Algorithms for security in robotics and networks

Borislav H. Simov
Iowa State University

Follow this and additional works at: http://lib.dr.iastate.edu/rtd
Part of the Computer Sciences Commons, Mathematics Commons, and the Optics Commons

Recommended Citation
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI
Algorithms for security in robotics and networks

by

Borislav H. Simov

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:
Giora Slutzki, Co-major Professor
Steven M. LaValle, Co-major Professor
Soma Chaudhuri
Carolina Cruz-Neira
David Fernández-Baca

Iowa State University
Ames, Iowa
2003
Graduate College
Iowa State University

This is to certify that the doctoral dissertation of

Borislav H. Simov

has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Co-major Professor

Signature was redacted for privacy.

Co-major Professor

Signature was redacted for privacy.

For the Major Program
To my wife Ting-May and son Christo,
To my parents Violeta and Hristo.
To my aunt Rilka
# TABLE OF CONTENTS

## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>iv</td>
<td>v</td>
</tr>
</tbody>
</table>

## 1 INTRODUCTION

1.1 Pursuit-evasion in the Plane

1.1.1 Definition and variations of the pursuit-evasion problem

1.1.2 Results

1.2 Dissertation Organization

## 2 AN ALGORITHM FOR SEARCHING A POLYGONAL REGION WITH A FLASHLIGHT

2.1 Introduction

2.2 Notation and Preliminaries

2.2.1 Visibility and configurations

2.2.2 Linking the snapshots together

2.2.3 Compact encoding of a solution

2.3 Shelters and Skeleton of the Search Space

2.3.1 Critical points

2.3.2 Skeleton of \( X_n \)

2.3.3 Correspondence between winning legal paths and winning relaxed paths

2.4 Finding a Solution

2.4.1 Finding a relaxed path in time \( O(m^2 + m \log n + n) \)

2.4.2 Transforming a relaxed path into a legal path

2.5 Implementation and Comparison with Previous Work

2.6 Conclusion

## 3 CLEARING A POLYGON WITH TWO 1-SEARCHERS

3.1 Introduction
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.1</td>
<td>Related work</td>
<td>34</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Notation and preliminaries</td>
<td>35</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Semantics of the pursuit</td>
<td>37</td>
</tr>
<tr>
<td>3.2</td>
<td>Canonical Pursuit</td>
<td>38</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Canonical configurations</td>
<td>38</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Canonical schedule</td>
<td>40</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Changes to the canonical information states</td>
<td>44</td>
</tr>
<tr>
<td>3.3</td>
<td>Finite Representation</td>
<td>49</td>
</tr>
<tr>
<td>3.3.1</td>
<td>VHC boxes</td>
<td>50</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Concave regions and red intervals</td>
<td>51</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Equivalence classes in ( I ) and finite search schedule</td>
<td>52</td>
</tr>
<tr>
<td>3.4</td>
<td>Algorithm for Finding a Finite Search Schedule</td>
<td>54</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Constructing ( V(G) )</td>
<td>54</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Constructing ( E(G) )</td>
<td>56</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Finding a winning finite schedule</td>
<td>62</td>
</tr>
<tr>
<td>3.5</td>
<td>Conclusion</td>
<td>62</td>
</tr>
<tr>
<td>4</td>
<td>Integrating Security in the MAC Layer of WDM Optical Networks</td>
<td>63</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>64</td>
</tr>
<tr>
<td>4.2</td>
<td>Assumptions about the Scheduling Scheme</td>
<td>66</td>
</tr>
<tr>
<td>4.3</td>
<td>Secure Protocol</td>
<td>68</td>
</tr>
<tr>
<td>4.4</td>
<td>Solution against Attacks by Eavesdroppers Working Alone</td>
<td>70</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Configuration of the channels in Solution 1</td>
<td>70</td>
</tr>
<tr>
<td>4.4.2</td>
<td>The protocol</td>
<td>71</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Analysis of Solution 1</td>
<td>73</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Application of Solution 1 to an existing protocol</td>
<td>75</td>
</tr>
<tr>
<td>4.5</td>
<td>Solution against Attack by Two or More Collaborating Eavesdroppers</td>
<td>76</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Configuration of the channels in Solution 2</td>
<td>76</td>
</tr>
<tr>
<td>4.5.2</td>
<td>The protocol</td>
<td>78</td>
</tr>
<tr>
<td>4.5.3</td>
<td>Analysis of Solution 2</td>
<td>80</td>
</tr>
<tr>
<td>4.5.4</td>
<td>Application of Solution 2 to an existing protocol</td>
<td>82</td>
</tr>
<tr>
<td>4.6</td>
<td>Conclusions and Future Work</td>
<td>82</td>
</tr>
</tbody>
</table>
**LIST OF FIGURES**

2.1 (a) a simple polygon (b) corresponding visibility obstruction diagram ............... 9  
2.2 The size of a rectilinear legal path is not bounded by $n$ ................................. 13  
2.3 An illustration of critical points .............................................................. 16  
2.4 Illustration for the proof of Lemma 2.8 .................................................... 19  
2.5 Skeleton (a) and square grid (b) of the polygon from Figure 2.1 .................... 20  
2.6 Horizontal separation of conservative regions in the proof of Lemma 2.10. .... 21  
2.7 Inductive hypothesis for a monotone path, Lemma 2.12 .............................. 23  
2.8 Inductive hypothesis for a nonmonotone path, Lemma 2.12 .......................... 23  
2.9 Position of $R_0$, $R_1$, $T_0$, and $T_1$ in the proof of Lemma 2.13 ............... 25  
2.10 Inductive step in Proposition 2.14. ......................................................... 26  
2.11 Outline of the algorithm for finding a winning relaxed path ......................... 29  
2.12 A polygon which requires recontamination (between frames (d) and (e)). ....... 31  

