F$. □

2.4.3 Integrality gap for *Traveling VPP*

Theorem 1 shows that the algorithm *Round and Connect*, recovers an integral solution from any LP-optimal solution to Traveling VPP and the solution cost is within $2F$ times the optimal value. This implies that the integrality gap between ILP-optimal and LP-optimal for Traveling VPP is at most $2F$. In the following, we show that $2F$ is also the integrality gap for *Traveling VPP* by giving an example that achieves this ratio.

Theorem 2. *The integrality gap of the Traveling VPP is* $2F$.

Proof. We show the integrality gap of Traveling VPP is at least $2F$ by giving a Traveling VPP instance where the $2F$ ratio is achieved.

In the Traveling VPP instance in Fig. 2.2, there are n surface patches. There are n clusters of viewpoints, denoted by $\mathcal{C}_1, \ldots, \mathcal{C}_n$ respectively, each of which corresponds to one surface patch. Each cluster has exactly F viewpoints, each of which covers only the corresponding surface patch of that cluster. We label a viewpoint by the cluster it belongs to and its index within that cluster, for example viewpoint $i_{C_1,1}$. There are two types of edges, e^1 and e^2, in the graph, superscript 1 or 2 denote the edge type. e^1 edges have the common edge cost $\epsilon \ll 1$ and e^2 edges have the common cost 1. The e^1 edges in each cluster form a complete graph; and the e^2 edges form a complete graph between the clusters. We also use an e^1 edge to connect the robot start position s to a single viewpoint of a single cluster. We further assume the view cost is negligible compared with traveling cost, i.e., $w_v \ll w_p$.

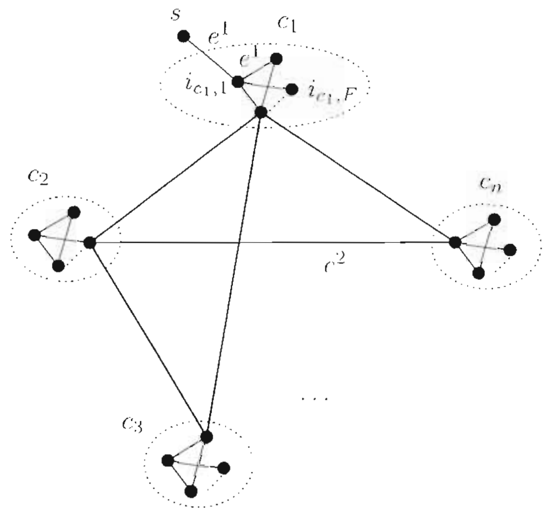


Figure 2.2: A Traveling VPP instance. There are $n \cdot F$ viewpoints, grouped into n clusters (circled by dashed curves). Each cluster contains exactly F viewpoints. For example, cluster C_1 contains viewpoints $i_{C_1,1}, \ldots, i_{C_1,F}$. There are n surface patches (not drawn). All viewpoints in a cluster, $i \in C_j$, only cover one surface patch indexed the same as the cluster, $j \in S$. Two types of edges, labeled by e^1 and e^2, connect the viewpoints. Inside each cluster, e^1 edges form a complete graph. Between clusters, e^2 edges form another complete graph among the representative viewpoints of the clusters, one representative per cluster.

It is not difficult to see that the ILP-optimal solution is to choose a single viewpoint from each cluster and construct a tree (which is also a path connecting these viewpoints) using $(n-1)$ e^2 edges, and the corresponding ILP-optimal value is approximately $(n-1) \cdot 1$ (neglecting view cost and e^1 edge costs). The LP-optimal solution, however, is to assign $\frac{1}{F}$ to each viewpoint and $\frac{1}{n-1} \cdot \frac{1}{F}$ to each e^2 edge. (Since e^1 edges do not contribute much to the objective function, we can simply ignore all the e^1 edges in the solution.) This solution is feasible since for any viewpoint, the cuts that separate it from other clusters have at least $n - 1$ e^2 edges, and the sum of such edge assignment is at least $(n-1) \cdot \frac{1}{n-1} \cdot \frac{1}{F} = \frac{1}{F}$, the viewpoint assignment. The corresponding LP-optimal value is thus approximately $\frac{1}{(n-1)F} \cdot \begin{pmatrix} n \\ 2 \end{pmatrix} \quad \frac{n}{2F}$. (There are all together $\begin{pmatrix} n \\ 2 \end{pmatrix}$ number of e^2 edges.) So the ratio between ILP and LP-optimal values approaches $2F$ assuming n is large, and the integrality gap for the general problem is at least $2F$.

In conclusion, since both the upper and lower bounds are $2F$, the integrality gap of Traveling VPP is $2F$. $\square$

The integrality gap result for Traveling VPP, Theorem 2, implies that the $2F$ approximation algorithm *Round and Connect* is the best possible for the LP relaxation given above.

2.5 Solving the relaxed LP

With the algorithm *Round and Connect*, we can recover an integral solution from a relaxed LP-optimal solution, with the approximation ratio of $2F$ for a general *Traveling VPP*. However, the corresponding relaxed LP formulation may have exponential number of connection constraints. (constraint (2.5)). In the following, we first consider a special but important case, the *Traveling VPP on a Tree*, i.e., the graph G connecting the viewpoints is a tree, and give a polynomially-sized LP relaxation formulation. It is motivated by tree structures commonly used in motion planning techniques to explore and represent the connectivity of the configuration space [BATM95, LK99]. For the general graph case, we suggest two ways: to use an alternative LP relaxation formulation with polynomial size based on multi-commodity flows; or to adopt the column generation approach to practically solve the LP. We cover the former approach in this section and refer to Appendix B for the column generation approach.

2.5.1 Traveling VPP on a Tree

Traveling VPP on a Tree is defined formally as: given a set of viewpoints, a set of surface patches, a tree connecting the viewpoints with edge costs, the unit view cost and the unit travel cost, plan a subset of the viewpoints and a subtree that connects them, with the minimum total cost as the summation of the view cost (unit view cost times the number of planned *viewpoints) and the travel cost (unit travel cost times the total edge cost of the planned subtree), such that every surface patch is covered by at least one planned viewpoint.*

First, we claim that the approximation ratio of algorithm *Round and Connect* improves to F for *Traveling VPP on a Tree*. This is because there is no integrality gap for "Steiner tree on a tree", since both the ILP-optimal and LP-optimal solutions of "Steiner tree on a tree" correspond to taking the union of the unique paths on the tree that connect the planned viewpoints to the start position.

However, we emphasize that *Traveling VPP on a Tree* is not a simple problem. First, note that the counterexample given in the introduction is also a *Traveling VPP on a Tree* instance. Second, we show, via a counterexample, that a greedy algorithm based on amortized costs can perform quite poorly (linear approximation ratio). The algorithm is to iteratively pick a viewpoint with the least amortized cost, i.e., the sum of the view cost and the shortest path cost to connect to the existing tree, divided by the number of uncovered patch(es) it covers, and iteratively grow the existing tree using this shortest path. Although greedy algorithms based on amortized cost have been shown to achieve the logarithmic approximation ratio (the best approximation ratio) for the SCP [Vaz01], it is not so for *Traveling VPP on a Tree*. Consider the example in Fig. 2.3. We have to choose either viewpoint i_1 (which covers all the surface patches, but connected to the start via a long edge) or all the remaining viewpoints (connected via much shorter edges). It is not difficult to see that the optimal solution is to choose $i_2, \ldots, i_n$ and edges $e_2, e_{i_2}, \ldots, e_{i_n}$, and the cost is roughly 1, the edge cost of e_2. The algorithm, however, will choose i_1 since the amortized cost of i_1, $\approx \frac{n-1}{n-1} = 1$, is less than that of any other viewpoint, $\frac{1+\epsilon}{1} = 1 + \epsilon$. The algorithmic solution cost is thus $(n - 1)$, arbitrarily worse than the optimal value (1). Intuitively, this is because the large cost edge is "underestimated" by the large number of surface patches in the amortized cost.

LP formulation for *Traveling VPP on a Tree*

The *Traveling VPP on a Tree* admits a polynomial-sized relaxed LP formulation. Since Linear Program is in P [Kha80], we have a polynomial time approximation algorithm (with view frequency as the approximation ratio) for *Traveling VPP on a Tree* by solving first its LP and using *Round and Connect* to recover an integral solution. In the following, we show the polynomial-sized formulation. Intuitively, for a viewpoint to be connected, only the cuts corresponding to the edges on its unique path (to the start s) are needed in the connection constraints, (constraint (2.5)), thus reducing dramatically the LP size.

Let p_i denote the unique path connecting viewpoint i to s. For an edge $e =< i_1, i_2 >$,

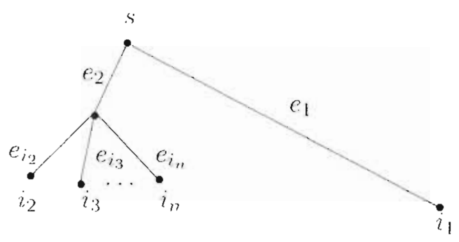


Figure 2.3: An instance of the Traveling VPP on a Tree. There are n viewpoints, labeled by $i_1, i_2, \dots, i_n$, and $n-1$ number of surface patches (not drawn). Viewpoint i_1 covers all the surface patch. Each of viewpoints $i_2, \dots, i_n$ covers only one surface patch, but all together they cover all the patches. Viewpoint i_1 is connected to s via a long edge e_1 with the cost of $n-1$. The remaining viewpoints are first connected to a common node (these connections have negligible costs) and then to s via a short edge e_2 with the cost of $1+\epsilon$, where ϵ is a small positive number. We also assume the traveling cost dominates, and the view cost is negligible.

with i_1 closer to the root of the tree, s, than i_2, we use T_e to denote the subtree of the original tree rooted at i_2, i.e., the subset of tree vertices that are connected to s via e. Note that e is the only edge that crosses the subset T_e, i.e., $\delta(T_e) = \{e\}$. The LP for *Traveling VPP on a Tree* is then given as:

$$
\begin{array}{lll}
\min & w_v \sum_{i\in\mathcal{V}} y_i + w_p \sum_{e\in E} c_e z_e & \\
\text{Subject to:}\quad \forall j \in \mathcal{S}, & \sum_{i\in\mathcal{V}(j)} y_i \geq 1 & (2.6)\\
\forall i \in \mathcal{V}, \forall e \in E : e \in p_i, & z_e \geq y_i & (2.7)\\
 & y_i, z_e \geq 0, i \in \mathcal{V}, e \in E &
\end{array}
$$

Since the covering constraints for the above formulation and for *Traveling VPP* are the same, we only need to show the equivalence of the connection constraints, (2.5) and (2.7). We show this equivalence by reductions in both directions. First, for $i \in \mathcal{V}$ and $e \in p_i$, since T_e is a cut that separates viewpoint i from the start position s and e is the only edge crossing T_e, according to (2.5), we have $\sum_{e'\in\delta(T_e)} z_{e'} = z_e \geq y_i$, (2.7). Second, for any cut T that separates i and s, there must be at least an edge $e \in p_i$ that crosses T, i.e., $e \in \delta(T)$.

(Otherwise i and s will not be separated by T.) So $z_e \geq y_i \Longrightarrow \sum_{e' \in \delta(T)} z_{e'} \geq z_e \geq y_i$, hence (2.5) and (2.7) are equivalent for the tree case.

Note that in the above formulation, the number of constraints is $O(|\mathcal{S}| + |E||\mathcal{V}|)$.

2.5.2 Alternative polynomial-sized LP relaxation formulation for Traveling VPP

Note that for our Traveling VPP formulation (2.1), it is the large number of connection constraints using cuts that prevents us from working with its relaxed LP and solving for the optimal solution. In the following, we show how to use flow formulation (rather than the cut) for this type of constraints. Thus the resulting LP formulation for Traveling VPP will have a polynomial size.

We first double the edges in our undirected graph G in the Traveling VPP formulation and build a digraph to direct the flows in the graph. With slight abuse of notation, we denote the resulting digraph by $G = (\mathcal{V}, E)$. We denote the start and end viewpoints of an arc e by $start(e)$ and $end(e)$ respectively. For each vertex, or viewpoint, i, let $\mathcal{I}(i)$ denote the set of arcs having i as their endpoint and let $\mathcal{O}(i)$ denote the set of arcs having i as their start-point. We then define the commodity flow for each viewpoint and arc pair, $f_{i,e}, i \in \mathcal{V}, e \in E$ to reinforce the connection between s and i if i is chosen, i.e., we require $\sum_{e \in \mathcal{O}(i)} f_{i,e} \geq y_i$. The idea is to define each viewpoint i as the source of the commodity i and s as the terminal for all the commodities. We require the y_i amount of commodity i to flow from source i to s. Then each arc assignment has to guarantee the flow capacity, i.e., $z_e \geq f_{i,e}, \forall i \in \mathcal{V}$. In addition, we require flow conservation for each flow and for each vertex on the graph that is not the terminal of that flow, i.e., $\sum_{e \in \mathcal{I}(i')} f_{i,e} = \sum_{e \in \mathcal{O}(i')} f_{i,e}, \forall i \neq i' \in \mathcal{V}$. So the overall relaxed LP is given as:

Traveling VPP (ILP using flows): (2.8)

$$\begin{array}{rll}
\min & w_v \sum_{i\in\mathcal{V}} y_i + w_p \sum_{e\in E} c_e z_e & \\
\text{Subject to: } \forall j \in \mathcal{S}\ . & \sum_{i\in\mathcal{V}(j)} y_i \geq 1 & (2.9)\\
\forall i \in \mathcal{V} & \sum_{e\in\mathcal{O}(i)} f_{i,e} \geq y_i & (2.10)\\
\forall i \in \mathcal{V} & \sum_{e\in\mathcal{I}(s)} f_{i,e} \geq y_i & (2.11)\\
\forall i, \forall i' \neq i, i' \neq s, i \neq s, & \sum_{e\in\mathcal{I}(i')} f_{i,e} = \sum_{e\in\mathcal{O}(i')} f_{i,e} & (2.12)\\
\forall e \in E, \forall i \in \mathcal{V}, & z_e \geq f_{i,e} & (2.13)\\
 & y_i,\ f_{i,e},\ z_e \geq 0 &
\end{array}$$

In the following, we show the equivalence of the above flow-based formulation (2.8) to the one we give before (2.1) by showing the feasible solution to one formulation corresponds to a feasible solution to the other.

Lemma 4. *The flow based relaxed LP formulation for Traveling VPP (2.8) is equivalent to the graph cut based formulation (2.1).*

Proof. We first show that any feasible solution $y_i, z_e, f_{i,e}$ to (2.8), corresponds to a feasible solution y_i, z_e to (2.1). Since the covering constraints are the same for both formulations, we only need to show the connection constraints. For any cut that separates viewpoint i from s, via the flow conservation law, we know the total amount of commodity flow i crossing the cut (from i to s) is at least the amount emanating from i, which in turn is lower bounded by the viewpoint assignment, (2.10). Since the arc assignment is lower bounded by the flow going through it, (2.13), we have the total number of edges crossing T is at least y_i.

Second, for any optimal LP solution y_i, z_e to (2.1), we can pick for each $y_i > 0$ the unique path from i to s in the solution[3], and assign the flow for commodity i and arcs (directing towards s) on the path to be y_i. It is clear that this construction gives a feasible flow solution to (2.8). □

[3]Note that the optimal LP solution must be a tree, i.e., the positively assigned edges form a tree connection rooted at s, since otherwise we simply delete, and thus reduce the overall cost, some edges while maintaining connectivity. It follows there exists a unique path from any tree node to the root.

Note that the size of the formulation (2.8) has $|\mathcal{V}| + |E|$ number of variables and $|\mathcal{S}| + O(|\mathcal{V}||E|)$ constraints.

2.6 Poly-log approximation algorithm via reduction to GST

By the reduction mentioned in Section 2.2, we can construct from an arbitrary Traveling VPP instance a GST instance with $O(|\mathcal{V}||\mathcal{S}|)$ vertices, $O(|\mathcal{V}||\mathcal{S}|)$ edges, and $|\mathcal{S}|$ groups. We then apply the randomized rounding algorithm in [GKR00] to achieve a $O(\log|\mathcal{V}|\log\log|\mathcal{V}|\log|\mathcal{S}|\log F)$, a poly-log, approximation ratio.

In conclusion, using both the LP based algorithms, deterministic (Round and Connect algorithm) and randomized (the rounding algorithm in [GKR00]) rounding algorithms respectively, we have the approximation ratio of either $O(F)$ or $O(\log|\mathcal{V}|\log\log|\mathcal{V}|\log|\mathcal{S}|\log F)$, whichever is smaller. This approximation result parallels that for SCP.

2.7 Applications of Traveling VPP Algorithm to Related Problems

In this section, we apply the algorithm Round and Connect to several related problems and extend the best known approximation ratios for these problem from poly-log to the minimum of poly-log and the order of frequency.

2.7.1 GST

Note that a GST instance is a Traveling VPP instance with 0-view cost. So the frequency bound is just the largest cardinality of the groups and applying algorithm Round and Connect gives us a frequency approximation ratio. Thus by applying the algorithm Round and Connect above and LP randomized rounding in [GKR00], the optimal solution to GST is approximated within the ratio of $\min(O(F), O(\log|V|\log\log|V|\log k\log N))$.

2.7.2 Traveling Purchaser Problem (TPP)

The (unlimited capacitated) Traveling Purchaser problem (TPP) [Ram81] is defined as follows. We have a set of warehouses, $\mathcal{W}$, connected by a graph $G = (\mathcal{W}, E)$ with edge costs, and a set of products $\mathcal{P}$. Each warehouse $w \in \mathcal{W}$ has a subset of the products with unlimited supplies, denoted by $\mathcal{P}(w)$, and the corresponding product prices, denoted by $d_{w,p}, w \in \mathcal{W}, p \in \mathcal{P}(w)$. TPP asks for certain warehouses for each product and the tour connection between the planned warehouses with the minimum total cost, the sum of the product purchase cost and the tour cost. Ravi and Salman give a poly-log approximation ratio for TPP [RS99].

The Traveling VPP is a special case of TPP where the prices are uniform for all the product and warehouse pairs. We now show how to reduce an arbitrary TPP instance to an instance of a weighted version of Traveling VPP. By "weighted", we mean the view costs associated with the viewpoints are not uniform. Since this weighted version has exactly the same constraints as Traveling VPP, the algorithm Round and Connect only rounds to 1 the viewpoints with assignments lower bounded by $\frac{1}{F}$. By exactly the same arguments as in Section 2.4, it is clear that the algorithm Round and Connect still has the frequency bound approximation ratio for the weighted version. The reduction from TPP to Traveling VPP is done by "duplicating" a warehouse according to the number of different product prices it offers and connecting them via 0-cost edges.

Formally, we define the price set of a warehouse to the set of different prices it offers for different products, i.e., $D(w) = \{d_{w,p}, w \in \mathcal{W}, p \in \mathcal{P}(w)\}$. We order the set $D(w)$ according to the price entities, and denote the i^{th} smallest price as $d(w)_i$. We then add warehouses $w_i, i = 2, \ldots, |D(w)|$ to the warehouse set $\mathcal{W}$, (We replace the viewpoint w by w_1 with a different product set described as follows.) and add edge $< w_1, w_i >$ with 0 edge cost. The warehouse w_i will cover only the product in its product set that has the price of $d(w)_i$, i.e., $\mathcal{P}(w_i) = \{p \in \mathcal{P}(w) : d(w,p) = d(w)_i\}$. By this construction, each warehouse has a unique price for all the products in its product set. This corresponds to an instance of the weighted Traveling VPP. Note that the frequency bounds for both instances are the same, since we have not increased the number of warehouses that offer a single product.

Thus, by solving the resulting weighted Traveling VPP instance, we can get the $O(F)$ approximation ratio for TPP, where F is defined as the maximum number of warehouses that sell a single product.

2.8 Issues Towards Implementation on a real robot-sensor system

In this section, we discuss several issues and constraints towards implementing our algorithm on a real robotic system.

Our Traveling VPP formulation assumes the viewpoint set $\mathcal{V}$ and the traveling graph G connecting these viewpoints are given. For given scenes, these viewpoints can be derived from the aspect graph of the scene [BD90], or by randomly sampling the sensor configuration space, the space of the sensor configurations that uniquely determines the viewpoints [GBL01]. For 2D polygons, in Chapter 4, we give a deterministic sampling algorithm that computes a polynomial sized viewpoint set and guarantees that the optimal solution can be approximated by the sampling viewpoints. We assume a binary coverage relationship mentioned. i.e., a viewpoint can either cover a surface patch or not. In reality, a viewpoint may cover a surface patch only partially. By subdividing surface patches, we can maintain binary coverage relationship.

Realistic sensor field of view constraints such as the line of sight constraint (i.e., a viewpoint sees a surface patch only if the line segment that connects them is not occluded). the range constraint (a viewpoint sees a surface patch only if the distance between them is within a range), and the incidence constraint (i.e., a viewpoint sees a surface patch only if the angle between the line connecting them and the surface normal is in a range) can be incorporated via viewpoint and surface patch visibility computations. The Traveling graph would essentially be a roadmap built in the configuration space of the robot. This is a standard and well-studied technique for robot motion planning [KSLO96, Lat91].

Chapter 3

Metric Traveling VPP with Visibility Range

In this chapter, we consider a special case of Traveling VPP where a point robot (in two-dimensional or three-dimensional world) equipped with a range sensor of visibility range R is asked to inspect a set of surface patches. We call it the *Metric Traveling VPP with Visibility Range* or Metric TVPP.

Metric TVPP is defined formally as: given a set of viewpoints, a set of surface patches, a graph connecting the viewpoints, the edges of which correspond to paths in the same space as the viewpoints, a sensing range that dictates that a viewpoint covers a surface patch only if they are within the sensing range distance, the unit view cost, and the unit travel cost, plan a subset of the viewpoints and a subset of the edges, with the minimum total cost as the summation of the view cost and the travel cost, such that every surface patch is covered by at least one planned viewpoint.

First, we give a reduction from SCP to Metric TVPP, which implies that the latter is a harder problem than the SCP, and any approximation algorithm for it cannot have a better approximation ratio than the best approximation ratio for the SCP.

Since the Metric TVPP is a restricted version of Traveling VPP, it is natural to ask whether we can find an algorithm for Metric TVPP with better approximation ratio. The answer is yes! And this better approximation algorithm follows the two-level approach by Danner and Kavraki [DK02]. We show this greedy algorithm has the approximation ratio in the same order of the approximation ratio for the set covering problem. Intuitively, the key insight is that the sensor range constraint in Metric TVPP implicitly couples the traveling and view components: in order to cover a surface patch, the robot has to travel to at least within the sensor range of it. Thus the two-level approach is no longer *decoupled*.

3.1 Notation and Formulation

First, we add some additional notations for *Metric Traveling VPP with Visibility Range*.

We denote the environment where both robot and surface patches reside by $\mathcal{P}$. We assume $\mathcal{P}$ is either a two-dimensional or a three-dimensional Euclidean space populated by obstacles, and is denoted by $\mathcal{P}^2$ and $\mathcal{P}^3$ respectively. For any two points x_1 and x_2, we denote the Euclidean distance between them by $\|x_1, x_2\|$. For robot traveling, we again use a complete graph $G = (\mathcal{V}, E)$ with metrics where the edge cost $c_e, e =< i_1, i_2 >$ between two viewpoints $i_1, i_2 \in \mathcal{V}$ is the shortest distance the robot travels. Note that the edge cost is lower bounded by the Euclidean distance, i.e., $\forall e =< i_1, i_2 >: c_e \geq \|i_1, i_2\|$. We denote the visibility range by D. By definition, the necessary condition for a viewpoint i to cover a surface patch j is that the distance between them is upper bounded by D. With a slight abuse of notation, we use $\|i, j\|$ to denote this distance. Rigorously, for a surface patch j that occupies a region $\mathcal{R}(j)$ (of co-dimension 1) in $\mathcal{P}$, the distance $\|i, j\|$ is the upper bound on the distances between any point belonging to surface patch j and the viewpoint i, i.e., $\|i, j\| = \sup_{x \in \mathcal{R}(j)} \|x, i\|$ [1]. We use w_v, the unit view cost or cost per viewpoint, and w_p, the unit traveling cost or cost per unit traveling distance, to allow users to specify the relative weights between sensing and traveling.