3.1 The two kinds of gap edges: (a) left, (b) right ............................................. 36  
3.2 Contamination regions induced: (a) by configurations $(r_0, q_0)$ and $(r_1, q_1)$ and  
(b) by the corresponding canonical configurations $(p_0, q_0)$ and $(p_1, q_1)$ ....... 39  
3.3 Projecting motion in the interior onto motion on the boundary ....................... 40  
3.4 Visibility obstruction diagram (b) for the polygon in (a). The white regions in  
(b) correspond to the set $X_r$ of the mutually visible pairs of points in $\partial P$. .... 42  
3.5 Example of a VHC box $B$, VHC core $C$, and corresponding extreme points,  
denoted as black circles. ........................................................................... 50  
3.6 Example of bitangents (shown as dashed lines) and bitangent points (shown as  
points $c, c', d$ and $d'$). ............................................................................ 54  
3.7 (a) Tangent defined by points $a, b$ and $c$. (b) Corresponding configurations in  
$X$. The points in $X_r$ are white, the points in $X - X_r$ are grey ..................... 59  
3.8 Jump interval and partition of $X$ into regions of different labels ...................... 61
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>The original protocol</td>
<td>68</td>
</tr>
<tr>
<td>4.2</td>
<td>The channels in Solution 1 (Individual Eavesdroppers)</td>
<td>71</td>
</tr>
<tr>
<td>4.3</td>
<td>Actions of a user in Solution 1</td>
<td>73</td>
</tr>
<tr>
<td>4.4</td>
<td>Actions of the principal in Solution 1 and Solution 2</td>
<td>74</td>
</tr>
<tr>
<td>4.5</td>
<td>Comparison of the throughputs of the original protocol (above) and Solution 1 (below)</td>
<td>75</td>
</tr>
<tr>
<td>4.6</td>
<td>The channels in Solution 2 (Collaborating Eavesdroppers)</td>
<td>77</td>
</tr>
<tr>
<td>4.7</td>
<td>Actions of a user in Solution 2</td>
<td>78</td>
</tr>
<tr>
<td>4.8</td>
<td>Comparison of the throughputs of Solution 1 (above), Solution 2 (below), and the modified Solution 2 (middle)</td>
<td>81</td>
</tr>
<tr>
<td>5.1</td>
<td>The Petersen graph, a (3,2) Moore graph</td>
<td>88</td>
</tr>
<tr>
<td>5.2</td>
<td>Examples of (a) a heap and (b) a left compact tree</td>
<td>94</td>
</tr>
<tr>
<td>5.3</td>
<td>Heap with $d = 3$, $N = 25$ and $K = 3$</td>
<td>95</td>
</tr>
<tr>
<td>5.4</td>
<td>Expected multicast link utilization for various $m$ on 64-node random trees with $d \leq 2$ versus $h_{2,m}$</td>
<td>101</td>
</tr>
<tr>
<td>5.5</td>
<td>Expected multicast link utilization for various $m$ on 64-node random trees with $d \leq 4$ versus $h_{4,m}$</td>
<td>102</td>
</tr>
<tr>
<td>5.6</td>
<td>Expected multicast link utilization for various $m$ in the 64-node hypercube versus $h_{8,m}$</td>
<td>103</td>
</tr>
<tr>
<td>5.7</td>
<td>Expected multicast link utilization for various $m$ in the 64-node shufflenet versus $h_{2,m}$</td>
<td>104</td>
</tr>
<tr>
<td>5.8</td>
<td>Expected cost versus destination set size, for multicast, multiple-unicast, and broadcast traffic, for a network with $K = 5$ and $d = 2$ ($N_K = 63$)</td>
<td>105</td>
</tr>
<tr>
<td>5.9</td>
<td>Maximum benefit point</td>
<td>106</td>
</tr>
<tr>
<td>5.10</td>
<td>Normalized maximum benefit point</td>
<td>106</td>
</tr>
<tr>
<td>5.11</td>
<td>Maximum benefit</td>
<td>107</td>
</tr>
<tr>
<td>5.12</td>
<td>Normalized maximum benefit</td>
<td>107</td>
</tr>
<tr>
<td>C.1</td>
<td>Example of a heap with $N = 13$ and $d = 2$</td>
<td>115</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

In this chapter we present an overview of visibility-based pursuit-evasion in the plane and also discuss the organization of the dissertation.

1.1 Pursuit-evasion in the Plane

In the last decade there has been a steady interest in robotics and computational geometry in designing motion strategies for pursuit-evasion scenarios. The basic task is to compute motion strategies for one or more robots (pursuers) to guarantee that unpredictable targets (evaders) will be detected using sensors. A key difficulty which makes the problem more challenging than basic exploration is that the evaders can sneak back to places already explored. Efficient algorithms that compute these strategies can be embedded in a variety of robotics systems to locate other robots and people. They can aid mobile surveillance systems that detect intruders using sonars, lasers, or cameras. Mobile robots can be used by special forces in high-risk military operations to systematically search a building in enemy territory before it is declared safe for entry. If robots have limited communication, the algorithms could be used to help robots locate each other for coordinated tasks such as map-building, localization, or target tracking. Beyond robotics, these algorithms could help in the training of firefighters engaged in a rescue effort, law enforcement officers in a hostage situation, or soldiers attempting to secure a potentially-hostile area. The strategies can also be used in virtual environments to help game developers design complicated pursuit strategies.

1.1.1 Definition and variations of the pursuit-evasion problem

We now proceed with a formal definition of the pursuit-evasion problem in the plane. Consider a simple connected finite region in the plane. The region can contain infinitely small objects who can move along continuous paths. The objects are of two different types: evaders and pursuers. The motions of an evader are random, i.e., unpredictable to the pursuers. The pursuers are equipped with 1 or more flashlights (sources of a ray of light) or alternatively with a source of light which illuminates all visible
points of the region. (We assume that the region is hollow: the points inside it are transparent, the boundary points are opaque yet they do not reflect light.) The goal of the pursuers is to illuminate all the evaders, or to verify that there are no evaders in the closed region, a.k.a. "clear the region". The pursuit-evasion problem is defined as finding a schedule for clearing the region. In its simple decision variant, given a region and a number of pursuers, the question is whether there exists a schedule of the pursuers which clears the region. Alternatively, there is a more general, minimization version: given a region, what is the minimal number of pursuers required to clear it? Also for a region which we can be cleared, we are interested in a clearing schedule: a description of the corresponding moves of the pursuers which guarantees that the region is clear.

The first, restricted version of a pursuit-evasion problem in the plane was introduced by Icking and Klein [20] who solved the "two-guards problem" (two pursuers had to maintain mutual visibility all the time). Independently, the general pursuit-evasion problem was first introduced by Suzuki and Yamashita [49], who considered several pursuers moving independently (no restriction on mutual visibility) in a polygonal region, collaborating in a search for an evader.

The pursuit-evasion problem can be classified according to several criteria, among which: number of the pursuers, visibility capability of the pursuer, shape of the area that has to be cleared, etc. Following is a description of some of the versions together with corresponding research.

**Shape of the region**

Most of the research in pursuit-evasion has been done with the assumption that the region to be cleared is a simple polygon. However, LaValle and Hinrichsen [25] considered a similar "curved world" problem in which the boundary of the region is a smooth curve with tangent defined at every point. In fact, we believe that the pursuit-evasion decision problem depends solely on the position of the concave regions of the boundary; the exact shape of region can be ignored since minor changes in the boundary will not change the result of the decision problem. Thus an algorithm which solves a version of the problem for polygons can be extended to a similar algorithm solving the corresponding version in which the region is bounded by a simple curve with a finite number of concave regions, i.e., a finite number of points in which the tangent switches from inside to outside the region.

**Number of flashlights for the pursuer**

There are different versions of the problem depending on the visibility capability of the pursuers. In the simplest case, a pursuer is equipped with a single flashlight and can see only along the ray of light
emitted by it. Furthermore, we can increase the power of the pursuer by allowing her to hold up to \( k \) flashlights, where \( k \in \mathbb{N} \), these pursuers are also known as \( k \)-searchers. Ultimately, we can allow each of the pursuers 360° vision, by essentially assuming that the number of flashlights \( k \) is infinite.

**Number of pursuers**

LaValle et al [26] showed that the problem of finding the minimal number of pursuers required to clear a given polygon is NP-hard. The proof, however, assumed that the polygon may have holes. It is not known whether a similar result can be obtained for polygons without holes.

**Independent pursuers vs. chain of pursuers**

Icking and Klein [20] defined and solved a search problem for two guards who move on the boundary of a polygon and are always mutually visible. Heffernan [16] later improved the time complexity of their result to the optimal \( \Theta(n) \), where \( n \) is the number of edges in the polygon.

Efrat et al [12] considered pursuit-evasion by a chain of \( k \) guards as a generalization of the search with two guards. Their pursuit is subject to the restriction that the first and the \( k \)-th guards always move on the boundary while guard \( i \), \( 1 < i < k \) moves in the interior of the polygon and maintains visibility with her neighbors, guards \( i - 1 \) and \( i + 1 \). Efrat et al [12] gave a polynomial-time algorithm for the \( k \) guards problem.

**1.1.2 Results**

LaValle et al [26] presented a complete algorithm for an \( \infty \)-searcher clearing a polygon, however its running time may be exponential in the number of edges of the polygon.

Recently, Simov et al presented an \( O(n^3) \) solution for a single 1-searcher in a polygon with \( n \) edges [42], a result which they later [28] improved to \( O(n + m \log n + m^2) \), where \( m \) is the number of concave regions of the polygon. Park et al [35] presented a polynomial-time solution for the case of a single 2-searcher and proved that adding more flashlights to a single pursuer does not increase the class of the polygons she can clear.

**1.2 Dissertation Organization**

The dissertation is organized as follows. In Chapter 2 we present a polynomial-time algorithm for clearing a polygon by a single 1-searcher. We extend the result to a polynomial-time algorithm for a
pair of 1-searchers in Chapter 3. Both chapters are based on research done together with Dr. Giora Slutzki and Dr. Steven LaValle.

Chapters 4 and 5 represent joint research with Dr. Srin Tridandapani, Dr. Jason Jue and Dr. Michael Borella in the area of computer networks. Chapter 4 presents a method of providing privacy over an insecure channel which does not require encryption. Chapter 5 gives approximate bounds for the link utilization in multicast traffic.

We conclude the dissertation with Chapter 6 by giving some directions for future research in visibility-based pursuit-evasion.
2 AN ALGORITHM FOR SEARCHING A POLYGONAL REGION
WITH A FLASHLIGHT

A paper published in the International Journal of Computational Geometry and Applications
Steven M. LaValle, Borislav H. Simov and Giora Slutzki

Abstract

We present an algorithm for a single pursuer with one flashlight searching for an unpredictable, moving target in a 2D environment. The algorithm decides whether a simple polygon with \( n \) edges and \( m \) concave regions can be cleared by the pursuer, and if so, constructs a search schedule in time \( O(m^2 + m \log n + n) \). The key ideas in this algorithm include a representation called "visibility obstruction diagram" and its "skeleton": a combinatorial decomposition based on a number of critical visibility events. An implementation is presented along with a computed example.

2.1 Introduction

Consider the following scenario: in a (dark, doorless) polygonal region there are two moving agents (represented as points). The first one, called the pursuer, has the task to find the second one, called the evader. The evader can move arbitrarily fast, and his movements are unpredictable by the pursuer. The pursuer is equipped with a flashlight and can see the evader only along the illuminated line segment it emits. The pursuer (a.k.a. 1-searcher) wins if she illuminates the evader with her flashlight or if both happen to occupy the same point of the polygon. Clearly, the pursuer should, at all times, be located on the boundary of the polygon and use the flashlight as a moving boundary between the portion of the polygon that has been cleared (i.e., the evader is known not to hide there) and the contaminated.
portion of the polygon (i.e., the part in which the evader might be hiding). If there is a movement strategy of the pursuer whereby she wins regardless of the strategy employed by the evader, we say that the polygon is searchable with one flashlight, or 1-searchable.

The problem above was introduced by Suzuki and Yamashita [49], who were interested in the existence and complexity of an algorithm which, given a simple polygon \( P \) with \( n \) edges, decides whether \( P \) is 1-searchable and if so, outputs a search schedule. Although the problem has been open for a while, no complete characterizations or efficient algorithms were developed. Naturally, several, restricted variants were considered. Independently of [49], Icking and Klein [21] defined the two-guard walkability problem, which is a search problem for two guards whose starting and goal position are given, and who move on the boundary of a polygon so that they are always mutually visible. Icking and Klein gave an \( O(n \log n) \) solution, which later was improved by Heffernan [17] to the optimal \( \Theta(n) \). Tseng et al [52] extended the two-guard walkability problem by dropping the requirement that the starting and goal positions are given. They presented an \( O(n \log n) \) algorithm which decides whether a polygon can be searched by two guards, and a \( O(n^2) \) algorithm which outputs all the possible starting and goal positions which allow searchability by two guards. Recently, Lee et al [30] defined 1-searchability for a room (i.e., a polygon with one door — a point which has to remain clear at all times) and presented an \( O(n \log n) \) decision algorithm and a method to construct a solution in time \( O(n^2) \). In this paper we solve the original problem defined in [49], and we show that it is a nontrivial generalization of the variants of 1-searchability defined in [21] and [30].

Originally, the problem of 1-searchability of a polygon was introduced in [49] together with a more general problem in which the pursuer (a.k.a. \( k \)-searcher) has \( k \) flashlights; when \( k \) is not bounded this corresponds to a 360° vision. For results concerning 360° vision refer to [49, 10, 15, 31] for search in polygons and to [25] for curved planar environments.