With the above notations and settings, the *Metric Traveling VPP with Visibility Range* is formulated as to plan a subset of the viewpoints, denoted by $\mathcal{V}'$, and a traveling path connecting them on G, denoted by E', such that the total cost, $w_v|\mathcal{V}'| + w_p \sum_{e \in E'} c_e$, is minimum. ($|\mathcal{A}|$ denotes the cardinality of set $\mathcal{A}$.)

3.2 Inapproximability of *Metric TVPP*

In this section, we give an L-reduction from SCP to the Metric TVPP. (Please see Appendix A for a quick recap of the L-reduction method to prove inapproximability result.)

[1]This distance definition is different from what is usually used in the literature, the infimum, i.e., $\|i, j\| = \inf_{x \in \mathcal{R}(j)} \|x, i\|$. The supremum definition is used here to model the visibility range.

Given an arbitrary SCP instance, we construct a two-dimensional Metric TVPP instance where the unit view cost is 1 and the traveling cost is negligible, i.e., $w_p \ll w_v = 1$. In the constructed Metric TVPP instance, the number of viewpoints and surface patches are respectively the same as the number of subsets and elements in the SCP instance. Any solution to the constructed Metric TVPP instance corresponds to a solution to the SCP instance with the same cost. This L-reduction extends the inapproximability result for SCP to Metric TVPP.

Theorem 3. *Metric TVPP is $O(\log m)$ inapproximable.*

Proof. In the following, we use Turing Machine model for the reduction construction.

Given an arbitrary SCP instance, we denote the universe of elements by $\mathcal{S} = \{s_j, j = 1, \ldots, m\}$, and a collection of its subsets by $\mathcal{V} = \{v_i : v_i \subseteq \mathcal{S}, j = 1, \ldots, n\}$. We construct a two-dimensional Metric TVPP instance as in Fig. 3.1. We first draw three horizontal line segments of length L, and at heights 0, H', and H respectively. We are going to put viewpoints on the bottom line segment, the obstacles on the middle segment, and the surface patches on top segments. The value of H is given while H' is to be given below. The left endpoints of the three line segments all start on a single vertical line. We divide the top line segment equally into three parts, and then divide equally the middle part $\overline{T_l T_r}$ into m pieces. These m pieces, more accurately the bottom surfaces of them, are the surface patches of the constructed Metric TVPP instance. The size of each patch is less than $\frac{L}{3m}$. Each surface patch corresponds to an element of the universe of the SCP instance. We denote the surface patches using the same labels as those for the elements in the SCP instance, $s_j, j = 1, \ldots, m$, to imply this correspondence. We then create viewpoints, corresponding to the subsets in the SCP instance, and put them on the bottom line segment (at height 0) with equal distances between two consecutive ones. Let r denote this distance. So we have $r = \frac{L}{n+1}$. We construct obstacles of negligible thickness on the middle line (at height H'). For convenience of description, we define the sensing cone of a viewpoint v_i to be $\triangle v_i T_l T_r$. For two neighboring viewpoints v_i and v_{i+1}, we define the intersection point of their sensing cone to be the intersection point of line segments $\overline{v_i T_r}$ and $\overline{v_{i+1} T_l}$. For example, in Fig. 3.1(a), the intersection point of sensing cones of v_1 and v_2 is X_1.

For the middle line of obstacles, we cut some openings, such that each viewpoint covers only the surface patches corresponding to those elements in its corresponding subset. It is obvious that we only need to do this for the part of the middle line inside the the sensing cone of the viewpoint. For example, in Fig. 3.1(b), for viewpoint v_i corresponding to subset $\{s_1, s_2, s_m\}$, we cut three openings such that only the three surface patches corresponding to s_1, s_2, and s_m are visible from v_i. As shown in Fig. 3.1(a), as long as the middle line (at height H') is below the intersection points of the sensing cones, for example X_1 and X_2 in the figure, for any viewpoint v, the obstacles in any other viewpoints' sensing cones cannot occlude the sensing surface patches that v covers. By simple planar geometry, the intersection points of the neighboring sensing cones are of the same height $\frac{rH}{\frac{L}{3}+r}$, which simplifies to $\frac{3}{n+4}H$ using $r = \frac{L}{n+1}$. So we require $H' < \frac{3}{n+4}H$.

The resulting Metric TVPP instance is to plan the minimum number of viewpoints and a connecting tour that cover all the surface patches. Since the traveling cost is negligible, the cost to minimize is just the number of viewpoints planned. The solution to the Metric TVPP instance implies a solution to the SCP instance, i.e., to choose the subsets corresponding to those viewpoints chosen. The two solutions have the same cost. Thus, the L-reduction from SCP to Metric TVPP is constructed.

The above reduction implies that Metric TVPP is at least as hard as the SCP, and any approximation algorithm for the Metric TVPP cannot have a better approximation ratio than the best approximation ratio for the SCP. Thus the inapproximability result for SCP [Fei98] implies this lemma. □

3.3 Two-level Algorithm for *Metric TVPP*

In this section, we give the the pseudo code of a two-level algorithm for *Metric TVPP*.

Algorithm 1. *Two-level Algorithm for* Metric TVPP

```
Step 1. Solve the SCP greedily:
    Iteratively choose the viewpoint that covers the most uncovered surface patches
    until all surface patches are covered.
    Output the chosen viewpoint set $\mathcal{V}'$.
Step 2. Solve the Metric TSP to connect $\mathcal{V}'$
    Construct the shortest path graph $G'$ on $\mathcal{V}'$, i.e., the complete graph on $\mathcal{V}'$ where
    the edge cost is the corresponding shortest path length on $G$
    Use Christofides' algorithm [Chr76] to construct the tour on $G'$ to connect $\mathcal{V}'$ [2].
```

[2] *Christofides' algorithm first constructs the minimum spanning tree T on G'; second, it computes the*

This algorithm solves the problem in two steps. In the first step, it solves the VPP or SCP part of Metric TVPP greedily. In the second step, it solves the Metric TSP to connect these picked viewpoints using the Christofides' algorithm [Chr76].

3.4 Algorithm Analysis

To analyze the approximation ratio of the two-level algorithm, Algorithm 1, we present an alternative algorithm, Algorithm 2, whose performance is no better than Algorithm 1. We emphasize that Algorithm 2 is only for analysis purpose and does not need to be implemented. We then give the approximation ratio of this alternative algorithm. This ratio thus also serves as the approximation ratio for Metric TVPP. In the following, we give the algorithm after a few definitions, and then analyze its performance.

3.4.1 Alternative algorithm

For any surface patch j, any two covering viewpoints must lie within $2D$ distance of each other. $\forall i_1, i_2 \in \mathcal{V}(j), e = (i_1, i_2), \quad c_e \leq 2D$. This is because there exists a free path between i_1 and i_2 having distance less than $2D$, as shown in Fig. 3.2. This path follows first the free visibility line from i_1 to surface j, and then the visibility line from j to i_2. For viewpoint i, we define the free region within its $2D$ distance, i.e., the set of points i can reach using the shortest path of length less than $2D$, the *domain* of i. We call two viewpoints *dependent* if one lies in the other's domain, and *independent* otherwise.

In the following, we give the pseudo code of the alternative algorithm.

Algorithm 2. *Alternative Algorithm for* Metric TVPP

```
    Step 1. Solve the SCP greedily to get $\mathcal{V}'$.
        (This step is exactly the same as Step 1 of Algorithm 1.)
    Step 2. Choose independent viewpoints:
        Iteratively choose a viewpoint from $\mathcal{V}'$ that is independent from already chosen
ones until all the surface patches are covered.
        Output the chosen viewpoint set $\mathcal{V}''$.
    Step 3. Solve the Metric TSP to connect $\mathcal{V}''$:
```

minimum cost perfect matching between vertices of T with odd degrees; third, it adds edges corresponding to this matching to T to make it Eulerian; last, it computes a tour on the resulting Eulerian graph [Chr76].

Construct the shortest path graph G'' on $\mathcal{V}''$.

Use Christofides' algorithm [Chr76] to construct the tour on G'' to connect $\mathcal{V}''$.

Step 4. Connect the viewpoints in $\mathcal{V}' \setminus \mathcal{V}''$ to its nearest neighbor in $\mathcal{V}''$ using the shortest path on G.

Note that Algorithm 2 chooses exactly the same viewpoints, $\mathcal{V}'$, as Algorithm 1. However, it constructs the tour differently: it first constructs the tour to connect only the viewpoints in $\mathcal{V}''$, called *centers*; and for $\mathcal{V}' \setminus \mathcal{V}''$, it constructs "detours" to their nearest centers. So clearly, the solution cost by Algorithm 2 is no better than that by Algorithm 1.

3.4.2 Analysis

In this section, after giving some notations, we first show that the optimal solution to Metric TVPP has to pass through as least a viewpoint in every domain of the centers, Lemma 5. This observation helps lower bound the optimal solution cost, Lemma 6, and upper bound the algorithmic solution cost, Lemma 7. Combining both bounds gives us the algorithmic approximation ratio.

Let OPT_{TVPP} denote the optimal solution cost to Metric TVPP. We denote the algorithmic solution cost (Algorithm 2) by $cost_{Alg}$, the view cost and traveling parts of which are denoted by $cost_{Alg}^{view}$ and $cost_{Alg}^{travel}$ respectively. We use OPT_{SCP} to denote the optimal SCP solution cost to cover all the surface patches, i.e.,

$$OPT_{SCP} = \min_{\mathcal{U} \subseteq \mathcal{V}: \bigcup_{i \in \mathcal{U}} \mathcal{S}(i) = \mathcal{S}} |\mathcal{U}|.$$

We call the tour that connects at least one point from every domain of the centers a *domain tour*. We use $OPT_{DOMtour}$ to denote the shortest length of such tours.

Lemma 5. *The tour in the optimal solution to Metric TVPP has to visit the domains of all centers, and is hence a feasible domain tour.*

Proof. For any center $i_1 \in \mathcal{V}''$, we arbitrarily pick one surface patch j from its surface patch set, i.e., $j \in \mathcal{S}(i_1)$. As shown in Fig. 3.2, any viewpoint i_2 of surface patch j's viewpoint set, i.e., $\forall i_2 \in \mathcal{V}(j)$, has to lie in the domain of i_1. Since the optimal solution to Metric TVPP is a feasible solution, at least one viewpoint from $\mathcal{V}(j)$ is chosen and visited by the constructed tour. $\square$

Lemma 5 leads to the following upper bound result for the cost of the optimal solution to Metric TVPP.

Lemma 6. *The cost of the optimal solution to Metric TVPP is at least the sum of the minimum view cost (ignoring traveling) and the optimal tour cost to visit every domain of the centers, i.e.,*

$$OPT_{TVPP} \geq w_v OPT_{SCP} + w_p OPT_{DOMtour} \tag{3.1}$$

Proof. The viewpoint set chosen by the optimal solution to Metric TVPP is a feasible solution to the corresponding SCP. Lemma 5 shows that the tour chosen is a domain tour. Hence their costs are lower bounded by OPT_{SCP} and $OPT_{DOMtour}$, respectively. □

Also we give the upper bound result for the cost of the algorithmic solution.

Lemma 7. *The algorithmic solution cost, $cost_{Alg}$, is at most $w_v OPT_{SCP} \cdot (1+6D\frac{w_p}{w_v})O(\log m) + 1.5 w_p OPT_{DOMtour}$, where m is the number of surface patches.*

Proof. We use the greedy SCP algorithm to get $\mathcal{V}'$. According to its approximation ratio [Vaz01], we have:

$$|\mathcal{V}'| \leq OPT_{SCP} \cdot O(\log m) \tag{3.2}$$

This implies that:

$$cost^{view}_{Alg} = w_v \cdot |\mathcal{V}'| \leq w_v \cdot OPT_{SCP} \cdot O(\log m) \tag{3.3}$$

We use Christofides' algorithm [Chr76] to get a tour of all the centers in $\mathcal{V}''$. This tour is at most 1.5 times the cost of the optimal Metric TSP solution to connect the centers. Since we do not know the value of the latter, we will instead use the cost of another feasible Metric TSP solution to connect the centers. This solution is to first travel on the optimal domain tour and then detour to the centers (and back) if needed, as shown by the dotted path in Fig. 3.3. By the definition of the domain tour, for any center, there must exist a point on the tour within its $2D$ distance. This implies that the tour constructed in Step 3 of Algorithm 2 has length at most $1.5(OPT_{DOMtour} + |\mathcal{V}''| \cdot 4D)$. The $4D$ factor corresponds to traveling from such a point on the domain tour to the center and then traveling back.

Further detouring (if needed) from a center to any of its dependent viewpoints in $\mathcal{V}' \setminus \mathcal{V}''$, Step 4 of Algorithm 2, incurs again a traveling distance of at most $4D$. So the overall traveling cost of the algorithmic solution is upper bounded as follow:

$$\begin{aligned} cost_{Alg}^{travel} &\leq w_p \cdot 1.5(OPT_{DOMtour} + 4D|\mathcal{V}''|) \\ &\quad + w_p \cdot 4D(|\mathcal{V}'| - |\mathcal{V}''|) \\ &\leq 1.5w_p(OPT_{DOMtour} + 4D|\mathcal{V}'|). \end{aligned} \tag{3.4}$$

The factor 1.5 above is due to the approximation ratio of the Christofides' algorithm [Chr76].

In conclusion, using both Eqs. (3.2), (3.3), and (3.4), the total cost of the algorithmic solution is upper bounded as follows:

$$\begin{aligned} cost_{Alg} &= cost_{Alg}^{travel} + cost_{Alg}^{view} \\ &\leq w_v OPT_{SCP} \cdot (1 + 6D\frac{w_p}{w_v})O(\log m) \\ &\quad +1.5w_p OPT_{DOMtour} \end{aligned} \tag{3.5}$$

□

With the upper and lower bounds of the optimal and algorithmic solution costs to Metric TVPP, we are ready to prove the approximation ratio result for Algorithm 2.

Theorem 4. *The approximation ratio of Algorithm 1 is* $\max\{1.5, (1 + 6D\frac{w_p}{w_v})O(\log m)\}$.

Proof. As mentioned before, the Metric TVPP solution by Algorithm 2 has no better cost than that by Algorithm 1. Combining the lower bound for the optimal solution cost OPT_{TVPP}, Lemma 6, and the upper bound for the algorithmic solution cost $cost_{Alg}$, Lemma 7, it is easy to show that the approximation ratio of Algorithm 2 is $\max\{1.5, (1 + 6D\frac{w_p}{w_v})O(\log m)\}$. □

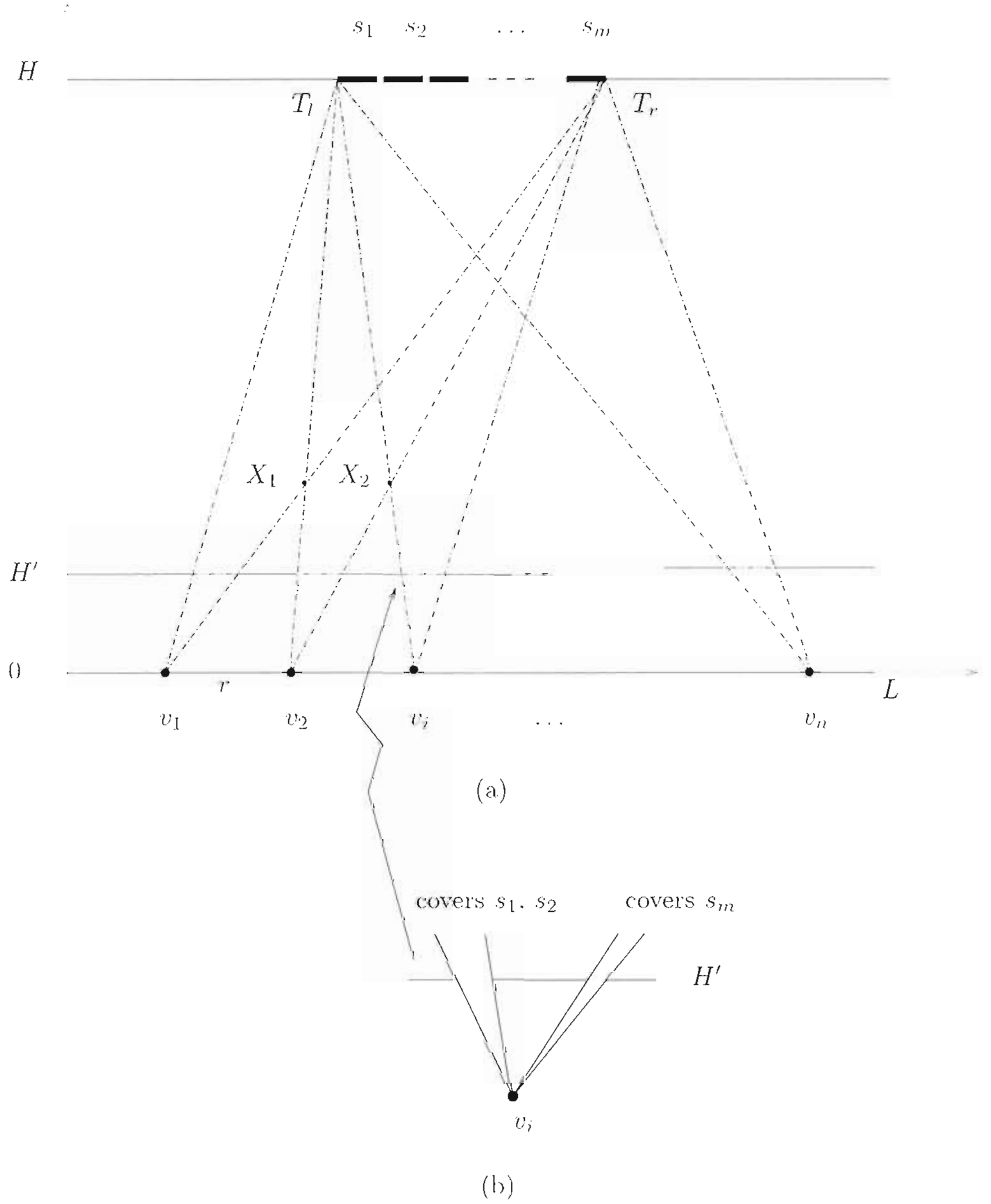


Figure 3.1: Reduction from an arbitrary SCP instance to a VPP instance. See the text for the explanation.

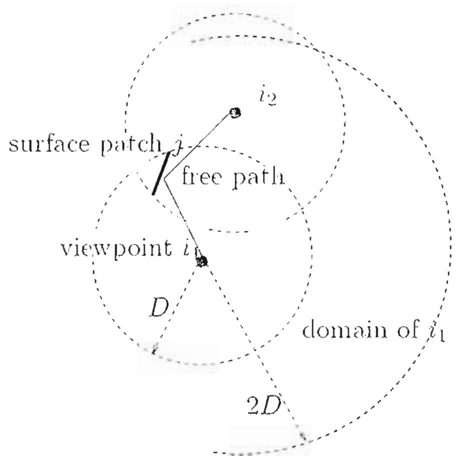


Figure 3.2: Domain and dependence. Viewpoints i_l and i_2 are dependent and have a free path of length $\leq 2D$, comprising of solid line segments joining the two viewpoints.

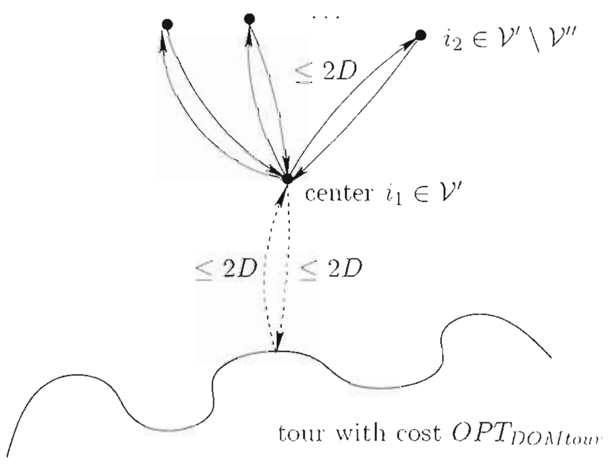

. . .
$i_2 \in \mathcal{V}' \setminus \mathcal{V}''$
$\leq 2D$
center $i_1 \in \mathcal{V}'$
$\leq 2D$
$\leq 2D$
tour with cost $OPT_{DOMtour}$

Figure 3.3: A feasible Metric TVPP solution that takes the optimal domain tour and detours first to the centers, and then to rest of the planned viewpoints.

Chapter 4

Generalized Watchman Route Problem

4.1 Introduction

A key distinction between the Traveling VPP and the WEC-GWRP is that while the former has a discrete viewpoint set given in advance, for the latter, we have to deal with a continuum of viewpoints, the entire polygon. This motivates us to compute a finite number of viewpoints inside the polygon to reduce the WEC-GWRP to Traveling VPP. We propose a novel sampling algorithm that computes a finite number of discrete viewpoints in the polygon ($O(n^{12})$, where n is the number of polygon vertices). We emphasize here that the number of computed viewpoints does not depend on any geometric parameter of the polygon as opposed to [Pap85] and [AMS05]. We show that if we restrict the problem to choose planned viewpoints only from these sample viewpoints, the cost of the optimal solution to the corresponding Traveling VPP instance is at most a constant times the cost of the true optimal WEC-GWRP solution. We then construct a Traveling VPP instance using the sample viewpoints and call the approximation algorithm in Chapter 2 for a solution. This implies that the cost of the resulting solution is at most the cost of the optimal solution to WEC-GWRP times the smaller of the order of the view frequency and a polynomial of $\log n$.

The sampling algorithm works in two steps: first it reduces the viewpoint space from the polygon (2D) to a bounded number of line segments (1D), and then from these line segments (1D) to a bounded number of points. In the first step, we decompose the polygon into *visibility cells*, computed via a partition such that the same polygon edges are entirely visible from all points in each cell. A cell is either an open 2D region, an open line segment, or a point (an end-point of the line segment). We call the closures of the line segments including the cell boundaries the *visibility segments*, and show that the optimal WEC-GWRP solution has a corresponding solution with the same cost, in which the planned viewpoints (other than the start position S) are restricted on the visibility segments.

Note that if travel cost is ignored, it suffices to sample one viewpoint arbitrarily on each visibility segment. However, due to the tradeoff between view and travel cost, we do not know which viewpoint on each visibility segment the optimal WEC-GWRP solution may choose. This motivates us to utilize the metric structure in the problem to guide our sampling from 1D to points. We define a local region of each visibility segment, called *domain*, and compute a bounded number of viewpoints inside the domain such that the optimal WEC-GWRP solution can be approximated (within a constant ratio) locally using these sample viewpoints. Please see Fig. 4.3 for a preview of the domain, a diamond shape consisting of two isosceles triangles of side angle α. Intuitively, when α is small, the part of any path inside it can be approximated well horizontally, since the domain is a narrow horizontal region. Similarly, when α is large, the part can be approximated well vertically. So by choosing an appropriate α, the part can be approximated in any direction. We also show that the optimal WEC-GWRP solution as a whole can be approximated within a constant ratio once all the local approximations are chained together. It turns out that there is an optimal α for which this constant approximation ratio is minimum. For sampling inside each domain, intuitively, we would like to impose an "ordering" on the visibility segments, which lets us exploit the weak "metric" between them. This is achieved by dividing domains into *strips* using the visibility cell vertices such that the visibility segment ordering remains the same within a strip.