Our model is motivated in part by the need in mobile robotics systems to develop simple sensing mechanisms and to minimize localization requirements (knowing the precise location of the robot). The “flashlight” could be implemented by a camera and vision system that uses feature detection to recognize a target. Alternatively, a single laser beam could be used to detect unidentified changes in distance measurements. Many localization difficulties are avoided since the robot is required to follow the boundary of the environment. Sensors could even be mounted along tracks that are fastened to the walls of a building, as opposed to employing a general-purpose mobile robot. Although it is obviously restrictive to consider only environments that can be cleared by a single pursuer, the problem considered in this paper is rather challenging. In addition, it may be possible to extend some of the ideas in the
algorithm for a single 1-searcher to allow the coordination of multiple pursuers, eventually broadening the scope of applications.

Although the algorithm presented in Section 4 is not complicated conceptually, a rigorous correctness proof requires rather delicate geometric considerations which are presented thoroughly in Sections 2 and 3. The concepts from these sections also provide a general framework in which other pursuit-evasion problems could be studied.

The rest of the paper is organized as follows. Section 2.2 introduces the notation and provides observations which reduce pursuit-evasion by a 1-searcher to a search problem on a torus. In Section 2.3.1 we define critical points on the boundary which are essential for determining the solution schedule of the 1-searcher. In Section 2.3.2 we use these points to reduce the search as presented in Section 2.2 to a search in a finite maze. In Section 2.4 we present an $O(m^2 + m \log n + n)$ algorithm which, given a polygon with $n$ edges and $m$ concave regions, decides whether the polygon is searchable and if so, constructs a schedule for the 1-searcher. Section 2.5 discusses an implementation of the algorithm, and also the relationship between our result and the results in [21, 17, 52], [12] and [30]. Section 2.6 concludes the paper with a summary and directions for future research.

2.2 Notation and Preliminaries

2.2.1 Visibility and configurations

Let $P$ be a simple polygon. (From now on, a polygon is always assumed to be simple.) We denote the boundary of $P$ by $\partial P$. We assume that $\partial P \subseteq P$ and that $\partial P$ is oriented in the clockwise (also called positive) direction. For any two points $a, c \in \partial P$, we write $(a, c)$ to denote the open interval of all points $b \in \partial P$ such that when starting after $a$ in positive direction along $\partial P$, $b$ is reached before $c$. We also use the notation $[a, c]$, $[a, c)$ and $(a, c]$ for the closed and half-closed intervals on $\partial P$.

Let $p_0, p_1, \ldots, p_{n-1}$ denote the vertices on $P$ ordered in the positive direction. The edges of $\partial P$ are $e_0, e_1, \ldots, e_{n-1}$, where edge $e_i$ has endpoints $p_i$ and $p_{i+1}$, where $i \in \mathbb{Z}_n$ (i.e., the indices are computed modulo $n$; e.g., $p_0 = p_n$).

We use the standard definition of visibility. For points $c, d \in \partial P$ we say that $d$ is visible from $c$, if every interior point of the line segment $\overline{cd}$ lies in $P - \partial P$. Obviously, if one point is visible from another, then the two are mutually visible. Note that no two points on the same edge of $P$ are mutually visible.

Consider a simple example of how the pursuer can clear the polygon in Figure 2.1(a). Initially, she is at point 0 with the flashlight pointing at 0. To start the search she moves from point 0 to point 14,
while at the same time\(^3\) she rotates the flashlight from point 0 to point 1. Next, the pursuer, while staying at 14, rotates the flashlight from point 1 clockwise to point 5. Then she moves from point 14 to point 13 constantly illuminating point 5. Following that, she rotates the flashlight from point 5 to point 7, and then moves from point 13 to point 10. After a final rotation of the flashlight from point 7 to point 8, the pursuer is at point 10 illuminating point 8. The search is completed when she moves from 10 to 9 and simultaneously rotates\(^3\) the endpoint of the flashlight from 8 to 9.

The following observations are immediate. Because of his unbounded speed, we can assume that the evader moves only along the boundary. As we mentioned above, the pursuer also moves along \(\partial P\) and uses the flashlight to separate the clear and contaminated portions of the polygon. Furthermore, without loss of generality, we will assume in the rest of the paper that the clear portion of the polygon is always to the left of the pursuer as she looks in the direction of the beam of light emitted by her flashlight. We call this assumption the **left invariant**. Thus, a single pair of points is sufficient to record the current status of the pursuit as seen by the pursuer. We define a **configuration** to be a pair \((p, q)\) of points \(p, q \in \partial P\), and the space of all configurations \(X\) to be:

\[
X = \partial P \times \partial P = \{ (p, q) \mid p, q \in \partial P \}.
\]

Let \(X_d \subset X\) denote the set of all **diagonal configurations** \((p, q)\) such that \(p\) and \(q\) lie on the same edge of \(\partial P\). We denote the set of all **feasible configurations**, \(X_v \subseteq X\), consisting of pairs \((p, q)\) of mutually visible points, i.e., such that \(q\) is visible from \(p\). Intuitively, for \((p, q) \in X_v\), \(p\) represents the position of the pursuer, while \(q\) represents the point illuminated by the flashlight. If the left invariance assumption holds for configuration \((p, q)\), then all points between \(p\) and \(q\) (along \(\partial P\)) are clear and all the points between \(q\) and \(p\) are contaminated. Finally, let the set of all **non-feasible configurations**, \(X_n = X - X_d - X_v\), consist of all pairs \((p, q)\) of points which do not lie on the same edge and are not mutually visible. Thus a non-feasible configuration represents a situation, that cannot arise because of the geometry of \(P\).

**Definition 2.1.** The **visibility obstruction diagram (VOD)** for a polygon \(P\) is defined as the 3-partition of \(X\), \((X_d, X_v, X_n)\), where \(X_d\), \(X_v\) and \(X_n\) are defined as above.

Figure 2.1 provides an example of a simple polygon and the VOD corresponding to it. The black squares along the diagonal of Figure 2.1(b) represent \(X_d\), the shaded area represents \(X_n\), and the white

\(^3\)Note that the beginning and the end of the search are the only times when the pursuer has to simultaneously move herself and rotate the flashlight.
area represents $X_v$.

2.2.2 Linking the snapshots together

In the previous section we have shown how a single snapshot of the pursuit in polygon $P$ can be represented as a feasible configuration in $X_v \subseteq X$, where $X$ is the VOD corresponding to $P$. Next we are interested in a way to encode an entire strategy of the pursuer. Without loss of generality we can assume that the pursuer starts at time $t = 0$ and finishes at time $t = 1$. For each $t$, $0 < t < 1$, let $T(t) \in X_v$ denote the feasible configuration that the pursuer is in at time $t$. That is, let $T(t) = (T_1(t), T_2(t))$, where $T_1(t) \in \partial P$ denotes the position of the pursuer and $T_2(t) \in \partial P$ denotes the point on the boundary illuminated by the flashlight at time $t$. We are interested in $T_1(t)$, $T_2(t)$ and $T(t)$ as functions of the time $t$, because we would like to represent a pursuer strategy (i.e., continuous motion of the pursuer and rotation of the flashlight) as a path in $X_v$. Clearly, every strategy of the pursuer can be mapped via $T$ into a path in $X_v$. On the other hand, does every path in $X_v$ have a corresponding pursuer strategy? The answer is "no". If $T(t)$ corresponds to a pursuer strategy, then it does not contain:

1. a vertical jump: this would imply discontinuity in $T_1(t)$, which is impossible since the pursuer must move continuously along the boundary. Therefore, if there are any discontinuities in $T(t)$,
they have to be horizontal jumps.

2. a horizontal jump over points in $X_v$: this would imply that a pursuer who is stationary at point $p \in \partial P$ and is rotating her flashlight from point $q_1$ to point $q_2$ does not illuminate some point between $q_1$ and $q_2$. But this is impossible, since all points between $q_1$ and $q_2$ are visible from $p$, and every visible point must be illuminated. Therefore, if there are any horizontal jumps in $T(t)$, they have to be over $X - X_v = X_d \cup X_n$.

3. a horizontal jump over $X_d$: this would correspond to the pursuer pointing the flashlight outside the polygon, which is impossible. Therefore, horizontal jumps in $T(t)$ are only possible over $X_n$.

4. a left-to-right horizontal jump over configurations in $X_n$: this would correspond to a stationary pursuer who tries to rotate the flashlight clockwise across an invisible interval — this will invalidate the left invariant. For example, in Figure 2.1(a), if the pursuer is at point 14 and rotates the flashlight from point 5 to point 12 (over the invisible points between 6 and 11), this will cause the whole polygon to be contaminated again, thus losing all the work to this point. This invalid move is shown as the left-to-right dotted arrow in Figure 2.1(b).

Finally, by combining (1)-(4) we conclude that if there is any discontinuity in $T(t)$, it can be only in the form of a horizontal jump from right to left over $X_n$. This corresponds to a stationary pursuer, who rotates the flashlight counterclockwise across an interval of invisible points. For example, in Figure 2.1(a), if the pursuer is at point 13 and rotates her flashlight from point 5 to point 2 (over the interval of invisible points between 3 and 4), the move is represented in Figure 2.1(b) as the right-to-left dotted arrow in $X$. We call this a recontamination move since in the example originally the points on the boundary between 13 and 5 are clear, while after the move only the points between 13 and 2 remain clear, i.e. the interval between the points 2 and 5 is contaminated again.

Note that, technically, recontamination happens every time when the pursuer rotates the flashlight in counterclockwise direction. However, since counterclockwise rotation over visible points is reversible (i.e., a subsequent clockwise rotation of the flashlight to the original position preserves the left invariant), we consider it as a trivial form of recontamination and reserve the term "recontamination" for the irreversible counterclockwise rotation of the flashlight over an interval of non-visible points.

**Definition 2.2.** Let $T(t) = (T_1(t), T_2(t)) \in X_v$, $0 < t < 1$, be a piecewise continuous path in $X_v$ which has exactly $k$ discontinuities ($k \geq 0$) at times $t_1, t_2, \ldots, t_k$, such that every discontinuity represents a horizontal jump from right to left over $X_n$; see the dotted horizontal segment labeled $J$ in Figure 2.1(b).
Let for some nonreflex vertex $p \in \partial P$, $T(0) = (p,p) = \lim_{t \to 0^+} T(t)$ be the beginning of the path, such that $T(t)$ approaches $T(0)$ from above and from the right. (This means that the path starts from a configuration in which almost the entire boundary is contaminated.) We call such a path $T(t)$ a legal path. Also, if $T(t)$ is a legal path in which for some nonreflex vertex $q \in \partial P$, $T(1) = (q,q) = \lim_{t \to 1^-} T(t)$ is the end of the path, such that $T(t)$ approaches $T(1)$ from below and from the left (this means that the path ends in a configuration in which almost the entire boundary is clear), we say that $T(t)$ is a winning legal path or simply a winning path.

For example, the winning path representing the solution described in Figure 2.1(a) is plotted as the rectilinear path of solid arrows in Figure 2.1(b) starting from the letter 'S' (representing $T(0)$) in the lower right corner, wrapping around and ending at the letter 'F' (representing $T(1)$).

From the discussion above it is clear that there is a one-to-one correspondence between (winning) pursuer strategies and (winning) legal paths. This implies the following proposition.

**Proposition 2.3.** Let $P$ be a polygon with a VOD $X$. $P$ is 1-searchable if and only if there exists a winning path in $X$.

Note that $T_1(t)$ and $T_2(t)$ are treated differently. Since the pursuer moves continuously on the boundary, then $T_1(t)$ is also continuous. On the other hand, since the endpoint of the flashlight can jump over invisible intervals of $\partial P$, $T_2(t)$ is piecewise continuous: we allow $T_2(t)$ to make horizontal jumps from right to left over $X_\mathbb{N}$. This is the main difference between the problem of clearing a polygon with a single 1-searcher as compared to the problem of clearing a polygon with two guards [21]. The two-guards problem can also be represented as a search in the VOD, with the only difference that $T_2(t)$ and therefore $T(t)$ have to be continuous, i.e., the recontamination move is disallowed.

If for a polygon $P$ there exists a winning path without a recontamination move, we say that $P$ can be cleared without recontamination. On the other hand, if all of the legal winning paths for $P$ contain a recontamination move, we say that clearing $P$ requires recontamination. (For a more detailed example of a polygon, which requires recontamination, see Figure 2.12, discussed in detail in Section 2.5.)

### 2.2.3 Compact encoding of a solution

In the previous section we showed that in order to check whether a polygon is 1-searchable, it suffices to verify whether there exists a winning path in its VOD. However, if in addition we want our

---

4The circularity of $\partial P$ implies that there is a vertical and horizontal wraparound along each of the axes, making $X$ into a torus.
algorithm to describe the winning path (i.e., output a winning strategy of the pursuer) we have to take into account several considerations. If a polygon is 1-searchable, then the number of existing winning paths is infinite. They may vary by shape or length, so, intuitively, we would like to output a relatively simple (shorter, smoother, with less detours) path $T(t)$. Even in this case, it is clear that we cannot expect the algorithm to print all the configurations $T(t)$ in the time interval $t \in [0,1]$. This implies that instead, we have to print a finite sequence

$$T(t_0 = 0), T(t_1), T(t_2), \ldots, T(t_{k-1}), T(t_k = 1),$$

which conforms to the following two requirements:

- the length of the sequence, $k$, is as small as possible, (ideally, $k$ is bounded by a polynomial in $n$, since we are looking for an algorithm which runs in polynomial time)
- each move from configuration $T(t_i)$ to configuration $T(t_{i+1})$, $0 \leq i < k$ belongs to a finite set of "elementary moves": these moves should be easy to describe and also easy to perform in a real-world implementation.