4.2 Problem definition

We now formally state the WEC-GWRP. Let $\mathcal{P}$ denote the given polygon (with or without holes), a closed set. Let $\partial\mathcal{P}$ denote its boundary, including the boundary of the holes. Let $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ and $\mathcal{E} = \{e_1, e_2, \ldots, e_n\}$ denote the set of polygon vertices and the set of polygon edges, respectively. Let $\mathcal{A}_r$ denote the set of reflex vertices of $\mathcal{P}$ (internal angle > 180 degrees). We use $\overline{X_1X_2}$ to denote the closed line segment between two points X_1 and X_2. Under the line-of-sight assumption, the *visibility* relation between two points X_1 and X_2, i.e., "X_1 is visible from X_2" or "X_2 is visible from X_1", denoted by $X_1 \diamond X_2$, is defined as:

$$X_1 \diamond X_2 \Leftrightarrow \overline{X_1X_2} \subseteq \mathcal{P}.$$

This definition permits the line segment between two visible points be incident on polygon vertices and/or edges. If X_1 is a viewpoint, we say "X_1 *covers* X_2". We also say viewpoint X covers a polygon edge e, denoted by $X \diamond e$, if all points of e are visible from X, i.e.,

$$X \diamond e \Leftrightarrow \forall X' \in e : X \diamond X'.$$

Let $S \in \mathcal{P}$ denote the start position of the watchman. Let $\mathcal{V}'$ denote a subset of viewpoints, i.e., $\mathcal{V}' = \{X : X \in \mathcal{P}\}$ and $route(\mathcal{V}')$ denote a route connecting the viewpoints in $\mathcal{V}'$ and S. Let w_v and w_p denote the weights for the view and travel costs, respectively. Let $|\mathcal{B}|$ denote the cardinality of a discrete set $\mathcal{B}$, and let $\|\phi\|$ denote the length of route ϕ.

The WEC-GWRP is defined as follows:

$$\begin{aligned} \min \quad & w_v|\mathcal{V}'| + w_p\|route(\mathcal{V}')\| \\ \text{Subject to} \quad & \forall e \in \mathcal{E}, \exists X \in \mathcal{V}' : \quad X \diamond e \end{aligned} \tag{4.1}$$

4.3 Sampling Algorithm

The sampling algorithm consists of two steps. It first constructs the *visibility cell* decomposition and restricts the planned viewpoints to be on the visibility segments. It then samples in the vicinity of each visibility segment. We define a local (w.r.t. a visibility segment) region called *domain* to quantify this vicinity concept. The shape of the domains is so designed that the behavior (both view and travel) of any watchman route is locally approximated within a constant ratio. In the following, we give details of each step.

4.3.1 Visibility cell decomposition

First, we give some definitions and observations to clarify the visibility cell decomposition. Our decomposition is a "finer" version than that given in [ZG05], i.e., each cell defined here is completely contained in a single cell defined in [ZG05]. This implies that the properties of the cells defined in [ZG05] are preserved here. Similar terminologies (not by exactly the same names) and results can also be found in [BLM92, GMR97].

Let $VP(X)$ denote the *visibility polygon* of a point $X \in \mathcal{P}$, i.e., the set of points in $\mathcal{P}$ that is visible from X. $VP(X)$ is a *star-shaped* simple polygon, whose edges are either those contained in $\partial\mathcal{P}$ or are *constructed edges* incident on reflex vertices. We call these constructed edges the *windows* of point X. We further extend each window in the direction from X to the incident reflex vertex until it hits the polygon boundary for the last time, and call it the *extended window*. An extended window is a single line segment that may contain parts outside the polygon $\mathcal{P}$. For example, in Fig. 4.1, the visibility polygon of vertex A_1 consists of a window $\overline{A_5X_1}$, and the corresponding extended window is $\overline{A_5X_3}$. We use $\mathcal{W}(X)$ to denote the set of extended windows of point X. The extended windows of the polygon vertices are of particular interest here. We call them the *critical extended windows*. The set of all critical extended windows is denoted by $C\mathcal{W}(\mathcal{P})$, i.e., $C\mathcal{W}(\mathcal{P}) = \cup_{A_i \in \mathcal{A}} \mathcal{W}(A_i)$. It is easy to see $|C\mathcal{W}(\mathcal{P})| \leq |\mathcal{A}_r||\mathcal{A}| = O(n^2)$.

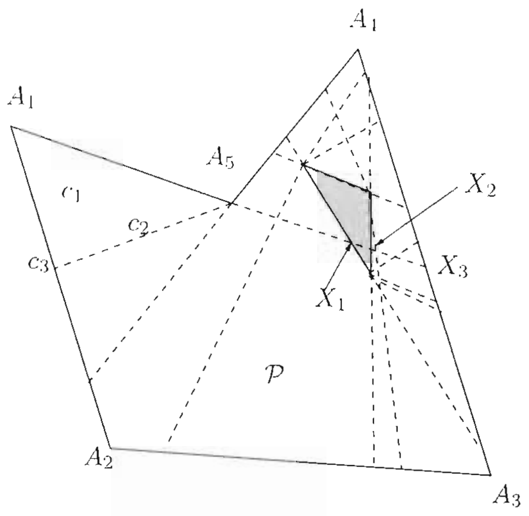


Figure 4.1: Visibility cell decomposition of polygonal $\mathcal{P}$. The shaded region is a hole.

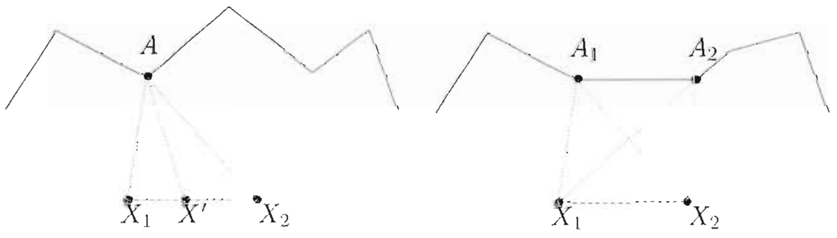
We use critical *extended* windows to partition the polygon into *visibility cells*. Let C denote the set of all visibility cells. See Fig. 4.1. The visibility cells include 2D open regions called *visibility faces*, e.g., c_1 in Fig. 4.1, open line segments (excluding the end-points) called *visibility edges*, e.g., c_2 in Fig. 4.1. and points called *visibility vertices*, e.g., c_3 in Fig. 4.1. We take the closure (to include the end-points) of all the visibility edges and call them the *visibility segments*. The set of all visibility segments is denoted by $\mathcal{L}$. Any visibility face is bounded by its *bounding visibility segments*. This *visibility cell decomposition* is efficiently computed by first computing the extended critical windows, computing the arrangement of these windows and the polygon edges, and then excluding parts of the arrangement that are outside the polygon. We refer to [O'R98] for efficient arrangement algorithms. By the *Zone Theorem* [ESS93], the number of visibility cells, $|\mathcal{C}|$, is bounded by $O(n^4)$.

Our visibility cell decomposition preserves the property shown in [ZG05], and is stated as Lemma 8.

Lemma 8. *All points in the same visibility cell have the same polygon edges entirely visible from them.*

Proof. We first show that the same polygon vertices are visible from any two points in the same visibility cell.

Suppose for any two points X_1 and X_2 of any visibility cell $c \in \mathcal{C}$, one polygon vertex $A \in \mathcal{A}$ is visible from X_1, but not X_2. See Fig. 4.2(a). Move along the line segment $\overline{X_1X_2}$ from X_2 to X_1, let X' denote the first point that A is visible from. There must be a reflex polygon vertex A' incident on line segment $\overline{AX'}$. The critical window defined by A and A' cuts $\overline{X_1X_2}$, which implies X_1 and X_2 cannot be in the same visibility cell.



(a) Visibility of vertex A (b) Visibility of edge $\overline{A_1A_2}$

Figure 4.2: The same vertices and edges are visible from all the points in a visibility cell.

Now we are ready to show the lemma. For any two points X_1 and X_2 of any visibility cell $c \in \mathcal{C}$, by the above argument, the same polygon vertices are visible from all the points on the segment $\overline{X_1X_2}$.

See Fig. 4.2(b). Assume $\overline{A_1A_2}$ is visible from X_1. We now show that it is also visible from X_2. Since by assumption $\overline{A_1A_2}$ is visible from X_1, the triangle $\triangle A_1A_2X_1$ is contained in $\mathcal{P}$. Since all the points on the segment $\overline{X_1X_2}$ is visible from A_2, the triangle $\triangle A_2X_1X_2$ is contained in $\mathcal{P}$. This implies the union of both triangles, the region $\square A_1X_1X_2A_2$ is contained in $\mathcal{P}$. So any point on $\overline{A_1A_2}$ is visible from X_2. $\square$

We now show that WEC-GWRP is equivalent to its restricted version where all the planned viewpoints are on the visibility segment, Lemma 9.

Lemma 9. *The optimal WEC-GWRP solution has a corresponding solution with the same cost, in which the planned viewpoints except the start position S are restricted to be on the visibility segments.*

Proof. First, we want to exclude the trivial case where to take a view at the start position S of the watchman route is the optimal WEC-GWRP solution. This can be easily done by checking if all the polygon edges are entirely visible from the start position S.

Now for any feasible WEC-GWRP solution, any planned viewpoint X that is not on any visibility segment cannot belong to the same visibility face as S. Otherwise, we can include S in and exclude X from the planned viewpoint set respectively and reduce path cost. Thus, the route connecting S and the planned viewpoint X must cross some bounding visibility segment of the visibility face that X belongs to. Note that according to our visibility definition, any polygon edge visible from X is visible from any point on the bounding visibility segment, and hence the crossing point. After replacing X with this crossing point, we have a feasible WEC-GWRP solution with the same cost and all planned viewpoints are on the visibility segments. $\square$

4.3.2 Sampling visibility segment domain

For a visibility segment l, as shown in Fig. 4.3, we draw a diamond shape consisting of two isosceles triangles with l as the common base. The sides of each triangle form an angle of $\alpha < 90$ degrees with the base. We will subsequently show how to determine α in Section 4.4. We define the domain of the visibility segment, denoted by $Dom(l)$, as the set of all points of polygon $\mathcal{P}$ inside the diamond (including the diamond boundary edges). In Fig. 4.3, $Dom(l)$ is the set of points in the diamond shape excluding the shaded area.

Lemma 10 states a simple, but crucial observation:

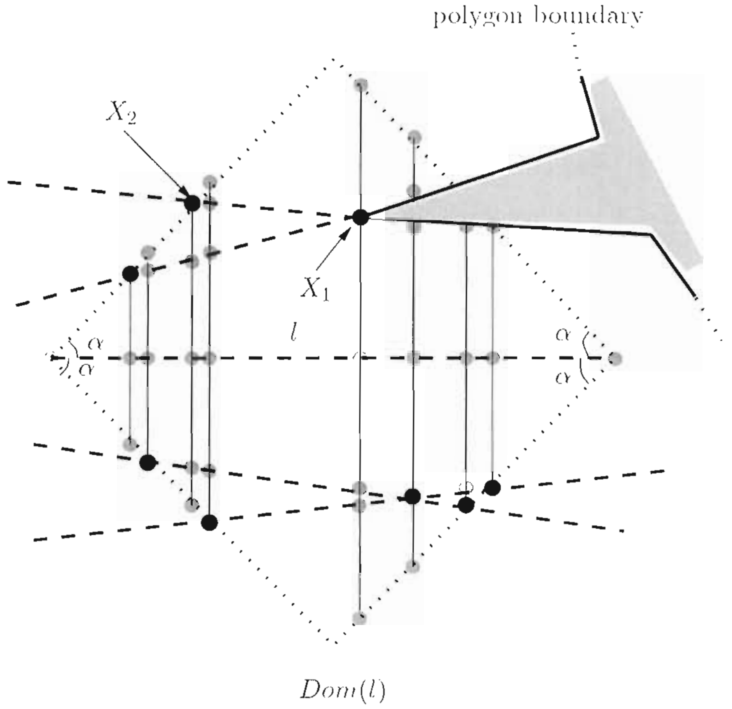


Figure 4.3: Domain of visibility segment l, $Dom(l)$. Inside $Dom(l)$, we draw vertical line segments from each visibility cell vertex, e.g., X_1, and from each intersection point between domain boundaries and visibility segments, e.g., X_2. The intersection points between these vertical line segments, other visibility segments, the polygon boundaries, and the domain boundaries are included in the viewpoint sample set.

Lemma 10. *For a visibility segment l, the slope (w.r.t. l) of any other visibility segment l' that intersects $Dom(l)$ is less than α, i.e.,*

$$\forall l' : l' \cap Dom(l) \neq \emptyset, |\angle l'l| \leq \alpha$$

Proof. Otherwise, the extended critical window collinear with l' intersects l and splits it into two edges. This contradicts that l is a single visibility segment. □

See Fig. 4.3. Inside each visibility segment domain $Dom(l)$, we draw vertical (perpendicular to l) lines from all the vertices of visibility cells and all the intersection points between the domain boundaries and the visibility segments. The number of such vertical lines is bounded by the number of visibility cell vertices plus twice the number of visibility segments (each visibility segment intersects at most two domain boundary edges). The segments of these vertical lines contained in $Dom(l)$, the other visibility segments, the polygon boundaries, and the boundaries of $Dom(l)$ intersect each other. We call these intersection points *sample viewpoints* and denote the set of sample viewpoints for all domains by Γ. The number of sample viewpoints in each domain is the number of vertices in the arrangements of the line segments described above, and is bounded by $(3|\mathcal{L}| + |\mathcal{L}| + n + 4)^2 = O(n^8)$, according to Zone Theorem [O'R98]. The terms in the brackets are the bounds on the number of vertical line segments in each domain, the number of other visibility segments, the number of polygon edges, the number of domain boundaries, respectively.) Thus, Γ is bounded:

$$|\Gamma| \leq |\mathcal{L}| \cdot O(n^8) = O(n^{12}).$$

We construct the complete graph $\mathcal{G}$ on Γ where the edge cost between two sample viewpoints is the shortest path distance between them in $\mathcal{P}$. This can be done by first constructing the *visibility graph* of Γ; and then the shortest path graph on the visibility graph. Note that all the reflex vertices are included in Γ. Now we have an *induced Traveling VPP* instance, with the set of viewpoints and traveling graph given as Γ and $\mathcal{G}$ respectively.

4.4 Sampling Algorithm Approximation Ratio Analysis

In this section, we show that the cost of the optimal solution to the induced Traveling VPP is at most a constant times that of the optimal solution to WEC-GWRP.

The idea is as follows. Assuming we have the optimal solution to the WEC-GWRP, we will find a solution to the induced Traveling VPP by first traversing the optimal route and partitioning it into pieces, then replacing each piece with a route passing through sample viewpoints while keeping the end-points of the piece fixed, and then moving the end-points to sample viewpoints. The partition process guarantees that the visibility segments that each piece passes through are ordered. The slope lemma, Lemma 10, then helps bound the length of the replacing piece w.r.t. that of the original piece on the optimal route.

For the optimal WEC-GWRP solution, let $\mathcal{V}^*$ denote the set of planned viewpoints, and let ϕ^* denote the shortest route connecting $\mathcal{V}^*$. As discussed in Section 4.1, we can assume all planned viewpoints are on the visibility segments. We denote the subset of visibility segments where the planned viewpoints are located by $\mathcal{L}^*$, i.e., $\mathcal{L}^* = \{l \in \mathcal{L} : \exists X : \mathcal{V}^*, X \in L\}$.

First, we give an observation of the route planned in the optimal solution, ϕ^*, Lemma 11.

Lemma 11. *The planned optimal route ϕ^* consists of a sequence of consecutive line segments. The end-points of these segments are either planned viewpoints, $\mathcal{V}^*$, or reflex vertices, $\mathcal{A}_r$.*

Proof. Note that ϕ^* is the shortest route connecting $\mathcal{V}^*$. The part of ϕ^* between any two consecutive planned viewpoints must be the shortest path in $\mathcal{P}$ connecting them, hence consists of a sequence of consecutive line segments, whose end-points are either the reflex vertices or these two points. Thus, ϕ^* as a whole has the property stated in the lemma. □

4.4.1 Partition of ϕ^*

In the following, we traverse ϕ^* and partition it in two steps. First, we partition it according to some visibility segment domains. Then, inside each domain, we further partition it using the vertical line segments incident on the cell vertices.

Partition according to domains

To help define the partition, we first introduce a labeling of some (not all) edges in $\mathcal{L}^*$. Note that all the edges in $\mathcal{L}^*$ are naturally ordered by the way ϕ^* traverses and intersects them. We start with the first visibility segment and label it l_1. Skip the following visibility segments whose corresponding planned viewpoints ($\mathcal{V}^*$) belong to the currently labeled domain, $Dom(l_1)$. We then label the next visibility segment that ϕ^* passes through after it exits $Dom(l_1)$ by l_2, and continue in this fashion. We partition ϕ^* according to the labeled visibility segments. Let $\phi_k^*, k = 1, \ldots, K$ denote these parts. Thus, the start- and end-points of ϕ_k^* are S_k and S_{k+1}, respectively. By definition, $S_k \in l_k \subset Dom(l_k)$ and $S_{k+1} \notin Dom(l_k)$ for $k \geq 1$. Please see Fig. 4.4 for an example.

Partition inside a domain

See Fig. 4.5. The vertical line segments in a domain partition it into vertical *strips*. The strips are bounded by three types of boundaries: the polygon boundary; the *strip boundary* that separates two neighboring strips; and the *domain boundary*, the boundary of the diamond shape defining a domain.

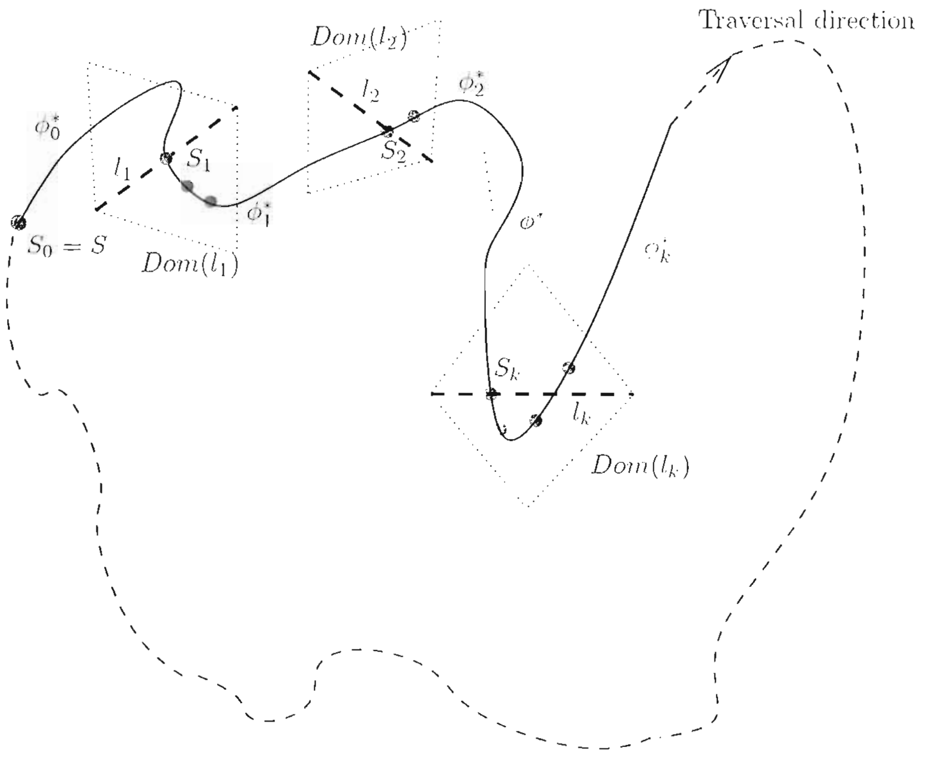


Figure 4.4: Partition of the optimal WEC-GWRP route ϕ^* according to the visibility segments and corresponding domains it crosses. Note that although shown disjointed, the labeled domains may intersect each other and planned viewpoints on ϕ_k^* may be contained in previous labeled domains.

Note that all the visibility segments are ordered inside a strip, since otherwise a vertical line at the intersection point where the order changes would have divided the strip into smaller strips. We order the visibility segments in a strip according to their intersection point with the left strip or domain boundary. The position relation between visibility segments, *above/below*, is thus well defined inside a strip.

We further partition each part of the optimal route, ϕ_k^*, inside the domain $Dom(l_k)$ by its strips. For a strip st, we denote the partitioned piece by $\phi_k^*(st) = \phi_k^* \cap st$. We denote the start-point and end-point of $\phi_k^*(st)$ by L_{st} and R_{st} respectively, also called the entry and exit points respectively in the following. We denote the highest and lowest visibility

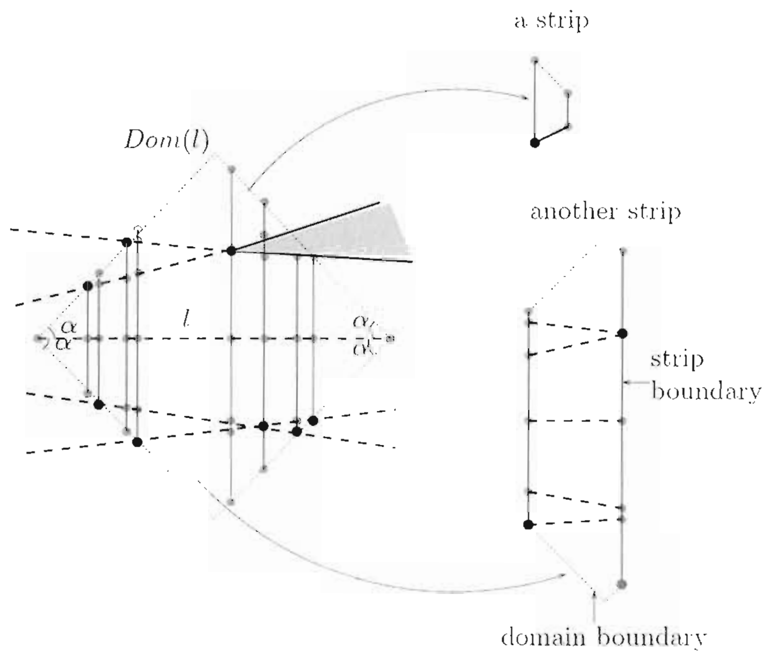


Figure 4.5: The vertical line segments from visibility cell vertices partition a domain into strips, bounded by domain boundary and/or polygon and strip boundary.

segments that $\phi_k^*(st)$ crosses (if applicable) by l_{high} and l_{low} respectively.

Categories and properties of $\phi_k^*(st)$

$\phi_k^*(st)$ can be categorized into five cases, according to whether its entry point is on the labeled visibility segment or on the strip boundary and whether its exit point is on the strip boundary or on the domain boundary. See Fig. 4.6 for illustrations. For cases (Ia) and (Ib), $\phi_k^*(st)$ enters on l_k (at point S_k, i.e., $L_{st} = S_k$), and exits either through the strip boundary, Case (Ia), or through the domain boundary, Case (Ib). For cases (IIa), (IIb) and (IIc), $\phi_k^*(st)$ enters st through a strip boundary and exits through either the other strip boundary, Case (IIa), or the same strip boundary, Case (IIb), or the domain boundary, Case (IIb).

We now show that $\phi_k^*(st)$ consists of at most three straight line segments:

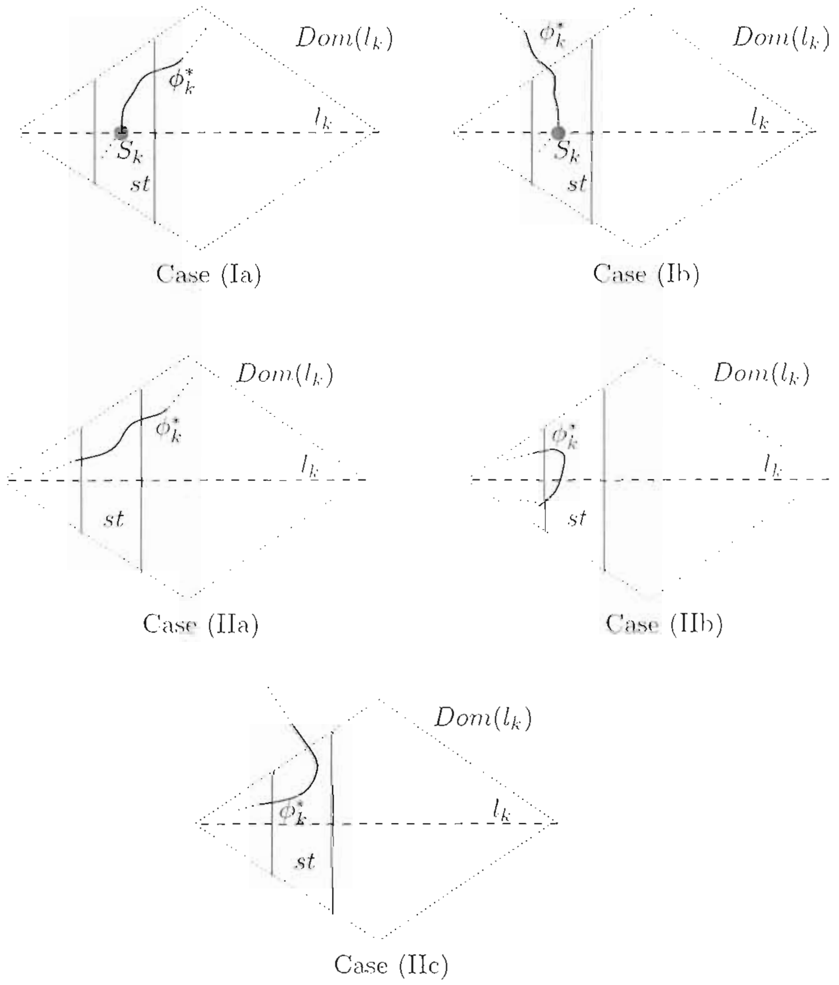


Figure 4.6: Five cases of $\phi_k^*(st)$. Case (Ia): $\phi_k^*(st)$ starts at S_k and exits from the strip boundary. Case (Ib): $\phi_k^*(st)$ starts at S_k and exits through the domain boundary. Case (IIa): $\phi_k^*(st)$ enters and exits through different strip boundaries. Case (IIb): $\phi_k^*(st)$ enters and exits through the same strip boundary. Case (IIc): $\phi_k^*(st)$ enters through the strip boundary and exits through the domain boundary.