Note that the two requirements are conflicting so next we present a reasonable tradeoff between them, by carefully defining the set of "elementary moves".

2.2.3.1 The naive approach: rectilinear paths in $X_v$

The most straightforward, yet flawed, approach to choosing a set of elementary moves is to assume that during any elementary move either the pursuer or the illumination point is stationary. Thus the motions of the pursuer and the rotations of the flashlight can be represented by an alternating sequence of vertical or horizontal directed segments in $X$ as follows:

1. Flashlight rotation in clockwise direction over visible points and stationary pursuer corresponds to a horizontal segment in $X_v$ directed from left to right. In Figure 2.1, the rotation from point 1 to point 5 in the beginning of the solution is represented by the first (horizontal) arrow in the path.

2. Flashlight rotation in counterclockwise direction over visible points and stationary pursuer corresponds to a horizontal segment in $X_v$ directed from right to left.

3. Motion in counterclockwise direction of the pursuer with the flashlight illuminating the same point corresponds to a vertical segment in $X_v$ directed up. In Figure 2.1, the move of the pursuer from
point 14 to point 13 while illuminating point 5 is represented by the second (vertical) arrow in the path.

4. Motion in clockwise direction of the pursuer with the flashlight illuminating the same point corresponds to a vertical segment in $X_v$ directed down.

5. Recontamination move.

One advantage of choosing the five moves above as elementary ones is that they are simple and easy to describe. However, the examples in Figure 2.2 suggest, that there is a serious drawback if we use only rectilinear moves to describe a winning strategy for the pursuer even in a relatively simple polygon. Detailed explanation follows.

Consider first the polygon in Figure 2.2(a). It can be searched, for example by the following rectilinear winning path:

$$(0,0), (15,1), (15,3), (13,3), (13,\beta), (\gamma,\beta), (\gamma,\alpha), (10,\alpha), (10,4), (8,4), (8,6), (7,7)$$

While this is not the only rectilinear winning path, an important property of the polygon is that in order to get from configuration $(13,3)$ to configuration $(10,4)$, we need to introduce intermediate stopping points $\alpha$, $\beta$ and $\gamma$.

Let us now change slightly the shape of the polygon. We move vertices 13 and 10 horizontally towards each other, leaving the rest of the vertices intact, see Figure 2.2(b). The original position of the edges $[10,11]$ and $[12,13]$ is shown as a dashed line. How does a rectilinear winning path for the new polygon look like? It will have at least 9 intermediate stopping points:

$$(0,0), (15,1), (15,3), (13,3), (13,e), (i,e), (i,d), (h,d), (h,c), (g,c), (g,b), (f,b), (f,a), (10,a), (10,4), (8,4), (8,6), (7,7)$$
Note that the polygons in Figure 2.2(a) and (b) are very similar. In particular, they have the same number of edges, \( n \), and furthermore, the visibility relation between edges and vertices is the same in the two polygons. So at least intuitively, the length of the minimal rectilinear winning paths should be the same. However, as we have shown, the second polygon requires a longer rectilinear winning path. Since we can use a similar construction to build a polygon in which the vertices 10 and 13 are arbitrarily close, it follows that we can increase the length of its rectilinear winning path to infinity. Thus the length of the path and hence the time to describe a solution, does not depend on the number of edges \( n \), therefore the proposed set of elementary moves is unsuitable for a compact encoding of a solution.

### 2.2.3.2 Encoding a solution with \( O(n^2) \) moves

Since several other papers dealing with 1-searchability \([49, 30]\) had to formally describe a strategy of the pursuer, a valid question at this point is, how did they encode a solution? While a detailed definition can be found in \([49, 30]\), we can summarize the approach, which we call conservative, as follows. The set of instructions contains the recontamination move, as well as, four other instructions, which allow simultaneous moves of the pursuer and rotations of the flashlight as long as those do not cross a vertex of \( \partial P \). It is not hard to show that this approach limits the maximal length of a solution to \( \Theta(n^2) \), therefore the time for an algorithm which has to print a solution is \( \Omega(n^2) \).

There is a minor drawback to using the conservative set of instructions to describe a solution. Consider the following example from Figure 2.2(a). Suppose that the pursuer is at point 15 and is illuminating point 1. If the pursuer starts to rotate the flashlight in clockwise direction until it illuminates point 3, this intuitively represents a single move: \((15,1) \rightarrow (15,3)\). However, in the conservative approach, we have to break the rotation of the flashlight at every vertex that we illuminate, thus in this case, the rotation will be described as: \((15,1) \rightarrow (15,2) \rightarrow (15,3)\). An extreme example would be the case of a regular \( n \)-sided polygon in which every pair of points is visible. Clearly, this is the simplest possible instance of the problem and can be cleared with a single rotation of the flashlight. On the other hand, if we use the conservative instructions, the single rotation has to be broken into a series of \( n \) instructions, one for each vertex that is illuminated by the flashlight.

Intuitively, it is not necessary to break up the moves of the pursuer or the rotations of the flashlight just because they cross a vertex. In the next section we present a set of elementary instructions which combines the advantages of the previous two approaches and allows us to encode every solution with \( O(m) \) instructions, where \( m \), the number of concave regions in the polygon is defined in Section 2.3.1.
2.2.3.3 Atomic moves

In this section we define the five atomic moves, which will be the basic building blocks for any legal path. One of the five moves is the recontamination move. The other four moves resemble the four conservative moves from the previous section: the pursuer is moving and simultaneously is rotating the flashlight. We call these moves northeast (NE), northwest (NW), southeast (SE) and southwest (SW) because these are the directions of the respective paths in the corresponding configuration space $X$ (the VOD of $P$).

**Definition 2.4.** Let $p_1, p_2, q_1, q_2 \in \partial P$ be in clockwise order on the boundary, such that, $c' = (p_2, q_1)$, $c'' = (p_1, q_2) \in X_v$. Let also $Q \subseteq X - X_d$ be the rectangle with lower left corner $c'$ and upper right corner $c''$. We say that there is an atomic NE move from $c'$ to $c''$ if:

(i) $Q \subseteq X_v$, i.e., every configuration in $Q$ is feasible, or

(ii) for every feasible configuration $c \in Q \cap X_v$, there is an atomic NE move from $c'$ to $c$ and an atomic NE from $c$ to $c''$.

Case (i) can be illustrated by the following example, see Figure 2.2(a). The move from configuration $(14,3)$ to configuration $(13,\beta)$ is a northeast one. To show this, we let $p_1 = 13$, $p_2 = 14$, $q_1 = 3$ and $q_2 = \beta$. Clearly, every point in $[q_1,q_2]$ is visible from every point in $[p_1,p_2]$. Thus the rectangle $Q$, as defined above, contains only feasible configurations and the case (i) holds. A move from $(15,1)$ to $(13,\beta)$ is an example for an atomic NE move, illustrating case (ii). The atomic SW, atomic SE and atomic NW moves are defined similarly.

An immediate observation is that each of the five rectilinear elementary moves defined in Section 2.2.3.1 is also an atomic one. Furthermore, if we now use the atomic moves, a solution (i.e., winning path) for the polygon in Figure 2.2(a) can be represented as:

$$(0,0) \rightarrow (15,1) \xrightarrow{NE} (13,\beta) \xrightarrow{NW} (10,\alpha) \xrightarrow{NE} (8,6) \rightarrow (7,7),$$

where NE and NW stand for the atomic northeast and northwest moves. An important observation is that Figure 2.2(b) has almost identical solution:

$$(0,0) \rightarrow (15,1) \xrightarrow{NE} (13,e) \xrightarrow{NW} (10,\alpha) \xrightarrow{NE} (8,6) \rightarrow (7,7).$$

This illustrates the clear advantage of the set of atomic instructions compared to the rectilinear ones.
defined in Section 2.2.3.1.

Also, a solution for the case of a \( n \)-sided regular polygon can be described with a single northeastern move, which is an advantage as compared with the conservative moves from Section 2.2.3.2.

2.3 Shelters and Skeleton of the Search Space

In Section 2.3.1 we use simple geometric conditions to introduce "critical points", which provide the basis for determining the alternating order of the moves of the pursuer and the rotations of the flashlight. In Section 2.3.2 we use the critical points to construct a more compact, "skeleton" representation of the VOD.

2.3.1 Critical points

Vertex \( p_i \in \partial P \) is a reflex vertex if the angle formed by incident edges \( e_{i-1} \) and \( e_i \), in the interior of \( P \), is greater than 180° (i.e., points \( p_{i-1}, p_i, \) and \( p_{i+1} \), form a left turn). Otherwise, \( p_i \) is a non-reflex vertex. A maximal subinterval of \( \partial P \) of the form \( C = (p_i, p_{j+1}) \), where all the vertices \( p_{i+1}, \ldots, p_j \) are reflex vertices, forms a concave region. Obviously, two concave regions cannot overlap and must be separated by non-concave regions (each of which contains at least one non-reflex vertex). For example, Figure 2.1(a) shows two concave regions labeled \( A \) and \( B \).

![Figure 2.3 An illustration of critical points](image)

Inasmuch as the concave regions of \( P \) represent the obstructions to visibility, their extreme edges represent the best hiding places for the evader (in view of his unbounded speed). Consider the \( k \)-th concave region \( C_k = (p_i, p_{j+1}) \) of \( P \), see Figure 2.3. Define shelters of \( C_k \) to be the two points \( a_k \) and \( b_k \).
$b_k$, the midpoints of the edges $e_i$ and $e_j$, respectively. Shoot a ray [7, 18] starting at point $a_k$ through $p_{i+1}$ and let $x$ be the point of $\partial P$ where the ray leaves $P$ for the first time. Define the threshold of $a_k$, denoted by $a'_k$, to be the point in $\overline{a_kx} \cap \partial P$ which is farthest away from $a_k$ in positive direction along the boundary. Similarly, shoot a ray starting at point $b_k$ through $p_j$ and let $y$ be the point of $\partial P$ where the ray leaves $P$ for the first time. Define the threshold of $b_k$, $b'_k$, to be the point in $\overline{b_ky} \cap \partial P$ which is farthest from $b_k$ in negative direction along the boundary. Given the points $a_k$ and $a'_k$ define $\mathcal{H}(a_k) = \{a_k, a'_k\} \subset \partial P$. Note that $a_k$ is not visible from any point in $\mathcal{H}(a_k)$. Thus, intuitively, whenever the pursuer stays in $\mathcal{H}(a_k)$, the evader can hide at $a_k$ and is guaranteed not to be detected by the pursuer. Similarly, we define $\mathcal{H}(b_k) = \{b'_k, b_k\} \subset \partial P$.

We have defined, for each concave region $C_k$ of $P$, four types of important points: two shelters, $a_k$ and $b_k$, and two corresponding thresholds, $a'_k$ and $b'_k$. These are collectively called the critical points of $C_k$ and the sets of all such points, for the various concave regions of $P$, are denoted, according to types, by $A$, $B$, $A'$, and $B'$ respectively:

\[
A = \{a_0, \ldots, a_{m-1}\}, \quad A' = \{a'_0, \ldots, a'_{m-1}\}
\]
\[
B = \{b_0, \ldots, b_{m-1}\}, \quad B' = \{b'_0, \ldots, b'_{m-1}\}.
\]

Note that in general $A'$ and $B'$ can be multisets.

**Lemma 2.5.** Let $P$ be a polygon with $n$ edges and $m$ concave regions. The critical points of $P$ and their order along $\partial P$ can be found in time $O(n + m \log n)$.

**Proof:** The points in $A$ and $B$ can be found in time $\Theta(n)$. To find the points in $A'$ and $B'$ we can use the ray-shooting algorithms of Chazelle et al [7]. After a preprocessing time of $O(n)$, we can find each threshold by a single ray-shooting query in time $O(\log n)$. Thus we perform $2m$ queries for a total time $O(n + m \log n)$. 

\[2.3.2 \text{ Skeleton of } X_n\]

One of the main ideas of our algorithm is that instead of trying to determine (and use) the exact shape of $X_n$, we will use the information obtained from the critical points to build an “equivalent” search space that is computationally much more convenient. This leads to a skeletal representation of the set of infeasible configurations, $X_n$, in the VOD of the polygon $P$, and will allow us to construct a solution in time $O(m^2 + m \log n + n)$, where $m$ is the number of concave regions of $P$. A detailed description of the new search space follows.
For any shelter \(a_i \in A\) define the horizontal wall \(\alpha_i^h\) and vertical wall \(\alpha_i^v\) to be segments in \(X_n\):

\[
\alpha_i^h = \{(a_i, q) \mid q \in \mathcal{H}(a_i)\}, \quad \alpha_i^v = \{(p, a_i) \mid p \in \mathcal{H}(a_i)\}.
\]

Similarly, each shelter \(b_i \in B\) induces the horizontal and vertical walls \(\beta_i^h\) and \(\beta_i^v\) respectively:

\[
\beta_i^h = \{(b_i, q) \mid q \in \mathcal{H}(b_i)\}, \quad \beta_i^v = \{(p, b_i) \mid p \in \mathcal{H}(b_i)\}.
\]

For instance, in Figure 2.5(a) the vertical segment between \((a_0, a_0)\) and \((a_0', a_0)\) represents the vertical wall \(\alpha^v_0\). Also the horizontal segment with wraparound between \((b_1, b_1)\) and \((b_1, b_1')\) represents the horizontal wall \(\beta_1^h\). Note that all walls are subsets of \(X_n\).