Lemma 12. *$\phi_k^*(st)$ consists of at most three straight line segments. If $\phi_k^*(st)$ consists of three segments, the exit point must lie between the highest and lowest visibility segments $\phi_k^*(st)$ touches, l_{high} and l_{low}.*

Proof. The proof is by contradiction. If the above mentioned condition is not satisfied, we can replace $\phi_k^*(st)$ by a strictly shorter route in st with the same entry and exit points, which crosses the same visibility segments as $\phi_k^*(st)$. This implies that ϕ^* cannot be optimal, because we can replace $\phi_k^*(st)$ by this shorter route and choose viewpoints on the visibility segments that are collinear with the viewpoints taken on $\phi_k^*(st)$ and hence reduce the solution cost.

Wlog, suppose $\phi_k^*(st)$ goes upwards first after it enters st at point L_{st}. When $\phi_k^*(st)$ changes directions, it has to go downwards. Otherwise, as shown in Fig. 4.7(a), we can simply replace the two upward sloping segments (solid line segments) by a single line segment (dashed-dotted line segment) and the route is shorter. Since this single line segment passes all the visibility segments the two segments pass, it has exactly the same set of entirely visible polygon edges. Similarly, $\phi_k^*(st)$ cannot change direction when going downwards. We can also easily show that if afterwards $\phi_k^*(st)$ goes upwards again, it has to exit st. Otherwise, as shown in Fig. 4.7(b) and (c), we can replace it with a shorter route.

If $\phi_k^*(st)$ consists of three segments, as shown in Fig. 4.7(d), and it exits the strip by intersecting the highest (resp. lowest) visibility segment, we can replace the first (resp. last) two segments by a single segment and reduce solution cost. So the exit point must lie between l_{high} and l_{low}.

Although Fig. 4.7 shows L_{st} is on the strip boundary, it generalizes trivially to cases where $L_{st} = S_k$. □

4.4.2 Approximating $\phi_k^*(st)$ using a route connecting sample viewpoints

We show, case by case, how to approximate $\phi_k^*(st)$ using a route, denoted by $\phi_k'(st)$, which connects sample viewpoints in Γ and the start- and end-points of $\phi_k^*(st)$.

Approximation for Case (Ia)

This is the case where a smaller α gives us a better approximation ratio. $\phi_k^*(st)$ consists of two segments, Fig. 4.8(a), or three segments, Fig. 4.8(b). The route $\phi_k'(st)$ consists of the

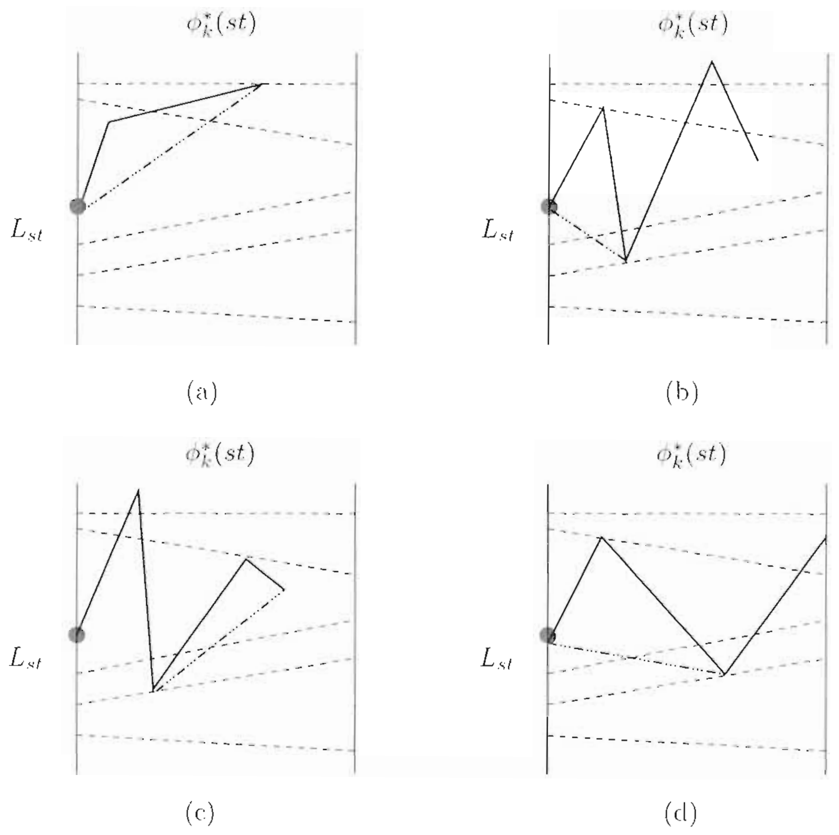


Figure 4.7: φ'_{st} consists of at most 3 line segments.

straight line segment between S_k and R_{st} and one or two detours on the strip boundary from R_{st}.

We now show that the alternative route can approximate the corresponding part of the optimal WEC-GWRP solution within a constant ratio (for a given α).

Lemma 13. *For case (Ia), $\phi'_k(st)$ can approximate $\phi_k^*(st)$ within the ratio of* $1 + \frac{4}{\cos\alpha}$.

Proof. In the following, we show for the subcases when $\phi_k^*(st)$ consists of two segments and when it consists of three segments, respectively. The one-segment case is true trivially.

See Fig. 4.9. $\phi_k^*(st)$ consists of two segments, a and b; $\phi'_k(st)$ consists of the segment c and a detour consisting of segments d_1 and d_2. β_1 corresponds to the slope of l_{high} and is measured counterclockwise from d, thus bounded by α, i.e., $-\alpha \leq \beta_1 \leq \alpha < \pi/2$. β_2 is

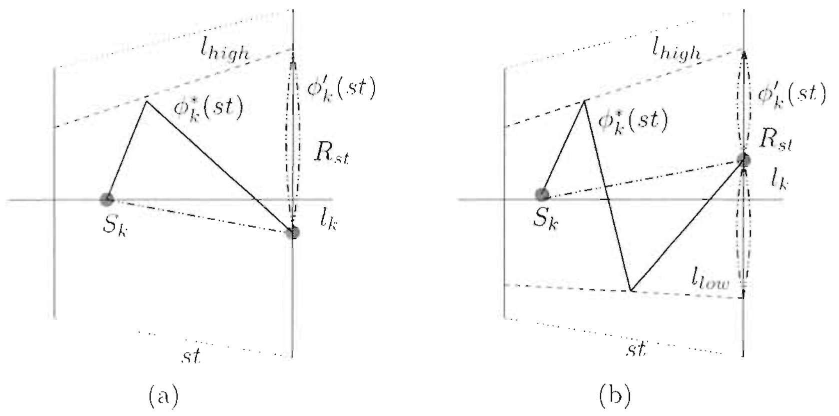


Figure 4.8: Approximate $\phi^*_k(st)$ for case (Ia). (a) $\phi^*_k(st)$ consists of two segments to reach l_{high}; $\phi'_k(st)$ consists of the line segment $\overline{S_kR_{st}}$ and a detour going upwards to l_{high} and back to R_{st}. (b) $\phi^*_k(st)$ consists of three segments to reach both l_{high} and l_{low}; $\phi'_k(st)$ consists of $\overline{S_kR_{st}}$ and two detours, one going upwards to l_{high} and back, and the other going downwards to l_{low} and back.

measured clockwise from d. So its range is between $-\alpha$ and $\pi/2$, i.e., $-\pi/2 \le -\alpha \le \beta_2 \le \alpha \le \pi/2$.

We have the following relations:

$$\begin{aligned}
&\|c\| \le \|a\| + \|b\| \\
&\|d_1\| = \|d\| \cdot \tan\beta_1 = \|b\| \cos\beta_2 \tan\beta_1 \\
&\|d_2\| = \|b\| \sin\beta_2
\end{aligned}$$

Since $-\alpha \le \beta_1 \le \alpha < \pi/2$, $\cos\beta_1 \ne 0$, we have,

$$\begin{aligned}
\|d_1 + d_2\| &= \|b\| \cdot |\cos\beta_2 \tan\beta_1 + \sin\beta_2| \\
&= \frac{\|b\|}{\cos\beta_1} |\sin(\beta_1 + \beta_2)| \\
&\le \frac{\|b\|}{\cos\beta_1} \le \frac{\|a\| + \|b\|}{\cos\beta_1} \le \frac{\|a\| + \|b\|}{\cos\alpha}.
\end{aligned}$$

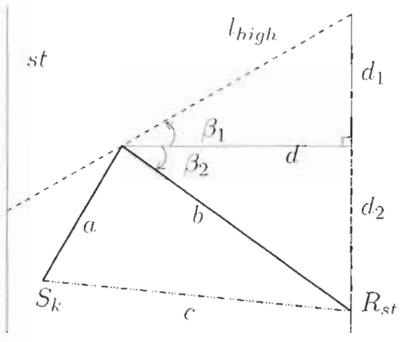


Figure 4.9: Approximate $\phi_k^*(st)$ for case (Ia) when $\phi_k^*(st)$ consists of two segments.

So we have,

$$\begin{aligned}\|\phi_k'(st)\| &= \|c\| + 2(\|d_1\| + \|d_2\|) \\ &\leq (\|a\| + \|b\|)(1 + \frac{2}{\cos\alpha}) \\ &= \|\phi_k^*(st)\|(1 + \frac{2}{\cos\alpha}),\end{aligned}$$

i.e., $\frac{\|\phi_k'(st)\|}{\|\phi_k^*(st)\|} \leq (1 + \frac{2}{\cos\alpha})$.

The case for the equality occurs for condition as shown in Fig. 4.10.

For the second sub-case where $\phi_k^*(st)$ consists of three segments, $\phi_k'(st)$ includes the segment $\overline{S_kR_{st}}$ and detours at R_{st} on the strip boundary to reach (and then come back) l_{high} and l_{low} respectively. In the following, we analyze the approximation ratio. We use an intermediate tour, a path consisting of $\overline{S_kR_{st}}$, a detour at S_k to reach l_{high} and back, and a detour at R_{st} to reach l_{low} and back. We first show the approximation ratio of this tour to $\phi_k^*(st)$, and then the approximation ratio of $\phi_k'(st)$ to this intermediate tour.

Let M denote the intersection point between the straight line segment $\overline{S_kR_{st}}$ and $\phi_k^*(st)$. We divide $\phi_k^*(st)$ into two parts, the part from S_k to M and the part from M to R_{st}. Note that each part consists of two line segments. We can approximate these two parts by the corresponding parts of the intermediate tour shown in Fig. 4.11(b1) and (b2), respectively. It consists of a straight line segment between S_k and M and a vertical detour at S_k to reach l_{high} and back, Fig. 4.11(b1), followed by a straight line segment between M and R_{st} and a vertical detour at R_{st} to reach l_{low} and back, Fig. 4.11(b2). From the above analysis for the two-segment case, we know the approximations are within $(1 + \frac{2}{\cos\alpha})$-ratio for both, thus

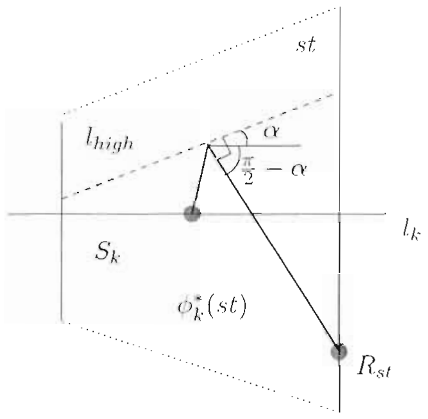


Figure 4.10: The worst case of the approximation of case (Ia). S_k is arbitrarily close to l_{high}; l_{high} is parallel to the domain boundary, i.e. its slope is α; and the line segment connecting R_{st} has slope $\frac{\pi}{2} - \alpha$, i.e. it is perpendicular to l_{high}.

the intermediate tour can approximate $\phi_k^*(st)$ within the same ratio. $(1 + \frac{2}{\cos\alpha})$.

Note the only difference of the intermediate tour from $\phi_k'(st)$ is that its detour at S_k, the solid loop shown in Fig. 4.11(b1), is not at the exit point. See Fig. 4.12. We now replace the detour at S_k, the solid loop, with a detour at R_{st} to reach l_{high} at point A and back, the dashed-dotted loop. From S_k, we draw a line segment l' parallel to l_{high} and it intersects the right strip boundary at point B. The part of the dashed-dotted loop between points A and B has the same length as the solid loop. In the triangle $\triangle S_k B R_{st}$, let θ denote the angle between $\overline{S_k B}$ and $\overline{S_k R_{st}}$ and β denote the angle between $\overline{S_k B}$ and $\overline{BR_{st}}$. By the law of sines, $\|\overline{BR_{st}}\| = \frac{\sin\theta}{\sin\beta}\|\overline{S_k R_{st}}\| \leq \frac{\|\overline{S_k R_{st}}\|}{\sin\beta} \leq \frac{1}{\cos\alpha}\|\phi_k^*(st)\|$. The last inequality is due to the slope lemma, Lemma 10, and the fact that $\|\overline{S_k R_{st}}\|$ is at most $\|\phi_k^*(st)\|$. Thus, $\|\phi_k'(st)\|$ is at most the length of the intermediate tour, which is at most $(1 + \frac{2}{\cos\alpha})\|\phi_k^*(st)\|$, plus the length of the part of the dashed-dotted loop between B and R_{st}, which is at most $\frac{2}{\cos\alpha}\|\phi_k^*(st)\|$. Consequently, $\|\phi_k'(st)\| \leq (1 + \frac{4}{\cos\alpha})\|\phi_k^*(st)\|$.

To summarize, we have that $\|\phi_k'(st)\|$ is at most $\|\phi_k^*(st)\|$ times $1 + \frac{4}{\cos\alpha}$. □

Approximation for Case (Ib)

Wlog, here we only consider the case where $\phi_k^*(st)$ exits from the top left domain boundary. By Lemma 12, if $\phi_k^*(st)$ has three segments, the exit point must lie between the highest and

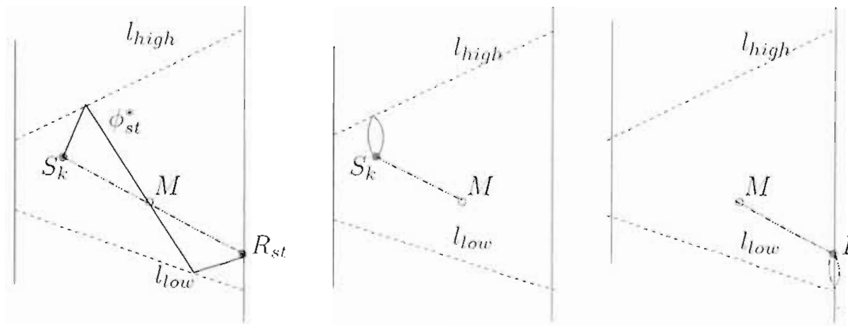


Figure 4.11: Approximate $\phi^*_k(st)$ for case (Ia) when $\phi^*_k(st)$ consists of three segments using an intermediate tour. The two parts of this intermediate tour are shown in (b1) and (b2) respectively.

the lowest visibility segments visited by $\phi^*_k(st)$. Since we know that the exit point is on the domain boundary, higher than l_{high}, $\phi^*_k(st)$ consists of at most two segments, as shown in Fig. 4.13.

We approximate $\phi^*_k(st)$ using the straight line segment connecting S_k and either the left or the right strip boundary vertex on the domain boundary where R_{st} is on, whichever gives us a shorter route, and a detour, if necessary, at that vertex to reach l_{low} on the strip boundary. In Lemma 14, we show the approximation ratio using the left domain vertex on strip st.

Lemma 14. *For case (Ib), $\phi'_k(st)$ can approximate $\phi^*_k(st)$ within the constant ratio of $\frac{1}{\sin\frac{\alpha}{2}} + \frac{2}{\cos\alpha}$.*

Proof. Note that the first sub-case where $\phi^*_k(st)$ consists of one segment, Fig. 4.13(a), is a special case of the second sub-case where $\phi^*_k(st)$ consists of two segments, Fig. 4.13(b), (when points S_k, C and B coincide). So in the following, we only show the approximation ratio for the two-segment sub-case. As shown in Fig. 4.13(b), $\phi'_k(st)$ consists of the two line segments $\overline{S_kV_{st}}$ and $\overline{V_{st}R_{st}}$, and the detour from V_{st} to reach l_{low} at point D and back. In the following, we bound them respectively.

First, we bound the total length of segments $\overline{S_kV_{st}}$ and $\overline{V_{st}R_{st}}$ w.r.t. $\|\phi^*_k(st)\|$. See Fig. 4.14(a). Let θ denote the angle between the two line segments $\overline{S_kV_{st}}$ and $\overline{V_{st}R_{st}}$, and $\alpha \leq \theta \leq \pi/2 + \alpha < \pi$. We have

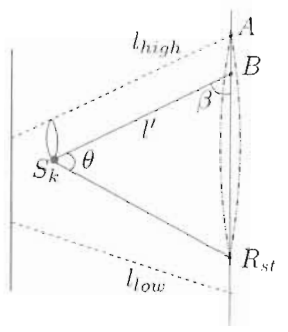


Figure 4.12: Approximate first part of $\phi_k^*(st)$.

$\|\overline{S_kR_{st}}\| = \sqrt{\|\overline{S_kV_{st}}\|^2 + \|\overline{V_{st}R_{st}}\|^2 - 2\|\overline{S_kV_{st}}\|\,\|\overline{V_{st}R_{st}}\|\cos\theta}$, i.e., $\|\overline{S_kR_{st}}\|$ is monotonically increasing w.r.t. θ.

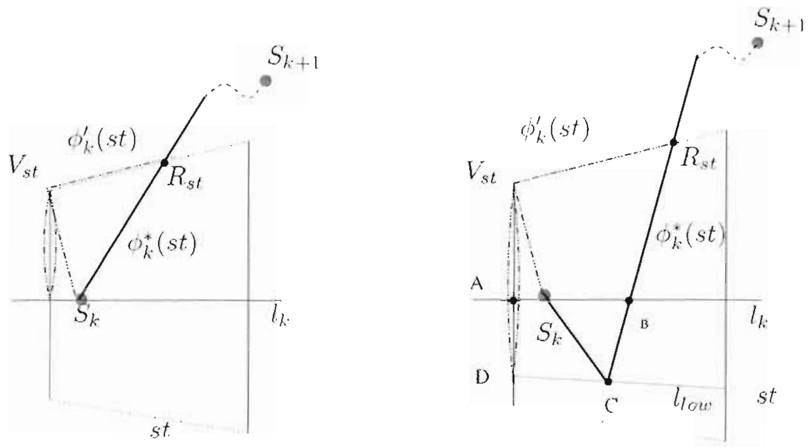
Note that α is the lower bound on θ. So if we fix $\|\overline{S_kV_{st}}\|$ and $\|\overline{V_{st}R_{st}}\|$ and vary θ, the ratio $\frac{\|\overline{S_kV_{st}}\|+\|\overline{V_{st}R_{st}}\|}{\|\overline{S_kR_{st}}\|}$ is largest when $\theta = \alpha$. This is the case where st is the leftmost strip in the domain $Dom(l_k)$ as shown in Fig. 4.14(b). For this case, i.e., when $\theta = \alpha$, we draw a perpendicular line segment from S_k to $\overline{V_{st}R_{st}}$ and denote the angle between it and $\overline{S_kR_{st}}$ by β. The value of β is measured clockwise from the perpendicular line segment and can be negative. β achieves its minimum value when point R_{st} is arbitrarily close to V_{st} and at this time $\beta = -(\pi/2 - \alpha)$. So, $\alpha - \pi/2 < \beta < \pi/2$. Using basic trigonometry formulae, we have

$$\|\overline{S_kR_{st}}\| = \frac{\sin\alpha}{\cos\beta}\|\overline{S_kV_{st}}\|,$$
$$\|\overline{S_kV_{st}}\| + \|\overline{V_{st}R_{st}}\| = (1 + \cos\alpha + \sin\alpha\tan\beta)\cdot\|\overline{S_kV_{st}}\|.$$

The ratio between them is bounded as follows:

$$\begin{aligned}\frac{\|\overline{S_kV_{st}}\| + \|\overline{V_{st}R_{st}}\|}{\|\overline{S_kR_{st}}\|} &= \frac{\cos\beta + \cos\alpha\cos\beta + \sin\alpha\sin\beta}{\sin\alpha} = \frac{\cos\beta + \cos(\alpha-\beta)}{\sin\alpha}\\ &= \frac{2\cos\frac{\alpha}{2}\cos\frac{\alpha-2\beta}{2}}{2\sin\frac{\alpha}{2}\cos\frac{\alpha}{2}} = \frac{\cos\frac{\alpha-2\beta}{2}}{\sin\frac{\alpha}{2}}\\ &\le \frac{1}{\sin\frac{\alpha}{2}}.\end{aligned}$$

The equality above is achieved when $\beta = \frac{\alpha}{2}$. By triangular inequality, $\|\overline{S_kR_{st}}\|$ is at



(a) (b)

Figure 4.13: Approximate $\phi^*_k(st)$ for Case (Ib). (a) $\phi^*_k(st)$ consists of one segment. (b) $\phi^*_k(st)$ consists of two segments.

most $\|\phi^*_k(st)\|$. Hence, we have:

$$\|\overline{S_kV_{st}}\| + \|\overline{V_{st}R_{st}}\| \le \frac{1}{\sin\frac{\alpha}{2}}\|\phi^*_k(st)\|. \tag{4.2}$$

Similarly to the proof of Lemma 13, the length of the detour at V_{st} (to point D and back) is at most $\|\phi^*_k(st)\|$ times $\frac{2}{\cos\alpha}$.

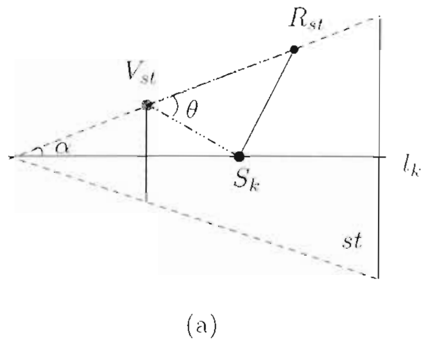
Summing up both bounds, $\|\phi'_k(st)\|$ is at most $\|\phi^*_k(st)\| \cdot (\frac{1}{\sin\frac{\alpha}{2}} + \frac{2}{\cos\alpha})$. □

Case (IIa)

Note that in the analysis of case (Ia), we did not use the fact that S_k is on l_k. So the result for case (Ia), Lemma 13, generalizes to case (IIa). For case (IIa), we have the freedom to detour at either the entry or the exit points of $\phi^*_k(st)$. We can arbitrarily choose one.

Case (IIb)

See Fig. 4.15 (a) and (b) for cases where $\phi^*_k(st)$ consists of two and three segments respectively. Similar to the proof of Lemma 13 for Case (Ia), $\|\phi'_k(st)\|$ is at most $\|\phi^*_k(st)\|$ times $1 + \frac{4}{\cos\alpha}$.



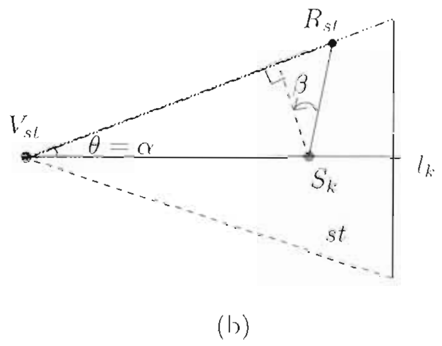


Figure 4.14: Illustration of the bound on the length of route $S_k V_{st} R_{st}$.

Case (IIc)

Result for case (Ib) generalizes to (IIc).

4.4.3 Chaining together the partitions

By its construction, $\phi'_k(st)$'s chained together form a continuous route, denoted by ϕ', a sequence of line segments with loops at their end-points shown in Fig. 4.16 (a). However, the start-point and end-point of $\phi'_k(st)$ may not belong to the sample viewpoint set Γ. We construct a solution $\hat{\phi}$ from ϕ' as follows. For S_k and end-points without detours, e.g. point C and S_k in Fig. 4.16(a), we can simply ignore it, since it is taken care of by the end-points of its previous and next strips. For example, in Fig. 4.16, let A and B denote the planned viewpoints that are immediately before and immediately after S_k on ϕ' respectively. We can bypass S_k and connect directly A and B, (the dotted line segment). For end-points with detours, we move them to the nearest sample viewpoint *inside* the loops. As illustrated in Fig. 4.16(b), the segments $\overline{WX}$ and $\overline{XZ}$ are replaced with $\overline{WY}$ and $\overline{YZ}$ respectively. By the triangular inequality, $\|\overline{WY}\| + \|\overline{YZ}\|$ is at most $\|\overline{WX}\| + \|\overline{XZ}\| + 2\|\overline{XY}\|$. Thus $\|\hat{\phi}\|$ adds to $\|\phi'\|$ at most the length of all the detours of ϕ' and is at most $2\|\phi'\|$.

It is clear that $\hat{\phi}$ is a solution to the induced Traveling VPP, i.e., all the planned viewpoints are from the sample viewpoint set. Using this solution, we can bound the cost of the optimal solution to the induced Traveling VPP w.r.t. the true optimal to the WEC-GWRP.

Theorem 5. *The cost of the optimal solution to the induced Traveling VPP is at most* 11.657 *times that of the optimal solution to the WEC-GWRP.*

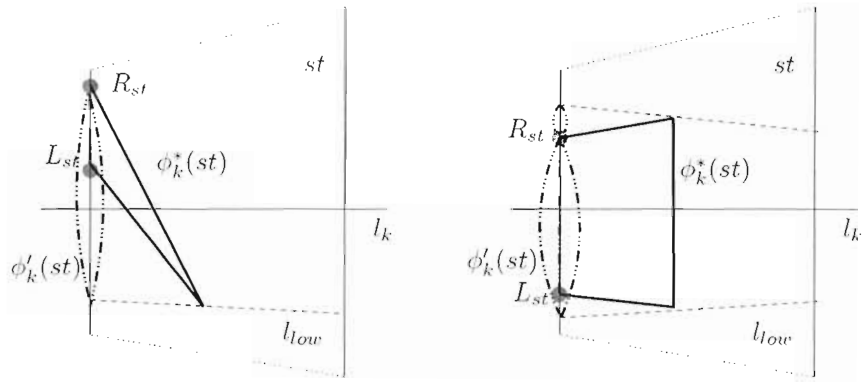


Figure 4.15: (a) case (IIb) when $\phi_k^*(st)$ consists of two segments. (b) case (IIb) when $\phi_k^*(st)$ consists of three segments.

Proof. By summarizing cases (Ia), (Ib), (IIa), (IIb) and (IIc), $\|\phi_k'(st)\|$ is at most $\|\phi_k^*(st)\|$ times $\max(1+\frac{4}{\cos\alpha}, \frac{1}{\sin\frac{\alpha}{2}}+\frac{2}{\cos\alpha})$. After chaining and moving the entry and exit points, the length of $\hat{\phi}$ is at most twice that of $\phi_k'(st)$. Since $\hat{\phi}$ is a feasible solution to the induced Traveling VPP, its cost is a lower bound on the optimal solution cost to the induced Traveling VPP. Thus, the cost of the optimal solution to the induced Traveling VPP is at most $2\max(1+\frac{4}{\cos\alpha}, \frac{1}{\sin\frac{\alpha}{2}}+\frac{2}{\cos\alpha})$ times the cost of the true optimal solution to the WEC-GWRP.

Now we choose the α value to minimize this ratio, i.e., to solve $2\min_\alpha \max(1+\frac{4}{\cos\alpha}, \frac{1}{\sin\frac{\alpha}{2}}+\frac{2}{\cos\alpha})$. The solution is ≈ 11.657, when $\alpha \approx 34^o$ (via numerical minimization using Matlab). □

After constructing the induced Traveling VPP using the sample viewpoints, we call the approximation algorithm in Chapter 2 to get a solution. The approximation ratio result of this Traveling VPP solver and Theorem 5 implies that the cost of the solution is at most the cost of the optimal WEC-GWRP solution times either the order of the view frequency or a polynomial of $\log n$, whichever is smaller.

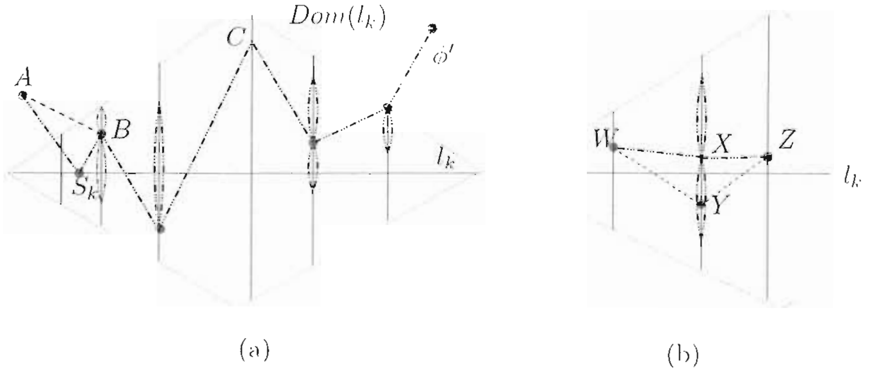


Figure 4.16: The approximation solution within a domain $Dom(l_k)$, after chaining together $\phi'_k(st)$ for the strips ϕ^* crosses.

4.5 Comparison with triangulation based sampling algorithm [AMS05]

To illustrate our sampling and compare with other relevant sampling algorithms such as the triangulation-based algorithm proposed by Aleksandrov *et al.* [AMS05], we give a simple example in which the latter requires substantially more sampling viewpoints than our sampling algorithm to achieve the same approximation ratio.

4.5.1 Recapitulation of triangulation-based sampling algorithm

In the following, we briefly cover the sampling algorithm proposed by Aleksandrov [AMS05] in the context of GWRP. Note that it has an approximation ratio of $(1+\epsilon)$. To achieve the same approximation ratio as our GWRP algorithm, ≈ 11.657, $\epsilon \approx 10.657$.

First, the region, the visibility cell decomposition in our case, is triangulated. Fig. 4.17 shows a triangulated region with triangular faces $\triangle A_1XA_2$, $\triangle A_2XA_3$, $\triangle A_3XA_4$, and $\triangle A_4XA_1$. Second, at each vertex, for example X in Fig. 4.17, calculate the shortest distance to the edges of its incident triangles excluding the incident edges to the vertex, and denote it by $d(X)$. In Fig. 4.17, $d(X)$ is the length of $\overline{XD}$, perpendicular to edge $\overline{A_3A_4}$. Then, in each triangulated face, for each vertex X, draw the angular bisector $\overline{XX'}$ (dividing the angle $\eta \triangleq \angle A_2XA_1$) and compute sampling viewpoints along the bisector in the following way. Let $r(X)$ denote the region (the shaded region in Fig. 4.17) around vertex X, a isosceles triangle with side length of $\frac{\epsilon}{7}d(X)$. $\overline{XX'}$ intersects the edge of $r(X)$ that is *not* incident on X at point P_0. P_0 is the also the first sampling viewpoint on $\overline{XX'}$. The ith sampling viewpoint P_i is the one such that $|\overline{XP_i}| = \left(1 + \sin(\eta/2)\sqrt{\epsilon/2}\right)^i \cdot |\overline{XP_0}|$. The distances from the sampling viewpoints to X form a geometric series and the total number is given by $\log_{1+\sin\frac{\eta}{2}\sqrt{\frac{\epsilon}{2}}}\left(\frac{|XX'|}{\cos\frac{\eta}{2}\cdot\frac{\epsilon}{7}\cdot d(X)}\right)$.

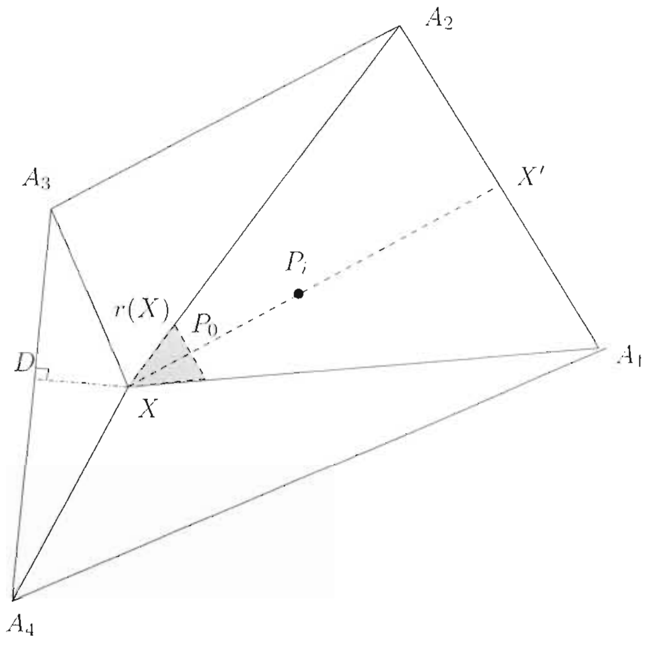
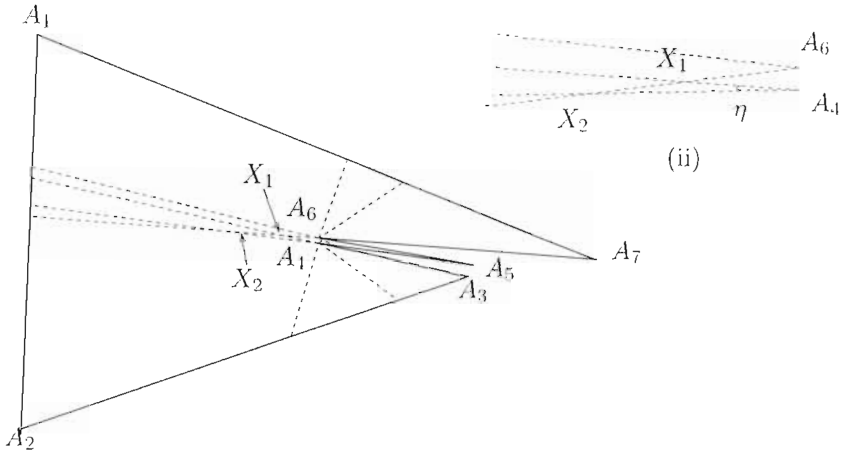


Figure 4.17: Illustration of the sampling algorithm proposed by Aleksandrov [AMS05].

4.5.2 Comparison using simple example example

Fig. 4.18(i) shows a simple polygon with its visibility decomposition. We now compare the number of computed viewpoints in the neighborhood of triangle $\triangle X_1X_2A_4$ by the two algorithms. The angle $\angle X_1A_4X_2$ is denoted by η and is roughly $1.15^o \approx 0.02rad$.

We now show the sampling viewpoints computed by our GWRP algorithm given in this chapter and by the sampling algorithm proposed by Aleksandrov [AMS05]. Fig. 4.19(i) shows the sampling viewpoints computed by the GWRP algorithm for visibility segment



(i)

Figure 4.18: (i) A simple polygon ($A_1A_2A_3A_4A_5A_6A_7$) with its visibility cell decomposition. (ii) The zoomed figure of triangle $\triangle X_1X_2A_4$ formed by extensions of $\overline{A_3A_4}$, $\overline{A_5A_4}$ and $\overline{A_7A_6}$.

$\overline{X_2A_4}$. There are 22 viewpoints in total. Fig. 4.19(ii) shows those computed by Aleksandrov *et al.*'s triangulation-based algorithm [AMS05]. Using $\epsilon \approx 10.657$, and $d(X) \approx \frac{A_4X'}{4}$, the number of sampling viewpoints is approximately 43. For even more *ill-shaped* polygons, for example, where $\eta \approx 0.04^o$, the number of viewpoints is approximately 1203. However, the number of sampling viewpoints computed by the GWRP algorithm is relatively unchanged.

From the above simple example, it is clear that the independence of the number of computed sampling viewpoints and the geometric parameters of the input polygon makes the GWRP algorithm have guaranteed performances for even *ill-shaped* polygons.

4.6 Application of sampling algorithm to generalized 2.5D terrain guarding problem

We believe that the sampling algorithm proposed here is a general technique and can also be used for other shortest route problems where one would like to get an approximation algorithm by first reducing the continuous input space to a discrete sample viewpoint set, and then solving the resulting discrete problem. For example, the algorithm can be applied to the following generalized version of the 2.5D terrain guarding problem [Eid02] with additional

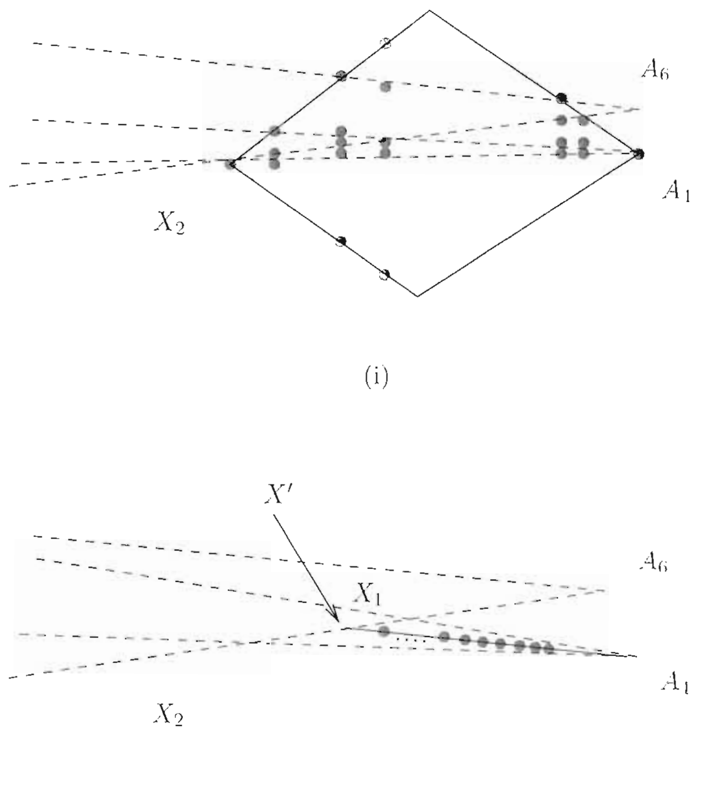


Figure 4.19: Computed sampling viewpoints by the GWRP algorithm, (i) and sampling algorithm Aleksandrov [AMS05], (ii).

travel cost in the objective function.

Suppose an autonomous aerial robot flying at a certain height h is asked to inspect a 2.5D terrain, i.e., to fly on a tour at this height and stop at some points to scan the terrain. A 2.5D terrain is defined as follows. A set of discrete points on the horizontal ground plane (of height zero) is first triangulated, and then each point is *lifted* by some height. The resulting 2D manifold in 3D is called the 2.5D terrain [Eid02]. We assume h is greater than the height of any point of the terrain. An existing algorithm for Terrain Guarding with Triangle Restriction, [Eid02], first partitions the plane at height h into cells such that all the points in the same cell *wholly* see/cover the same subset of terrain triangles, and then chooses only the cell vertices as the viewpoints to reduce the problem to a SCP instance. The greedy algorithm for SCP approximates the problem within a ratio of the logarithm of the number of terrain triangles.

We introduce the problem of Generalized Terrain Guarding with Triangle Restriction (TR-GTG), of planning a tour at the horizontal plane of height h and a set of discrete viewpoints on it such that every triangle of the terrain is visible from at least one planned viewpoint, while minimizing the total cost as a weighted sum of the view cost, proportional to the number of planned viewpoints, and the travel cost, the length of path planned. To solve the TR-GTG problem, we first apply the visibility cell decomposition algorithm to the plane at height h and partition it into visibility cells. We then use the sampling algorithm proposed in this paper to reduce the problem to a finite set of sample viewpoints, and then construct a Traveling VPP instance. The Traveling VPP solver in Chapter 2 gives us an approximation algorithm with the approximation ratio in the order of either the view frequency or a polynomial of $\log n$, whichever is smaller.

Chapter 5

Simulation Results for GWRP

To test our approximation algorithm for the GWRP, and more importantly, as the first step toward implementations on real robot systems, we develop a preliminary implementation of the algorithm for real environments such as the the Robotic Algorithms and Motion Planning (RAMP) lab at School of Engineering School, Simon Fraser University. The implementation takes either 2D polygons or real robot generated 2D maps as input. In the latter case it first computes a 2D polygon approximation of the real map for a valid GWRP input. It then computes a GWRP solution, i.e., a set of points in the input polygon as the planned viewpoints, connected via a route. For visualization, we simulate the solution, i.e., draw 2D polygon on the screen, the planned viewpoints, and the route connecting them. Hence, we refer to this implementation as a simulator. In the following, we give details of this simulator implementation and the results for some polygons.

5.1 Simulator Implementation

The simulator consists of three parts, the polygon generation, the sampling algorithm, LP optimizer, and the algorithm Round and Connect.

5.1.1 Polygon generation

We offer two types of polygon inputs: a User Interface (UI) that allow users to draw a polygon; and a text file listing the coordinates of the polygon vertices in either the clockwise or the counterclockwise order. The UI is a window on the screen that takes the sequence of mouse clicks as the polygon vertices, i.e., the coordinates of the mouse specify those of the corresponding polygon vertex. See Fig. 5.1 for such a manually created polygon. The text file input mode can be used for maps generated from the real scenes used in the robot mapping and navigation applications.

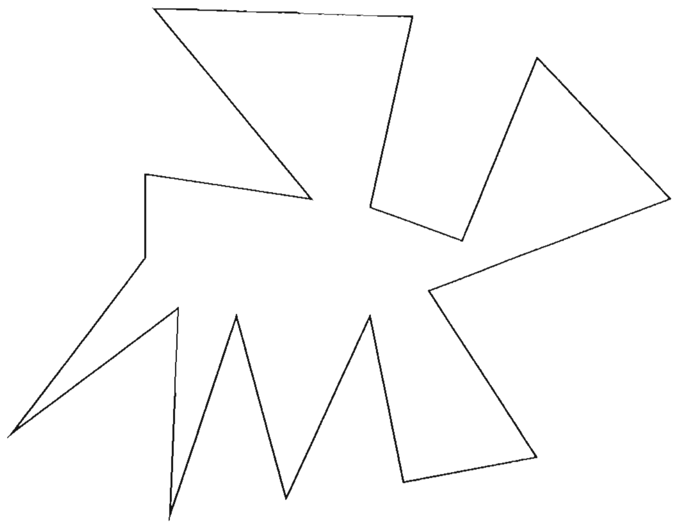


Figure 5.1: A polygon input created manually.

A mobile robot, an ActivMedia PowerBot [Mob], equipped with a Hokuyo laser range finder [Hok], is used to gather the map data. We can either manually control or run an *active* simultaneous localization and mapping (SLAM) algorithm [TBF05] to automatically control the robot to move around the lab area. The collected laser range finder's scanning data are fed to a SLAM algorithm to generated a *map*. For the active SLAM case, the map can also be generated on the fly. Afterwards, we run the DP-SLAM v2.0 developed at Duke University [EP04] on the data to get the maps. For example, in Fig. 5.2, we show the generated map of a part of the RAMP lab layout, School of Engineering Science, Simon Fraser University. In the figure shown, the grey level of each pixel indicates the occupancy status of the corresponding region in the map, with the colors *white*, *dark*, and *grey* denoting *unknown region*, *obstacle* and *free region* respectively.

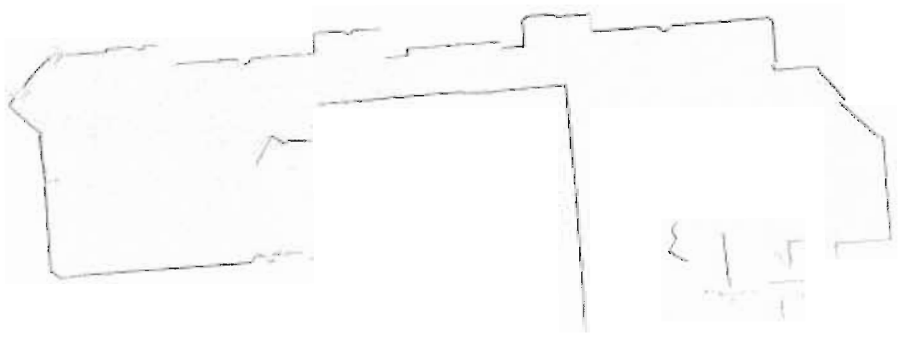


Figure 5.2: A map generated by DP-SLAM.

We then use the algorithm proposed by Latecki and Lakaemper [LL06] on the generated map to get its polygon representation, i.e., the polygon boundary. The polygon computed for the map in Fig. 5.2 is shown in Fig. 5.3.

5.1.2 Sampling Algorithm Implementation

As discussed in Chapter 4, the sampling algorithm consists of: the visibility cell decomposition that partitions the polygon into visibility cells and computes the visibility segments; and the visibility segment domain sampling that computes the sample viewpoints inside the domains of all the visibility segments.

Visibility cell decomposition

Visibility cell decomposition corresponds to computing the line arrangement of the critical extended windows and the polygon edges. To compute the critical extended windows, we need to compute the reflex vertices of the input polygon. This takes linear time by traversing the polygon counterclockwise and checking if the three consecutive polygon vertices correspond to a *right* turn. (When traversing counterclockwise, the polygon interior is on the *left*

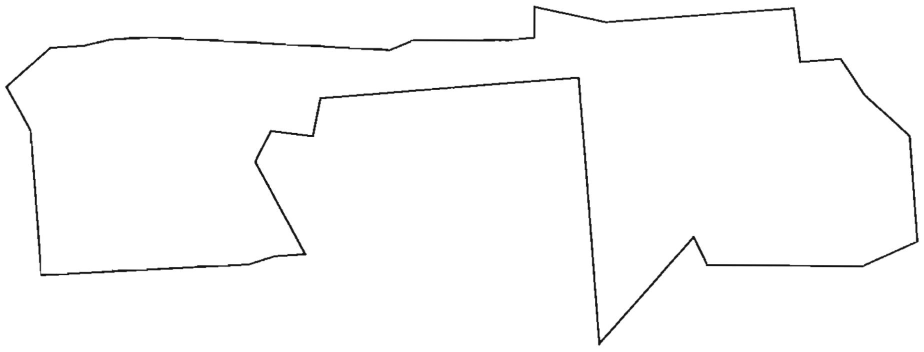


Figure 5.3: The polygon computed from the map in Fig. 5.2.

side.) Or equivalently, for two vectors corresponding to the two polygon edges incident on the vertex, we check the signed magnitude of their cross product. Reflex vertices are those with negative magnitudes. For example, in Fig. 5.4, to check if polygon vertex A_i is a reflex vertex, we compute the cross product $\overrightarrow{A_iA_{i-1}} \times \overrightarrow{A_iA_{i+1}}$ of the two incident vectors, $\overrightarrow{A_iA_{i-1}}$ and $\overrightarrow{A_iA_{i+1}}$. The corresponding formula is as follows.

$$\begin{aligned}\overrightarrow{A_{i-1}A_i} \times \overrightarrow{A_iA_{i+1}} &= \overrightarrow{(x_i - x_{i-1}, y_i - y_{i-1})} \times \overrightarrow{(x_{i+1} - x_i, y_{i+1} - y_i)} \\ &= (x_i - x_{i-1}) \cdot (y_{i+1} \quad y_i) - (x_{i+1} - x_i) \cdot (y_i - y_{i-1})\end{aligned}$$

To compute the critical extended windows, we create a ray starting from a reflex vertex, say A_1, which is collinear with another polygon vertex, say A_2, and in the direction $\overrightarrow{A_2A_1}$. The part of the ray between A_1 and the last point where this ray intersects the polygon is an critical extended window. In a brute force manner, we can compute all the critical extended windows in $O(n^3)$, where n is the number of polygon vertices, since there are $O(n^2)$ rays and it takes $O(n)$ times to find the last intersection point between a ray and the polygon. The number of critical extended windows is $O(n^2)$.

We then use the "incremental algorithm", Chapter 6.3 of [O'R98], to compute the arrangement of the set of line segments that include all the critical extended windows and the polygon edges. The incremental algorithm computes the arrangements incrementally

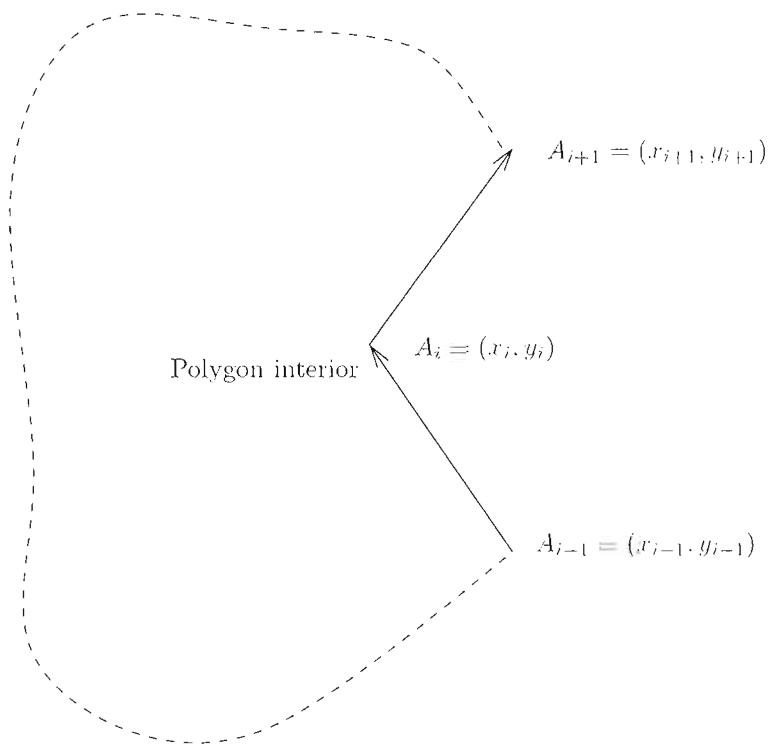

$A_{i+1} = (x_{i+1}, y_{i+1})$
Polygon interior
$A_i = (x_i, y_i)$
$A_{i-1} = (x_{i-1}, y_{i-1})$

Figure 5.4: Check the reflex vertices.

for each line segment, i.e., the segments are put in a list and handled one at a time. For a new line segment w_i, we "walk" along it through all the previously handled segments $w_1, w_2, \dots, w_{i-1}$ to construct the cells in $O(i-1)$ time. The whole process takes $O(k^2)$ time for k line segments and takes $O(n^4)$ for the arrangements of all the critical extended windows, where n is the number of polygon vertices. During the incremental algorithm, we can also easily tell whether a cell belongs to the polygon by the simple observation that to walk from the interior of the polygon and cross the polygon boundary results in a cell that is outside the polygon. Thus, the visibility cell decomposition takes $O(n^4)$ time to compute $O(n^4)$ number of visibility segments.

5.1.3 Visibility segment domain sampling

To compute the sample viewpoints inside a visibility segment domain, we first compute the arrangements of the set of the line segments, including both the boundary edges of the domain and those incident on the visibility vertices inside the domain and vertical to the visibility segment that defines the domain. We then compute the vertices of the arrangements that are inside the domain. In the following, we give details of this process.

To compute the domain boundary edges, we first compute the vertices of the domain. The vertices include the end-points of the visibility segment that defines the domain, and the remaining two "apex" vertices. For the domain corresponding to the visibility segment l, let $X_1 = (x_1, y_1)$ and $X_2 = (x_2, y_2)$ denote the end-points of l, and let $X_3 = (x_3, y_3)$ and $X_4 = (x_4, y_4)$ denote the two apex vertices of the domain. By simple geometry using domain side angle α, X_3 and X_4 are computed as follows. In the following, we use $R^+(\alpha)$ and $R^-(\alpha)$ to denote the 2D rotation matrix (2×2) of rotating counterclockwise and clockwise α angle respectively.

$$X_3 = \overrightarrow{X_1X_3} + X_1 = \frac{1}{2\cos\alpha} R^+(\alpha)\overrightarrow{X_1X_2} + X_1$$

$$X_4 = \overrightarrow{X_1X_4} + X_1 = \frac{1}{2\cos\alpha} R^-(\alpha)\overrightarrow{X_1X_2} + X_1$$

$$\text{i.e.,} \quad \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} = \frac{1}{2\cos\alpha} \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \end{bmatrix} + \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

$$\begin{bmatrix} x_4 \\ y_4 \end{bmatrix} = \frac{1}{2\cos\alpha} \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \end{bmatrix} + \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

Using $\alpha \approx 34^o$ which gives the best approximation ratio, we have $\sin\alpha \approx 0.5601$ and $\cos\alpha \approx 0.284$.

Using the apex vertices and the endpoints of the visibility segment, we can compute the edges of the visibility segment domain. To check whether a visibility vertex is inside a domain takes constant time, by shooting from the vertex a ray and counting the number of intersection points between the ray and the domain boundary edges. If this number is odd, then the vertex is inside the domain; if the number is even (including 0), it is outside the domain.

Computing the sample viewpoints corresponds to computing the intersection points as stated in Section 4.3.2: we draw vertical line segments from each visibility cell vertex, and from each intersection point between domain boundaries and visibility segments. The intersection points between these vertical line segments, other visibility segments, the polygon boundaries, and the domain boundaries are included in the viewpoint sample set. This process is straightforward and we skip the details.

5.1.4 Generating and Solving LP Formulation

LP formulation

To compute the LP formulation, we first need to compute the covering relations between the sample viewpoints and the polygon edges, and then compute the visibility graph of the sample viewpoints. The formulation generation then follows Section 2.5.2. In the following, we give details of the covering relation computation. (The visibility graph construction is straightforward and skipped here.)

Computing the covering relation between a viewpoint X and a polygon edge e corresponds to computing whether e is entirely visible from X. First, we check if both endpoints of e, denoted by $S(e)$ and $T(e)$ respectively, are visible from X, or equivalently, whether the line segments $\overline{XS(e)}$ and $\overline{XT(e)}$ intersect any of the other polygon edges $\mathcal{E} \setminus \{e\}$. ($\mathcal{E}$ is the set of all polygon edges.) This takes linear time. If both $S(e)$ and $T(e)$ are visible from X, the e is not entirely visible from X if and only if one of the polygon holes is contained (entirely) in the triangle $\triangle S(e)T(e)X$. Checking this condition takes at most $O(n)$ time.

Solving Traveling VPP LP

We use the ILOG Cplex [CPL], a commercial LP solver, to solve the LP formulation for the induced Traveling VPP, Eq.(2.8). It takes a plain text file as input and outputs another text file as the solution.

5.1.5 Implementation of Algorithm *Round and Connect*

As input to Round and Connect, the LP optimal solution is given in a file with the extension of ".txt", in which the viewpoint fractional assignments, y_i^*, are given. Also we input the surface patch sets of these viewpoints to the round and connect algorithm.

In the *Round* step, by rounding iteratively the largest y_i^* to 1 until no uncovered surface patch is left, we have the viewpoint solution set $\mathcal{V}'$.

The *Connect* step computes a Steiner tree connection of the viewpoints in $\mathcal{V}'$. We give a 2-approximation solution by using the minimum spanning tree [Vaz01]. The detailed steps are given as follows. First, we construct the visibility graph [dBvKOS00] of the sample viewpoints, $\mathcal{V}'$, and the reflex vertices of the polygon. Second, we construct the shortest path graph, VG, of V', i.e, a complete graph, CG, on V' where the edge cost of viewpoints v_1 and v_2 is the shortest path distance between them on VG. Third, we compute the minimum spanning tree $MST(V')$ over CG. For each edge of $MST(V')$, we compute the corresponding shortest path on VG. Last, we take the union of the edges on these paths. The resulting edge set is the edge solution set E'. To show the result, we draw the line segments corresponding to the edges in E'.

5.2 Simulation Results

In this section, we show the simulation results for some user-generated polygons and two polygon approximations of the layout of the research labs at Simon Fraser University, a smaller one with only the RAMP lab and nearby corridors, Fig. 5.11, and the larger one with the RAMP lab and some other research labs in the 8000 level of Applied Science building at Simon Fraser University, Fig. 5.12. In the simulations, we use a PC with Intel Core2 2.13GHz Processor and 2G RAM for the sampling algorithm and the round and connect algorithm, and a Sun Blade 1000 with two 750 Mhz Sun UltraSPARC III CPU's and 2GB DRAM for the LP optimizer.

The simulation results for different problem instances are shown in Figs. 5.5 to 5.12. They are labeled as Env. 1 to Env. 8 respectively. The number of vertices and reflex vertices respectively of these input polygons and the view and travel weights for these environments are listed in Table 5.1. For the user-generated polygons, we make some arbitrary assumptions on w_p and w_v. For the two maps of real scene, with our experiences on the real robot systems, we assume the cost for traveling 1 meter is equal to taking one viewpoint. This explains why there is a sudden change of the w_p's for the real maps from the user-generated ones. In the figures, we show the input polygon, the sampling viewpoints (denoted by crosses in the figure) computed by the sampling algorithm, and the output of the round and connect algorithm, a Steiner tree connecting the robot start position, denoted by a small circle, and the planned viewpoints, denoted by crosses. We list the results, the number of sampling viewpoints computed, the view cost, travel cost and total cost respectively for the resulting Traveling VPP solution, in Table 5.2. We list the running times for the sampling algorithm, LP optimizer, and the round and connect algorithm in Table 5.3 Note that in the run time reported for the sampling algorithm and LP optimizer, the larger parts (more than 50% and more than 60% respectively) are for I/O (input/output), i.e., to output an LP to a text file and to read in an LP from the text file.

Env	# polygon vertices	# reflex vertices	W_P (per centimeter)	W_v (per viewpoint)
1	14	5	1	10
2	22	10	1	5
3	11	4	1	10
4	17	7	1	10
5	13	7	1	5
6	9	4	1	5
7	55	23	1	100
8	91	32	1	100

Table 5.1: Simulation parameters for different environment

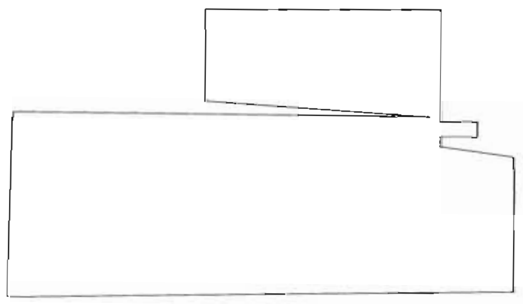
Env	# sampling viewpoints	View cost	Travel cost	Total cost
1	49	40	1.56	41.56
2	219	20	9.33	29.33
3	42	40	2.08	42.08
4	75	40	4.25	44.25
5	269	15	2.5	17.5
6	151	15	3	18
7	425	500	986	1486
8	937	800	2000	2800

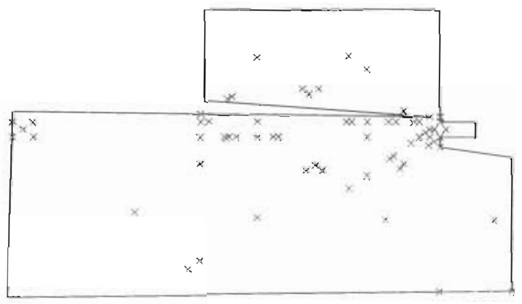
Table 5.2: Simulation result summary

The test results clearly show the power of the algorithms designed. Although we cannot compare it with the optimal solution due to the nature of the problem (NP-hard and log-inapproximable), the results look reasonably close to the optimal solution.

5.3 Discussion

Currently, our implementation can handle polygon with at most 100 vertices and 40 reflex vertices. The reason is that we use a data file that includes the whole LP formulation as the input to the LP optimizer, CPlex. The size limit (set by CPlex) of this LP input determines the most complex polygon for our implementation. However, one can use for example the column generation method described in Appendix C to leverage this problem, since it always maintains a reduced-sized *active* LP and includes variables *as needed*.


(a) The input polygon


(b) The sampling algorithm output

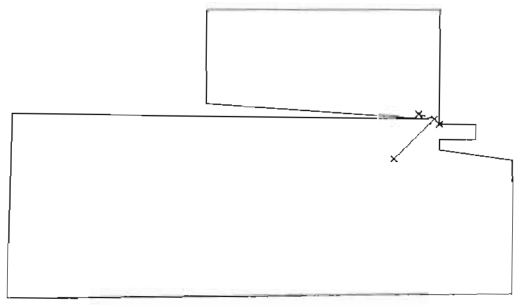
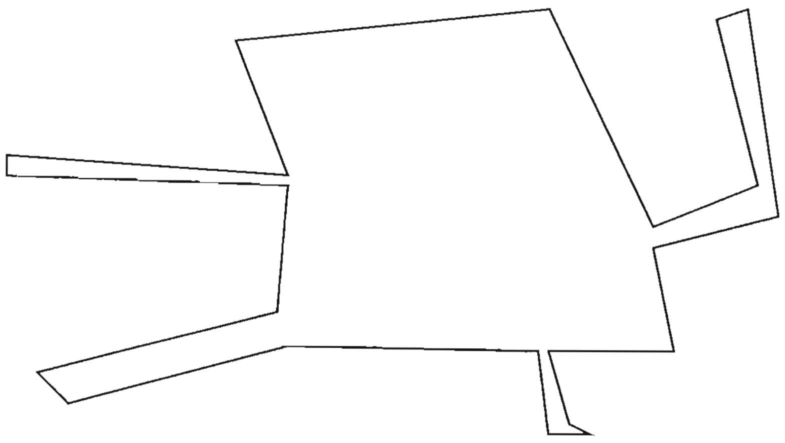
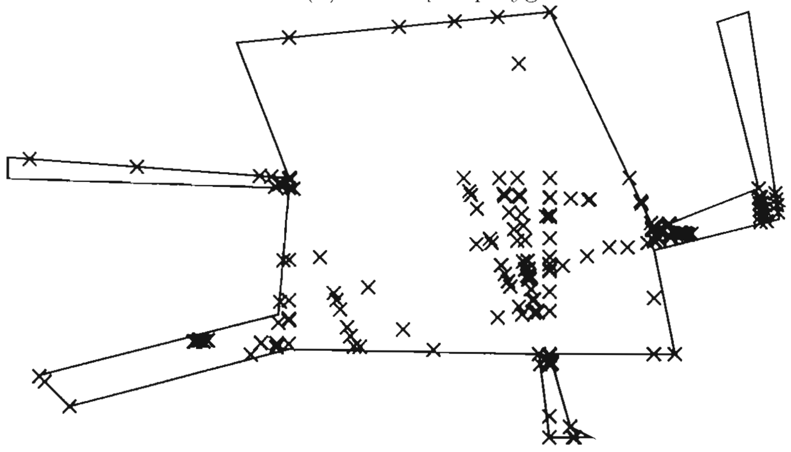

(c) The *Round and Connect* algorithm output

Figure 5.5: The simulation result of Environment 1.


(a) The input polygon


(b) The sampling algorithm output

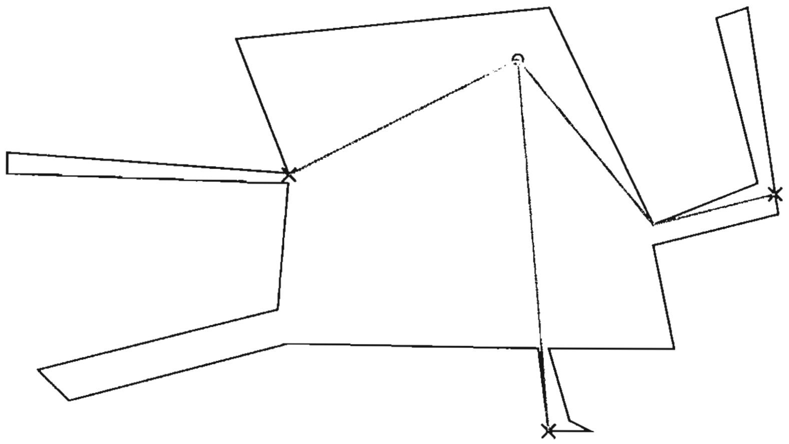
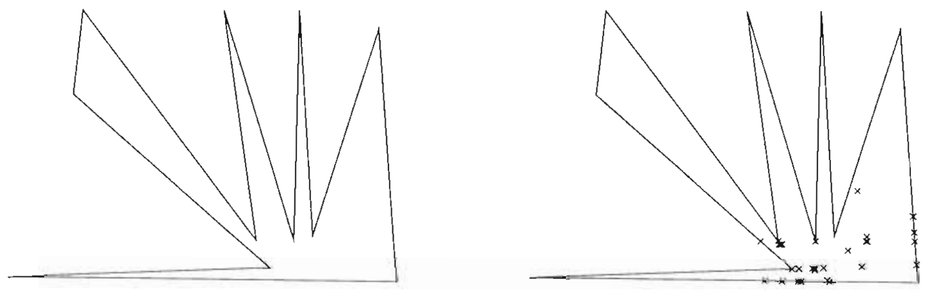
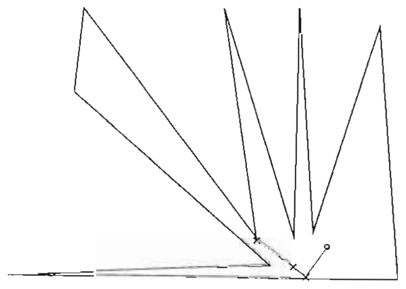

(c) The *Round and Connect* algorithm output

Figure 5.6: The simulation result of Environment 2.

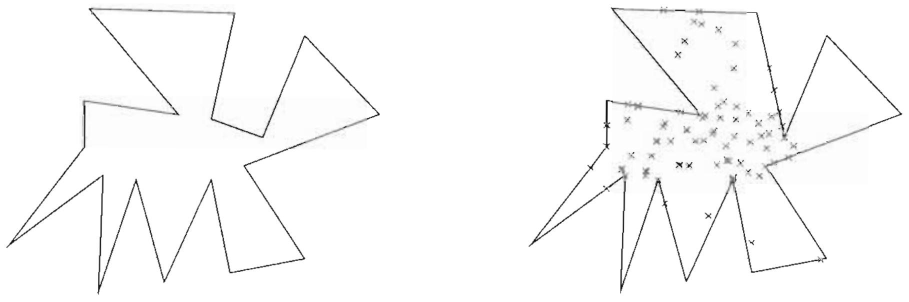


(a) The input polygon (b) The sampling algorithm output

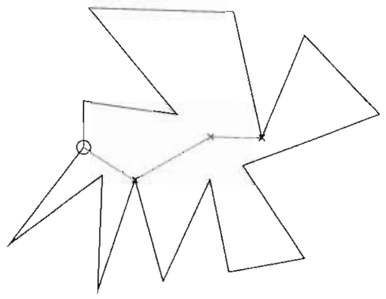


(c) The *Round and Connect* algorithm output

Figure 5.7: The simulation result of Environment 3.

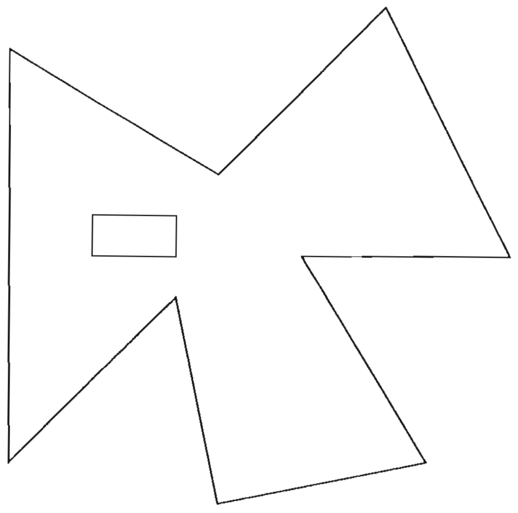


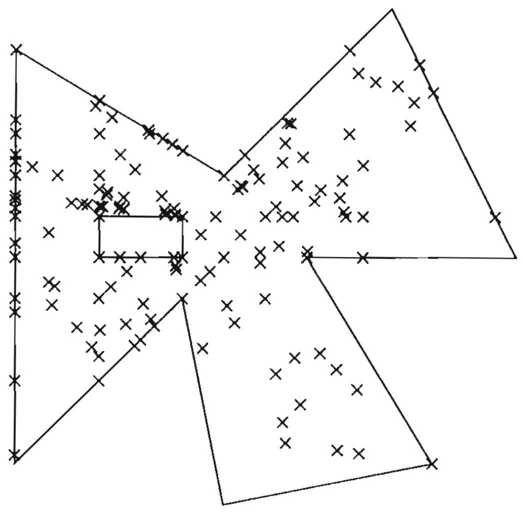
(a) The input polygon (b) The sampling algorithm output



(c) The *Round and Connect* algorithm output

Figure 5.8: The simulation result of Environment 4.


(a) The input polygon


(b) The sampling algorithm output

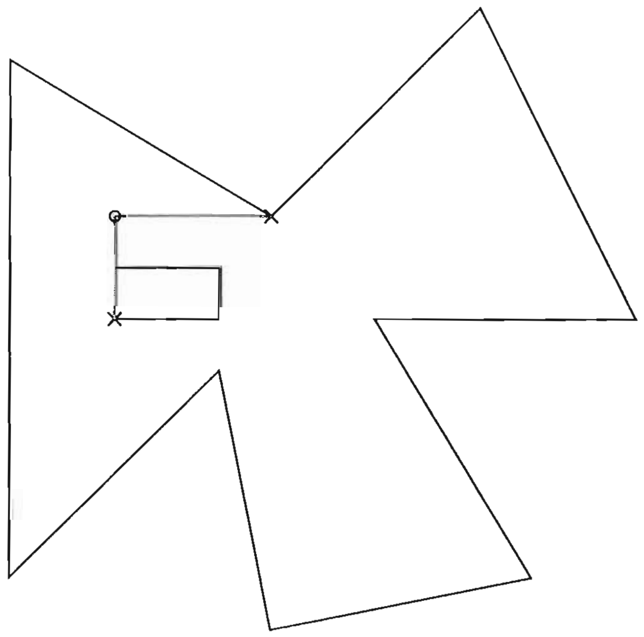
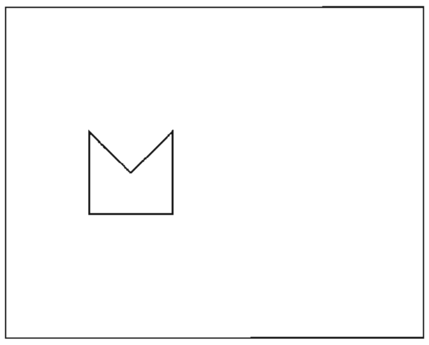
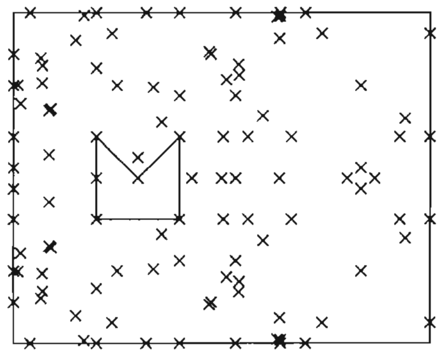

(c) The *Round and Connect* algorithm output

Figure 5.9: The simulation result for Environment 5, a polygon with a hole.


(a) The input polygon


(b) The sampling algorithm output

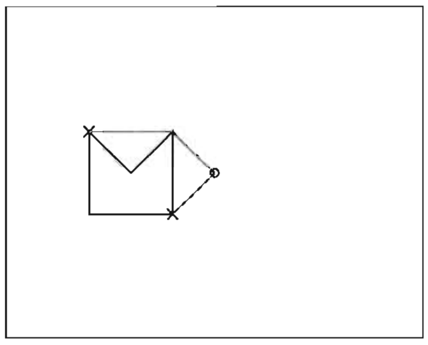
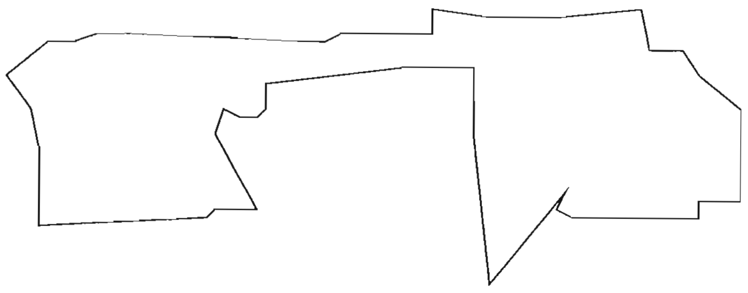
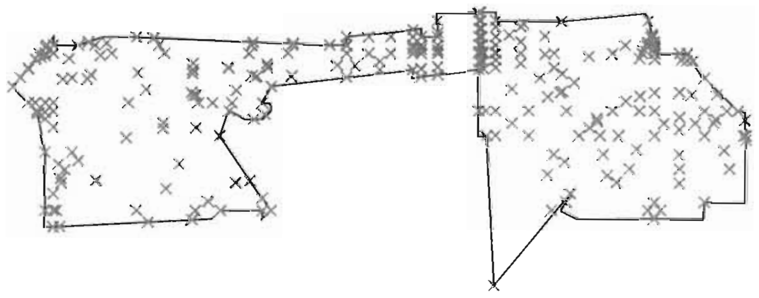

(c) The *Round and Connect* algorithm output

Figure 5.10: The simulation result for Environment 6: a polygon with a hole.


a) The input polygon


(b) The sampling algorithm output

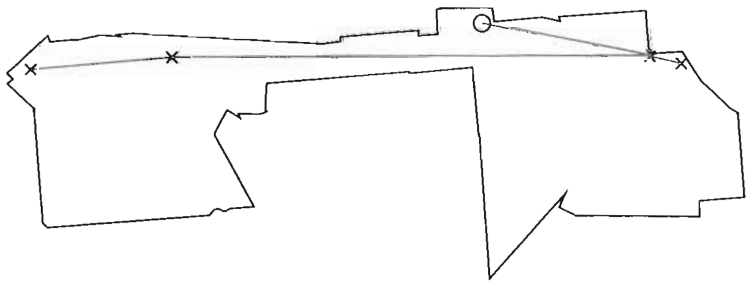
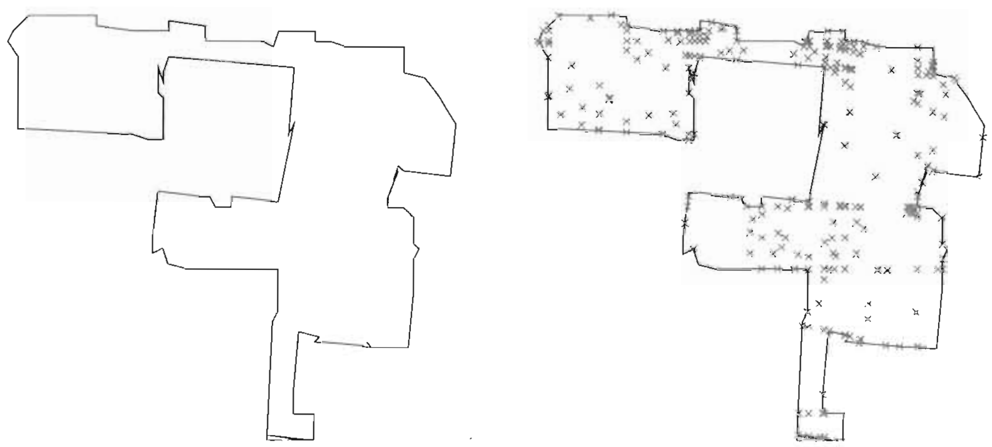

(c) The *Round and Connect* algorithm output

Figure 5.11: The simulation result for the layout of the RAMP lab. (Environment 7.) Scale is 1:200.


a) The input polygon (b) The sampling algorithm output

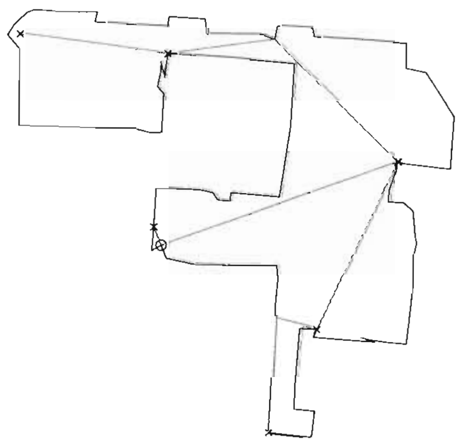

(c) The *Round and Connect* algorithm output

Figure 5.12: The simulation result for the layout of the RAMP and neighboring labs. (Environment 8.) Scale is 1:300.

Env	Sampling algorithm	LP optimizer	Round and connect algorithm
1	18	7	1
2	7	4	2
3	2	1	0.3
4	62	55	1
5	250	118	2
6	134	78	1
7	34	25	3
8	3445	3438	3

Table 5.3: Running time summary (in seconds)

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we considered the problem of planning a sequence of viewpoints and a connecting path by the robot to completely inspect the surfaces of objects of interests. Time and energy consumptions, particularly for remote missions, are critical factors for the successful completion of these automated object inspection tasks. This motivates us to formulate this problem as an optimization problem, View Planning Problem with Combined View and Travel Cost (Traveling VPP), which minimizes the combined total cost of both view and travel. To the best of our knowledge, this thesis is the work in robotics to solve the problem in a global and unified way.

6.1.1 Traveling VPP

First, we considered a discrete version where the viewpoints and a graph that connects them are given as the input. By reduction to the Group Steiner Tree (GST) problem, we showed that Traveling VPP cannot be approximated within the poly-log ratio. We gave an integer linear program (ILP) formulation that minimizes the weighted sum of travel and view costs under the constraints that all the surfaces of the objects are covered by the planned viewpoints and that planned viewpoints are connected. This formulation provides the foundation to solve the problem optimally.

To design an approximation algorithm, we turned to the LP relaxation approach, which address the overall problem in a unified fashion. We gave an LP relaxation based approximation algorithm, Round and Connect, which takes the optimal solution of the LP relaxation of Traveling VPP and outputs an integral solution including the planned viewpoints and the Steiner tree that connects them. We showed that Round and Connect can recover an integral solution whose cost is at most the LP optimal cost times twice the view frequency, $2F$. This implies that Round and Connect has the approximation ratio of $2F$. We also showed, via reduction from Traveling VPP to GST, that the poly-log approximation algorithm for GST is applicable and Traveling VPP can be approximated by a poly-log ratio. Thus, Traveling VPP is approximable within the smaller ratio of the order of view frequency or a poly-log ratio.

That the LP relaxation can potentially have exponential number of constraints motivated us to either give an LP formulation with manageable size, or solve the problem practically by always keeping an active sub-problem with reduced size. We used multi-commodity network flows to achieve the former, and the column generation technique for the latter.

6.1.2 Metric TVPP

We also considered a special case of the discrete problem where a mobile robot is used to inspect objects in 2D or 3D and the sensor has a visibility range. We call this problem "Metric View Planning Problem with Travel Cost and Visibility Range" (Metric TVPP). The insight in the problem is that the visibility range couples the two different objectives, view and travel, and it is possible to use existing solving techniques for the view planning problem. Consequently, we analyzed the two-level approach of Danner and Kavraki [DK02], which first solves the view planning problem without considering the travel cost and then solve the tour problem to connect the planned viewpoints at the first level. Although we showed that this two-level approach performed arbitrarily poorly for the general Traveling VPP, for Metric TVPP we showed that its approximation ratio is in the order of a logarithm of the number of surfaces. This ratio is also in the same order as the inapproximability result of Metric TVPP. An intuitive explanation is that the visibility range ties the view and travel, and the two-level algorithm is no longer *decoupled*.

6.1.3 WEC-GWRP

We then considered a related geometric problem, the Watchman Route Problem with Discrete View Cost (GWRP) where a point robot is required to inspect a 2D polygon (possibly without holes) and viewpoints space is the continuous polygon interior. We tackled a NP-hard and log-inapproximable version of GWRP, in which any polygon edge is required to be entirely visible from at least one planned viewpoint. We call this problem Whole Edge Covering GWRP (WEC-GRWP). We gave a deterministic polynomial sampling algorithm that computes a polynomial number of viewpoints using the geometric structure of the polygon and outputs a Traveling VPP instance using these viewpoints. We showed that the optimal solution of the induced Traveling VPP instance is within a constant (≈ 11.657) ratio of the optimal WEC-GRWP solution. Combined with the approximation algorithm we provide for Traveling VPP, WEC-GWRP admits an approximation algorithm.

The special advantage of the sampling algorithm is that the number of sampled viewpoints computed is a polynomial of the number of viewpoints and does not depend on any geometric parameter of the polygon. This contrasts to the algorithms by Papadimitriou [Pap85] and Aleksandrov *et al.* [AMS05], which the number of viewpoints needed also depend on the size of the input polygon and the regularity of the polygon triangulation. We believe that the sampling algorithm is a general technique and can be applied to other problems, for example, the problem Terrain Guarding with Triangle Restriction covered in Chapter 4.

6.1.4 Experiments

We then conducted some preliminary experiments/simulations of the approximation algorithms for some 2D environments, including simulated 2D environments and real robot-generated 2D maps of the lab areas at RAMP lab, Simon Fraser University. For the real maps, we first compute a polygon approximation and then apply our approximation algorithms. We display the computed sampled viewpoints and the traveling path of the robot. This is an important step towards implementation on real robot systems.

6.2 Future Work: Traveling VPP in Unknown Environment

It is interesting to extend our Traveling VPP to the unknown case where the viewpoints, the traveling graph, and the surface patches of the objects are not known in advance. The algorithm hence has to decide, based on current information, whether to explore the possible viewpoints, called graph exploration, or to view the "bounding boxes" of the objects to get more information about them.

In the following, we give the formulation of Unknown Traveling VPP after stating some assumptions of the problem. To clarify the problem, we compare the formulation with the *online problems* defined in the optimization literature [BEY98]. In Appendix C, we discuss a simple case, called *Traveling VPP on Explored Graph*, where the surface patch set is given and the traveling graph is unknown, and give a competitive algorithm that combines a competitive online graph search algorithm and our Traveling VPP approximation algorithm. For future work, we would like to investigate the case where both the surface patch set and the graph are not known in advance.

6.2.1 Problem assumptions

To clarify, we make the following assumptions.

- Inspection and Exploration Sensors

  We have two types of sensors onboard the robot, one used for object inspection and thus called inspection sensor, denoted by IS, and the other used for robot traveling graph exploration thus called exploration sensor, denoted by ES. The IS is "expensive" and the cost associated with each IS view is counted in our objective function to minimize; the ES, on the other hand, is considered "cheap" and the cost of each ES scan is ignored. We define the process before the next $IS(ES)$ scan an $IS(ES)$ iteration.

  In real robotic applications, the ES corresponds to a planar range sensor (a laser beam scans in a plane) used for simultaneous localization and mapping (SLAM) [TBF05], which can, in real time, process 2D sensory input (range data in a plane obtained via a single scan of a laser beam) to build a map representation of the environment W, and estimate the robot location w.r.t. the map built; the IS corresponds to an area scan laser range sensor (a laser beam scans an area to provide a 3D range image) that can build a surface representation of the visible portion of the scanned object.

- Surface Coverage Completion Criterion

  We assume that a surface coverage completion criterion denoted by O is available, i.e., we know at any time whether the task is completed or not after calling O. (The existence of such a task completion criterion is critical to our problem.)

  For our inspection tasks, such a completion criterion O can be implemented as, for example, to compute whether all the missing parts in the constructed object surface model are "filled". In the following, we show how to implement such a completion criterion O using the online surface patch set generation model.

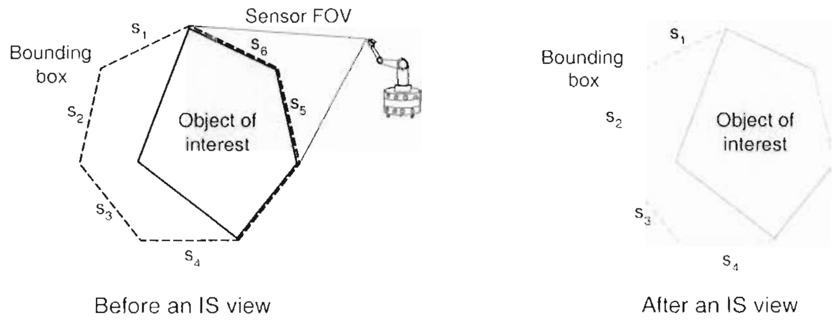
- Online Surface Patch Set

  Intuitively, the online surface patch set is the set of the surfaces of the bounding box of the object that the robot needs to take scans for complete surface construction based on current knowledge. The online surface patch set is generated and modified after each IS scan action, denoted by A_{IS}. We also require the set to be covered before the next iteration.

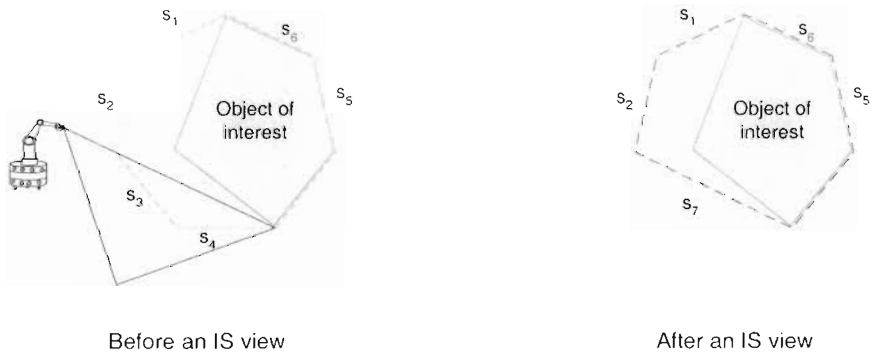
  The online surface patch set is generated in the following fashion. We are initially given a set of "bounding boxes" and the objects of interest are contained in respective boxes. The unscanned surface patch set of these bounding boxes become our initial online surface patch set to cover. After each inspection scan/view is taken, the bounding boxes are shrunk by newly scanned object surface or constructed edges or boundary of the sensor field of view (FOV) at views taken. The unscanned surface patches of the modified bounding boxes become the new online surface patches to cover.[1]

  We call the unscanned surface patches of the bounding boxes at IS iteration k the online surface patch set at iteration k, denoted by $\mathcal{S}^k$. Fig. 6.1 shows the required surface patch sets, $\mathcal{S}^k$ and $\mathcal{S}^{k+1}$ respectively, before and after an IS view.

[1]Note that due to the fact that the new surface patch set after a sensor scan is a proper subset of the union of the former surface patch set and those corresponding to the constructed surfaces and sensor FOV boundaries, the interior of the bounding boxes are always shrinking.



(i) S^{k+1} is a proper subset of S^k



(ii) General case

Figure 6.1: The online surface patch set before and after a IS view.

Due to the geometric nature of the problem, the change from $\mathcal{S}^k$ to $\mathcal{S}^{k+1}$ cannot be completely arbitrary. For example, Fig. 6.1(i) shows that $\mathcal{S}^{k+1} = \{s_1, s_2, s_3, s_4\}$ is a proper subset of $\mathcal{S}^k = \{s_1, s_2, s_3, s_4, s_5, s_6\}$; Fig. 6.1(ii) shows that $\mathcal{S}^{k+1} = \{s_1, s_2, s_5, s_6, s_7\}$ include some (but not all) elements of $\mathcal{S}^k$ and a constructed surface s_7. Note that since the bounding boxes always shrink, $\mathcal{S}^k$ cannot be a proper subset of $\mathcal{S}^{k+1}$. Also, assuming omnidirectional line of sight of the IS, the number of reflex vertices on the object surface bound the number of constructed surface boundaries. This is especially useful when analyzing the algorithmic performance and factoring the intrinsic problem complexity in the analysis.

Using the online surface patch set model, the completion criterion O can be implemented as to check whether the current online surface patch set $\mathcal{S}^k$ is empty or not.

- Online Robot Traveling Graph Exploration

  In exploring the robot traveling graph, the robot uses the ES to update the graph it keeps for further robot traveling path planning and exploration purposes. We assume that (with an appropriate SLAM mechanism implemented) the robot explores the environment, or equivalently, with the SLAM mechanism implemented, the robot has a fairly accurate map explored and can locate itself precisely.

  The graph at an ES iteration k, denoted by $G^k = (\mathcal{V}^k, E^k)$, includes the *explored* viewpoints in $\mathcal{V}^k$, the *explored* edges in E^k. An edge e is *explored* if the volume swept by the robot if it were to move along e is a subset of the free environment $\mathcal{W}_{free}$; a viewpoint v is *explored* if it is connected to the robot current position s^k via explored edges, i.e., $\exists p \in \mathcal{PATH}(E^k) : s^k = start(p) \wedge v = end(p)$, and the subset of online surface patches that v covers is known. In addition, we maintain, for each explored viewpoint $v \in \mathcal{V}^k$, an *excursion edge set* $EE^k(v)$, the set of *unexplored* edges incident on v at iteration k.

  A graph exploration iteration k starts by the robot planning an exploration action, denoted by A_{ES}, including a known/explored viewpoint v of the current graph G^k, i.e., $v \in \mathcal{V}^k$, and an edge e_u in its excursion edge set $EE^k(v)$, i.e., $e_u \in EE^k(v)$. The robot then travels from its current position s^k to v and then starts traversing edge e_u. If an edge e_u is explored, the viewpoint v' that e leads to is added to the graph and its excursion edge set is computed; if e_u is *un-explorable*, i.e., the edge is blocked by obstacles in the environment, it is eliminated from $EE^k(v)$, and we add to the graph the free part $e'_u, e'_u \subset e_u$ and the viewpoint e'_u leads to. The exploration cost include the path distance from s^k to v and the edge cost, either c_{e_u} or $c_{e'_u}$.

6.2.2 Problem definition

The unknown Traveling VPP is defined as follows. In the following, let k denote the iteration of an unknown Traveling VPP algorithm. (It can be either an IS iteration or an ES iteration.) Let K be the last iteration before completion. In the following, we use $\|p\|$ to denote the length of a path p.

Unknown Traveling VPP

"Given a surface covering completion criterion O, a partially known and iteratively changing surface patch set S^k, a viewpoint set $\mathcal{V}^k, k = 0, 1, \ldots$ with known covering relation (w.r.t. S^k), and a partially explored graph $G^k = (\mathcal{V}^k, E^k), k = 0, 1, \ldots$, plan an optimal online strategy that, if the task is incomplete according to O, decides the next action A^k of the robot, $A^k \in \{A_{IS}, A_{ES}\}$.

- *$A_{ES} = (v, e_u), v \in \mathcal{V}^k, e_u \in EE^k(v)$. The corresponding cost is $w_p(\|p(s^k, v)\| + c_{e_u})$, if e_u is explorable; or $w_p(\|p(s^k, v)\| + c_{e'_u})$, if e_u is un-explorable.*
- *$A_{IS} = v, v \in \mathcal{V}^k$. The corresponding cost is $w_v + w_p\|p(s^k, v)\|$.*

A strategy is called "optimal" if the total cost until task completion by the online algorithm is minimized. The total cost, denoted by cost(A), includes both the exploration and traveling cost, the total distance the robot traveled, and the view cost, proportional to the total number of IS scans, i.e.,

$$\begin{aligned} cost(A) &= \sum_{k=1}^{K} (w_v + w_p\|path(s^k, v)\|)(A == A_{IS}) \\ &\quad + w_p(\|path(s^k, v)\| + c_{e_u(e'_u)})(A == A_{ES}), \end{aligned}$$

where $(A == A_{IS})$ and $(A == A_{ES}$ denote the binary decision variables whether the next action is to scan or not and whether to explore or not respectively."

Please note the difference between the model based off-line and the online cases. Unlike the model-based case where a deterministic sequence of traveling path and viewpoints is asked, the online case asks for an algorithm that decides the "best" action according the current knowledge.

6.2.3 Unknown vs. online

To further clarify the unknown Traveling VPP formulation, here we distinguish it from the online problem defined in the optimization literature [BEY98].

In an *online optimization problem*, at each iteration a part of the problem input is generated and has to be solved before the input from the next iteration comes. To tell the quality of an *online algorithm*, the cumulative solution cost until the online input stops coming is often compared with the *offline* optimal solution cost, defined as the optimal solution cost if all the online-generated inputs are known in advance. The quotient between the online and offline optimal solution costs is called the *competitive ratio*.

To distinguish the online problem and our unknown Traveling VPP formulation, let us formulate a third problem, the *model-based online Traveling VPP*, and compare it with the unknown Traveling VPP. For model-based online Traveling VPP, there is no notion of bounding box and all the surface patches given in each iteration are in the input set of the corresponding offline problem.

<u>Model-based Online Traveling VPP</u>

"*For a given viewpoint set $\mathcal{V}$ connected via a known robot traveling graph $G = (\mathcal{V}, E)$, at each iteration k, a surface patch subset S^k is given for the robot to scan. We further require $S^k \cap S^{k'} = \emptyset, k' < k$, since once a surface patch is scanned it is no longer required to be scanned again. The robot has to plan a set of viewpoints $\mathcal{V}^k$ and the connecting graph edge set E^k such that all the elements in S^k are covered. The total cost until completion is the cumulative total view and traveling cost, i.e., $\sum_{k=1}^{K} w_v|\mathcal{V}^k| + w_p \sum_{e\in E^k} c_e$.*"

In the following, we list some key differences between the unknown Traveling VPP and model-based online Traveling VPP. Note that for the model-based online Traveling VPP the traveling graph is known.

- Online surface coverage:

  By definition, the model-based online Traveling VPP requires the surface patches generated at each iteration are covered before the next set of patches are given, while the unknown Traveling VPP does not.

- Surface patch set generation:

For the model-based online Traveling VPP, the surface patch set generated at each iteration does not have any intersection with previous generated sets, while for the unknown Traveling VPP, the sets can have somewhat arbitrary relation as discussed earlier while defining the online patch surface path set for the unknown case.

Also, for the model-based online Traveling VPP, the online surface patches at each iteration are also included in the offline problem, i.e., the offline optimal solution still has to cover them. For the unknown Traveling VPP, different sensing outcomes resulting from different viewpoints planned at each iteration lead to different online surface patch set in the next iteration. For example, see Fig. 6.1 (i) and (ii). Thus, the union of these online surface patch sets of all iterations may change as a result of the course of the online actions.

Appendix A

Background: Inapproximability and L-reduction

Although different NP-complete optimization problems are considered at the same complexity level in terms of solving for the optimal solutions, they can have different difficulty levels in obtaining approximation solutions. This concept is quantified by the *inapproximability* or the *hardness of approximation*, which refers to a problem specific constant or a function of the input size as the lower bound on the approximation ratio of any polynomial algorithm, assuming some generally-believed complexity class relations, for example $P \neq NP$. For example, the inapproximability result for SCP says that the optimal solution to SCP cannot be approximated within its $(1 - o(1)) \ln n$ ratio unless NP admits quasi-polynomial algorithms [Fei98].

We use the simplest form of a common technique, called *L-reduction*, to establish the inapproximability result for Metric TVPP, which works similarly as the reduction method [GJ79] used in proving NP-completeness results. Rather than covering the theory, we refer to [KV00] for a detailed coverage on L-reduction. An intuitive explanation of the form we use is as follows. Suppose the problem of interest is P_1 and we know the inapproximability result for problem P_2. If, given an arbitrary instance of P_2, we can construct in polynomial time an instance of P_1 such that the optimal solution costs of the two instances are the same, and for any solution to the P_2 instance, we can construct in polynomial time a solution to the P_1 instance with the same cost, we can claim that the inapproximability result of P_2 extends to P_1. Otherwise, if there exists an algorithm to P_1 with a better approximation ratio, we can recover a better approximation solution to any P_2 instance by first solving the constructed P_1 instance using this better algorithm. This is contradictory to the inapproximability result we know for P_1. Note that if the inapproximability result depends on the input size, for example that for SCP is $\log n$ where n is the number of elements in the universe, we will have to introduce the size difference between the constructed P_1 instance and the P_2 instance to P_1's inapproximability result. However, this is not the case for our reduction given in Section 3.2 where the given SCP instance and the constructed Metric TVPP instance have the same number of surface patches.

Appendix B

Column Generation Technique Applied to Traveling VPP

Column generation method is a practical way to solve a LP formulation, and it is very close to the simplex method [DDS05]. It always works with a subproblem of the LP, with only a reduced number of variables. It adds variables when they can improve the object function; and drops those nonprofitable variables. The criteria that tell whether variables are profitable are called the *column generation rules*.

For Traveling VPP, since the primal program has a large number of constraints, it is more convenient to work with the dual program using the column generation method. In the following, we give the dual program for Traveling VPP, derive the column generation rules for solving the dual program, and use the complementary slackness conditions (CSCs) to get the solution.

B.1 Dual program and complementary slackness conditions for Traveling VPP

Corresponding to the constraints (2.4), we associate the dual variables, α_j, to each surface patch j; and corresponding to the constraints (2.5), we associate the dual variables, β_{iT}, to each pair of view node i and cut T that includes i but excludes s. The corresponding dual linear program (DLP) is given as:

DLP: $\max \sum_{j \in \mathcal{S}} \alpha_j$

Subject to:

$$\forall i \in \mathcal{V}: \quad \sum_{j \in \mathcal{S}(i)} \alpha_j - \sum_{T: i \in T} \beta_{iT} \leq w_v \tag{B.1}$$

$$\forall e \in E: \quad \sum_{i \in \mathcal{V}} \sum_{T: i \in T \wedge e \in \delta(T)} \beta_{iT} \leq w_p \, c_e \tag{B.2}$$

$$\alpha_j, \beta_{iT} \geq 0$$

As per the duality theorem, a feasible solution to the primal linear program (its dual) provides bounds on the optimal solution to the dual (primal) and achieve the same optimal value when the complementary slackness conditions (CSCs) are satisfied.

CSC of Traveling VPP:

Primal CSC:

$$y_i > 0 \Rightarrow \sum_{j \in \mathcal{S}(i)} \alpha_j = \sum_{T: i \in T} \beta_{iT} + w_v$$

$$z_e > 0 \Rightarrow \sum_{i \in \mathcal{V}} \sum_{T: i \in T \wedge e \in \delta(T)} \beta_{iT} = w_p c_e$$

Dual CSC:

$$\alpha_j > 0 \Rightarrow \sum_{i \in \mathcal{V}(j)} y_i = 1$$

$$\beta_{iT} > 0 \Rightarrow \sum_{e \in \delta(T)} z_e = y_i$$

In the column generation paradigm, we are working with the DLP, which contains a large number of variables, α_j associated with surface patches and β_{iT} associated with view point i and a cut T of view set $\mathcal{V}$ that contains i. The idea is simple and related to the Simplex method: we always work with a reduced version of the LP by only considering a small number of dual variables, corresponding to the basic variables in the Simplex algorithm for LP, and add columns/non-zero dual variables whenever the dual solution can be improved. From the primal program point of view, after solving the LP relaxation using only the basic varialbes, we can use the CSCs to get the corresponding primal variable solution; we check the infeasibility of the primal constraints and for infeasible constraint, add the corresponding dual variable.

B.2 Column generation algorithm

The infeasibility of the constraint (2.4) for a surface patch j, or the rule to add α_j, says that we should add α_j to the reduced LP if the sum (over its viewpoint set) of the (fractional) viewpoint assignments is less than 1. We can simply add the fractional assignments of the views that can see surface patch j and compare it with 1. So assuming we have the optimal primal solution to the reduced LP, this takes $O(|\mathcal{V}||\mathcal{S}|)$ time, where $|\mathcal{V}|$ and $|\mathcal{S}|$ are the numbers of surface patches and views respectively. (By some additional data structures, for example the links between surface patch and the views that see it, we can reduce the checking time.)

The rule for adding β_{iT} for a specific viewpoint i asks to check among all the cuts whether there exists one cut T such that the number of edge crossing T is less than the viewpoint assignment y_i. In the following, we show this corresponds to the classic min-cut problem.

Lemma 15. *The column generation rule for adding the dual variable β_{iT} for a viewpoint and cut pair, for a specific viewpoint i is equivalent to requiring the minimum value of such summation for all possible cuts T to be strictly less than the viewpoint assignment, i.e.,*

$$\min_{T \subset \mathcal{V}: i \in T \wedge s \notin T} \sum_{e \in \delta(T)} z_e < y_i \tag{B.3}$$

Proof. First, if $\min_{T \subset \mathcal{V}: i \in T \wedge s \notin T} \sum_{e \in \delta(T)} z_e < y_i$ and supposing T' is such a minimum cut, i.e.,

$$\beta_{iT'} < y_i, \quad T' = \arg \min_{T \subset \mathcal{V}: i \in T \wedge s \notin T} \sum_{e \in \delta(T)} z_e,$$

the primal constraint corresponding to the dual variable $\beta_{iT'}$ is not feasible and $\beta_{iT'}$ should be added.

Second, for any dual variable β_{iT} if the primal constraint is infeasible, i.e., $\sum_{e \in \delta(T)} z_e < y_i$, by fixing viewpoint i, we have

$$\min_{T' \subset \mathcal{V}: i \in T' \wedge s \notin T'} \sum_{e \in \delta(T')} z_e \leq \sum_{e \in \delta(T)} z_e < y_i$$

□

By recalling the definition of the min-cut problem as finding the $s - t$ cut (i.e., the cut or partition of the graph vertices separating the vertices s and t on the graph) with the minimum cut capacity (defined as the sum of the capacities of the edges crossing the cut), the subproblem of adding β_{iT} for a specific i is equivalent to the min-cut problem.

Lemma 16. *The subproblem of adding β_{iT} for a specific i is equivalent to the min-cut problem by assigning the edge assignments z_e as the capacities for the edges.*

Proof. The proof proceeds by constructing the min-cut problem from the primal solution z_e. By assigning the edge capacities c_e as the edge assignments z_e for all the edges, we have a min-cut problem instance,

$$\underline{\textbf{Min-Cut Problem}} \quad \text{Minimize} \sum_{e \in \delta(T)} z_e, \quad \forall T \subset \mathcal{V} : i \in T \wedge s \notin T$$

The solution of the above min-cut problem is clearly $\min_{T \subset \mathcal{V}: i \in T \wedge s \notin T} \sum_{e \in \delta(T)} z_e$. So according to the subproblem formulation of adding dual variable β_{iT} for a specific i, (B.3), we can decide whether to add such dual variable βiT by comparing the optimal solution with y_i. □

By identifying the second type of subproblem, adding β_{iT}, to be the min-cut problem, we can apply efficient algorithms for solving min-cut problems. Specifically, according to the duality between min-cut and max-flow problems [PS82], the existing efficient max-flow algorithms can be readily applied.

Appendix C

Traveling VPP on Explored Graph

In the following, we consider the case where the robot traveling graph is not known in advance but the surface patch set is given. So the robot has to first explore the graph efficiently to expand its known viewpoint set before deploying a plan using the known viewpoints and on the known traveling graph. We call this problem *Traveling VPP on Explored Graph*. It is not difficult to see the competitive ratio for the online graph searching algorithm is a lower bound on this online version, since the adversary[1] can hide one viewpoint at where the online graph search algorithm fails to explore to see all the surface patches. In the following, we show how to combine a competitive online graph search algorithm and a online Traveling VPP solver to get a offline Traveling VPP algorithm with double the competitive ratio for the graph search.

The online graph search problem is closely to Traveling VPP on Explored Graph. It refers to searching an unknown graph for a goal with unknown position on the graph [FKK+04]. In the next section, we show the competitive ratio for this problem is a lower bound on the competitive ratio for the online Traveling VPP, even in the case where the surface set is given in advance (thus the only unknown part is the graph for traveling). In [FKK+04], the authors give an online algorithm using the spiral search heuristics in the breadth first fashion that achieves the same performance (up to a constant factor) as the best offline search algorithm.

Intuitively, the approach to Traveling VPP on Explored Graph is to decide when to stop

[1]The notation of *adversary* is often used in the online algorithm analysis. It is modeled as the adversary that provides the online input to the problem such that the performance of the online algorithm is the worst.

exploring the graph and start realizing a Traveling VPP solution using the explored graph. We want the graph to be sufficiently explored in the sense that high quality viewpoints are not missed; and at the same time, it is not necessary to exert too much exploration effort.

In the following, we give an online algorithm that answer the above question. We use the competitive ratio analysis to show that the performance of the algorithm proposed is bounded when compared with the optimal offline algorithm when the graph is given.

We use G to denote the (partially) unknown graph and use G^k, superscripted by the iteration label, to denote the known part of the graph after the exploration iteration k.

C.1 Online Algorithm

C.1.1 Online graph search algorithm

We first assume that we have available a competitive online graph search algorithm $\mathcal{A}$. It is competitive in the sense that for any view point v, the shortest path length from the robot start position s to v is within a factor f of the total traveling length for the robot to search v by algorithm $\mathcal{A}$. By denoting the shortest path length between s and v by $\mathsf{sp}(s,v)$ and total robot traveling length to search v by $\mathcal{A}(v)$, we have the online algorithm guarantees

$$\mathcal{A}(v) \leq f \cdot \mathsf{sp}(s,v) \tag{C.1}$$

For example in case the graph is two rays emitting from s, the online graph search problem becomes the "lost cow problem" and the iterative doubling algorithm has the competitive ratio, f, of 9.

C.1.2 Online algorithm for Traveling VPP

With a competitive online graph search algorithm $\mathcal{A}$, we start by using $\mathcal{A}$ to search the graph iteratively. Each iteration ends when some new viewpoint(s) is(are) explored. At the end of an iteration, labeled by k, we use the explored graph G^k to solve the Traveling VPP. Assuming the corresponding Traveling VPP can be solved optimally, we compare the corresponding cost, OPT_{G^k} with the (weighted) exploration cost so far. If the exploration cost is equal to the Traveling VPP cost, we stop the exploration and solve the Traveling VPP using the explored graph so far.

C.1.3 Analysis

According to the algorithm, the exploration stops at iteration k when the (weighted) cumulative exploration cost is equal to the optimal Traveling VPP solution cost using the explored graph, i.e.,

$$w_p \mathcal{A}^k = OPT_{G^k}^{TVPP} \tag{C.2}$$

In the following, we show the Traveling VPP solution cost using these explored viewpoints, $OPT_{G^k}^{TVPP}$ is bounded within at a certain ratio of the offline optimal solution cost, OPT_G^{TVPP}.

Assume the optimal offline algorithm uses viewpoints $v_1, v_2, \ldots, v_n$ and edges $e_1, e_2, \ldots, e_m$ to solve the Traveling VPP and the corresponding cost is given as:

$$OPT_G^{TVPP} = w_P \cdot OPT_{(s,v_1,v_2,\ldots,v_n)}^{Tree} + w_v \cdot n,$$

where $OPT_{(s,v_1,v_2,\ldots,v_n)}^{Tree}$ denotes the optimal Steinter tree that connects the viewpoints $v_1, v_2, \ldots, v_n$ and s.

It is obvious that the shortest path distance from s to any viewpoint in the tree is a lower bound on the tree cost, i.e.,

$$OPT_{(s,v_1,v_2,\ldots,v_n)}^{Tree} \geq \mathsf{sp}(s, v_j), \quad 1 \leq j \leq n$$

Consequently, this shortest path distance is also a lower bound on the optimal offline solution, i.e.,

$$OPT_G^{TVPP} \geq w_p \cdot \mathsf{sp}(s, v_j), \quad 1 \leq j \leq n \tag{C.3}$$

In the following, we consider two cases at iteration k.

First, if the optimal solution, namely viewpoints $s, v_1, v_2, \ldots, v_n$ and edges $e_1, e_2, \ldots, e_m$, are known when we stop the exploration, i.e., $s, v_1, v_2, \ldots, v_n \in G^k$, clearly $OPT_{G^k}^{TVPP} = OPT_G^{TVPP}$. And the total online algorithmic cost is $\mathcal{A}^k + OPT_{G^k}^{TVPP} = 2OPT_G^{TVPP}$. So the competitive ration is 2.

Second, if G^k is a proper subset of G, i.e., $\exists v_j \in G : v_j \notin G^k$. (In case $\exists e_i \in G : e_i \notin G^k$, we can simply (imaginarily) put a viewpoint at the midpoint of e_i and this reduces to $\exists v_j \in G : v_j \notin G^k$.) Since our graph exploration algorithm is competitive, we must have

$$f \cdot \mathsf{sp}(s, v_j) \geq \mathcal{A}^k. \tag{C.4}$$

By Eq. (C.3), we have the total online algorithm cost:

$$\begin{aligned}
cost^{\text{online algorithm}} &= w_p\mathcal{A}^k + OPT_{G^k}^{TVPP} \\
&= 2w_p\mathcal{A}^k \quad \text{(By algorithm stopping criterion)} \\
&\leq 2f \cdot w_p\mathsf{sp}(s, v_j) \quad \text{(By competitive ratio of } \mathcal{A}\text{, Eq. (C.4))} \\
&\leq 2f \cdot OPT_G^{TVPP} \quad \text{(By bound on the Traveling VPP cost, Eq. (C.3))}
\end{aligned}$$

In summary, the competitive ratio of our online algorithm is $2f$.

Bibliography

[AMS05] L. Aleksandrov, A. Maheshwari, and J.-R. Sack. Determining approximate shortest paths on weighted polyhedral surfaces. *Journal of the ACM*, 52(1):25–53, January 2005.

[BA06] P. Blaer and P. Allen. View planning for automated site modeling. In *Proc. IEEE International Conference on Robotics and Automation*, pages 2621–2626, Orlando, USA, May 2006.

[BATM95] P. Bessiere, J. Ahuactzin, E. Talbi, and E. Mazer. The ariadne's clew algorithm: Global planning with local methods. In K. Goldber, D. Halperin, J. Latombe, and R. Wilson, editors, *Algorithmic Foundation of Robbotics*, pages 39–47. A K Peters, Ltd., 1995.

[BD90] K. Bowyer and C. Dyer. Aspect graphs: an introduction and survey of recent results. *International Journal of Imaging Systems and Technology*, 2:315–328, 1990.

[BEY98] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[BLM92] P. Bose, A. Lubiw, and J. Munro. Efficient visibility queries in simple polygons. In *Proc. Fourth Canadian Conference on Computational Geometry*, pages 23–28, 1992.

[BMKM05] B. Ben-Moshe, M. Katz, and J. Mitchell. A constant-factor approximation algorithm for optimal terrain guarding. In *Proc. 16th ACM-SIAM Sympos. on Discrete Algorithms*, pages 515–524, 2005.

[CEK06] C. Chekuri, G. Even, and G. Kortsarz. A greedy approximation algorithm for the group steiner problem. *Discrete Applied Mathematics*, 154(1):15–34, 2006.

[Chr76] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report 338, Grad School of Industrial Administration, Carnegie Mellon University, 1976.

[CN91] W. Chin and S. Ntafos. Watchman routes in simple polygons. *Discrete and Computational Geometry*, 6(1):9–31, 1991.

[CPL] Ilog cplex. http://www.ilog.com/products/cplex. Last accessed: Sep. 26, 2007.

[dBvKOS00] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2nd edition, 2000.

[DDS05] G. Desaulniers, J. Desrosiers, and M. Solomon. *Column Generation*. Springer, 2005.

[DK02] T. Danner and L. Kavraki. Randomized planning for short inspection paths. In *Proc. IEEE International Conference on Robotics and Automation*, pages 971 – 976, 2002.

[EHP02] A. Efrat and S. Har-Peled. Guarding galleries and terrains. In *Proc. the IFIP Seventeenth World Computer Congress - TC1 Stream / 2nd IFIP International Conference on Theoretical Computer Science*, pages 181–192, 2002.

[Eid02] S. Eidenbenz. Approximation algorithms for terrain guarding. *Information Processing Letters*, 82:99–105, 2002.

[EP04] A. Eliazar and R. Parr. Dp-slam 2.0. In *Proc. 2004 IEEE International Conference on Robotics and Automation*, pages 1314– 1320, 2004.

[ESS93] H. Edelsbrunner, R. Seidel, and M. Sharir. On the zone theorem for hyperplane arrangements. *SIAM Journal on Computing*, 22(2):418–429, 1993.

[ESW01] S. Eidenbenz, C. Stamm, and P. Widmayer. Inapproximability results for guarding polygons and terrains. *Algorithmica*, 31:79–113, 2001.

[Fei98] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634 – 652, July 1998.

[FKK$^+$04] R. Fleischer, T. Kamphans, R. Klein, E. Langetepe, and G. Trippen. Competitive online approximation of the optimal search ratio. In *Proc. the 12th Annual European Symposium on Algorithms*, pages 335–346, 2004.

[FKN04] S. Fekete, R. Klein, and A. Nuchter. Online searching with an autonomous robot. In *Proc. Workshop on Algorithmic Foundation of Robotics*, pages 350–365, 2004.

[GB04] S. Ghosh and J. Burdick. Exploring an unknown polygonal environment with discrete visibility. Unpublished Manuscript, Tata Institute of Fundamental Research, Mumbai, India, 2004.

[GBL01] H. Gonzalez-Banos and J. Latombe. A randomized art-gallery algorithm for sensor placement. In *Proc. Seventeenth ACM Symposium on Computational Geometry*, pages 232 – 240, 2001.

[Gho87] S. Ghosh. Approximation algorithm for art gallery problems. In *Proc. the Canadian Information Processing Society Congress*, pages 1–7, 1987.

[GJ79] M. Garey and Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Co., New York, 1979.

[GKR00] N. Garg, G. Konjevod, and R. Ravi. A polylogorithmic approximation algorithm for the group steiner tree problem. *Journal of Algorithms*, 37:66–84, 2000.

[GMR97] L. Guibas, R. Motwani, and P. Raghavan. The robot localization problem. *SIAM Journal on Computing*, 26(4):1120–1138, 1997.

[GP02] G. Gutin and A. Punnen, editors. *The Traveling Salesman Problem and Its Variations*. Kluwer Academic Publishers, 2002.

[GW92] M. Goemans and D. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1992.

[HK03] E. Halperin and R. Krauthgamer. Polylogarithmic inapproximability. In *Proc. STOC*, pages 585–594. 2003.

[Hok] Hokuyo Automatic Co. Ltd. Urg-04lx scanning laser range finder.. http://www.hokuyo-aut.jp/products. Last accessed: Sep. 26, 2007.

[IKD03] V. Isler, S. Kannan, and K. Daniilidis. Local exploration: online algorithms and a probabilistic framework. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1913 – 1920, 2003.

[Kha80] L. Khachiyan. Polynomial algorithms in linear programming. *U.S.S.R. Comput. Maths. Math. Phys.*, 20(1):53–72, 1980.

[Kin06] J. King. *A 4-Approximation Algorithm for Guarding 1.5-Dimensional Terrains*. volume Volume 3887/2006 of *Lecture Notes in Computer Science*, pages 629–640. Springer Berlin / Heidelberg, 2006.

[KSLO96] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):556–580, 1996.

[KV00] B. Korte and J. Vygen. *Combinatorial Optimization*. Springer-Verlag, 2000.

[Lat91] J. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[LK99] S. LaValle and J. Kuffner. Randomized kinodynamic planning. In *Proc. IEEE International Conf. on Robotics and Automation*, pages 473–479, 1999.

[LL86] D. Lee and A. Lin. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, 32:276–282, 1986.

[LL06] L. Latecki and R. Lakaemper. Polygonal approximation of laser range data based on perceptual grouping and EM. In *2006 IEEE International Conference on Robotics and Automation*, pages 790–796, Orlando, Florida, May 2006.

[LPC$^+$00] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital Michelangelo project: 3D scanning of large statues. In *Proc. ACM SIGGRAPH 2000*, pages 131–144, 2000.

[MMP87] J. Mitchell, D. Mount, and C. Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 16(4):647–668, August 1987.

[Mob] MobileRobots Inc. Powerbot. http://www.activrobots.com/ROBOTS/power.html. Last accessed: Sep. 26, 2007.

[MP91] J. Mitechell and C. Papadimitriou. The weighted region problem: Finding shortest paths through a weighted planar subdivision. *Journal of the ACM*, 38(1):18–73, January 1991.

[Nil05] B. Nilsson. Approximate guarding of monotone and rectilinear polygons. In *Proc. 2005 International Colloquium on Automata, Languages and Programming*, pages 1362–1373, 2005.

[O'R87] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press., 1987.

[O'R98] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, second edition, 1998.

[OS83] J. O'Rourke and K. Supowit. Some NP-hard polygon decomposition problems. *IEEE Transactions on Information Theory*, 29(2):181–190, 1983.

[Pap85] C. Papadimitriou. An algorithm for shortest-path motion three dimensions. *Information Processing Letters*, 20:259–263, 1985.

[PS82] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithm and Complexity*. Prentice Hall, Englewood Cliffs, N.J., 1982.

[Ram81] T. Ramesh. Traveling purchaser problem. *Opsearch*, 18:78–91, 1981.

[RS99] R. Ravi and F. Salman. Approximation algorithm for the traveling purchaser problem and its variation in network design. In *Proc. of the 7th Annual European Symposium on Algorithms*, pages 29–40, 1999.

[She92] T. Shermer. Recent results in art galleries. *Proceedings of the IEEE*, 80(9):1384–1399, September 1992.

[SK04] C. Swamy and A. Kumar. Primal-dual algorithms for connected facility location problems. *Algorithmica*, 40:245–269, 2004.

[Sla97] P. Slavik. The errand scheduling problem. Technical Report 97-02, Sate University of New York at Buffalo, 1997.

[SR06] Z. Sun and J. Reif. On finding approximate optimal paths in weighted regions. *Journal of Algorithms*, 58:1–32, 2006.

[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):207–224, March 2006.

[SRR00] W. Scott, G. Roth, and J. Rivest. Performance-orientated view planning for model acquisition. In *Proc. International Symposium on Robotics (ISR 2000)*, pages 212–219, Québec, Canada, May 2000.

[SRR01] W. Scott, G. Roth, and J. Rivest. View planning with a registration component. In *Proc. the 3D Imaging and Modeling Conference (3DIM)*, pages 127–134, Québec, Canada, May 28 - June 1 2001.

[SRR03] W. Scott, G. Roth, and J. Rivest. View planning for automated three-dimensional object reconstruction and inspection. *ACM Computing Surveys*, 35(1):64–96, March 2003.

[Tan04] X. Tan. Approximation algorithm for the watchman route and zookeeper's problems. *Discrete Applied Mathematics*, 136(2-3):363–376, 2004.

[TBF05] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, 2005.

[Urr00] J. Urrutia. *Art Gallery and Illumination Problems*, chapter 22, pages 973-1027. Amsterdam, Netherlands: North-Holland, 2000.

[Vaz01] V. Vazirani. *Approximation algorithms*. Spinger, 2001.

[ZG05] A. Zarei and M. Ghodsi. Efficient computation of query point visibility in polygons with holes. In *Proc. Twenty-first Annual Symposium on Computational Geometry*, pages 314–320, 2005.