**Definition 2.6.** For a polygon \(P\) define the skeleton \(S \subset X_n\) to be

\[
S = D \cup A^h \cup B^h \cup A^v \cup B^v,
\]

where:

- \(D = \{(p, p) \mid p \in \partial P\}\) is the diagonal wall, or simply the diagonal.
- \(A^h = \bigcup_{k=0}^{m-1} \alpha_k^h, B^h = \bigcup_{k=0}^{m-1} \beta_k^h\), are the two sets of horizontal walls,
- \(A^v = \bigcup_{k=0}^{m-1} \alpha_k^v, B^v = \bigcup_{k=0}^{m-1} \beta_k^v\), are the two sets of vertical walls.

Let \(\alpha_i^h\) (respectively, \(\alpha_i^v\)) be a horizontal (respectively, vertical) wall. The tip of the wall \(\alpha_i^h\) (respectively, \(\alpha_i^v\)) is the configuration \((a_i, a_i')\) (respectively, \((a_i', a_i)\)). The tips of the walls \(\beta_i^h\) and \(\beta_i^v\) are defined similarly. Note that a tip need not belong to \(X_n\). However, as the next lemma shows, for every tip \(\tau\) (of a horizontal or vertical wall) there is a feasible neighborhood \(Q \subset X_n\) such that \(\tau\) belongs to the (topological) closure of \(Q\). For example, in the lower left corner of Figure 2.5(a) the black circle represents configuration \((a_0', a_0)\) which is the tip of the vertical wall \(\alpha_0^v\). The dotted rectangle represents a neighborhood \(Q\) lying in \(X_n\).

**Lemma 2.7.** Let \((\alpha_k', \alpha_k)\) be the tip of \(\alpha_k^v\). There exists a rectangle \(Q\) with left upper corner \((\alpha_k', u_1)\) and lower right corner \((v, u_2)\), where \(a_k \in (u_1, u_2)\) and \(a_k' \in (a_k, v)\), such that the interior of \(Q\) lies entirely in \(X_n\). A similar statement holds for the tips of the other three types of walls.

**Proof:** The proof follows directly from the definitions of the critical points \(a_k\) and \(a_k'\). Let \(a_k\) lie on the edge \(e_i = \overline{p_i p_{i+1}}\) of \(\partial P\); refer to Figure 2.3 in Section 2.3.1. Let \(u_1 = p_i\) and \(u_2 = p_{i+1}\). Also,
let \( v \in (a'_k, a_k) \) be sufficiently close to \( a'_k \), such that the entire edge \( e_i \) is visible from every point in the interval \( (a'_k, v] \). Clearly, the interior of \( Q \) lies in \( X_v \). □

The next lemma establishes that all horizontal and vertical walls in the skeleton \( S \) touch the diagonal.

**Lemma 2.8.** Let \( \overline{v_1v_2} \) be a horizontal (respectively, vertical) segment, such that \( v_1, v_2 \in X_v \) and \( \overline{v_1v_2} \cap D = \emptyset \). If \( \overline{v_1v_2} \cap X_n \neq \emptyset \), then \( \overline{v_1v_2} \cap (A^v \cup B^v) \neq \emptyset \) (respectively, \( \overline{v_1v_2} \cap (A^h \cup B^h) \neq \emptyset \)).

**Proof:** The two cases being similar, we assume the hypotheses for a horizontal segment \( \overline{v_1v_2} \), where \( v_1 = (p, q_1), v_2 = (p, q_2) \), see Figure 2.4(a). By assumptions, there exists a point \( s \in (q_1, q_2) \), such that \( u = (p, s) \in X_n \). Then there exist configurations \( u_1 = (p, s_1) \) and \( u_2 = (p, s_2) \), such that \( \overline{u_1u_2} \) is the maximal horizontal subsegment of \( \overline{v_1v_2} \) which contains \( u \) and whose interior lies in \( X_n \). Since none of the points in \( (s_1, s_2) \) are visible from \( p \), there must exist a concave region \( C_k \) which obstructs the view from \( p \). Moreover, exactly one of the points \( s_1 \) and \( s_2 \) lies on \( C_k \) and is visible from \( p \), while the other point is not visible from \( p \) and does not lie on \( C_k \), see Figure 2.4 (b,c). We consider both cases:

- If \( s_2 \in C_k \) and \( (p, s_2) \in X_v \), then \( s = a_k \in (s_1, s_2) \) and \( p \in \mathcal{H}(a_k) \), see Figure 2.4(b). Thus \( \overline{u_1u_2} \cap A^v \neq \emptyset \).

- If \( s_1 \in C_k \) and \( (p, s_1) \in X_v \), then \( s = b_k \in (s_1, s_2) \) and \( p \in \mathcal{H}(b_k) \), see Figure 2.4(c). Thus \( \overline{u_1u_2} \cap B^v \neq \emptyset \).

It follows that \( \overline{v_1v_2} \cap (A^v \cup B^v) \neq \emptyset \), and since \( \overline{v_1v_2} \subset \overline{v_1v_2} \), this completes the proof of the lemma. □

Lemma 2.7 and Lemma 2.8 illustrate the intuition behind the term “skeleton”: all horizontal and vertical walls in \( S \) span from the diagonal to a horizontal or vertical extrema (tips) of \( X_n \). Thus, in a sense, these walls form a supporting frame for \( X_n \). For example, Figure 2.5(a), shows the VOD together with the skeleton \( S \) for the polygon in Figure 2.1.
Next we show that if we relax somewhat the restrictions that the regions \( X_n \) and \( X_d \) impose on legal paths, we can build a discrete representation of \( X \), which, as we shall see, will allow us to find a solution in time \( O(m^2 + m \log n + n) \). The idea is to replace the regions \( X_n \) and \( X_d \) by the skeleton \( S \). The new search space is called \textbf{skeletal obstruction diagram (SOD)} and is illustrated in Figure 2.5(b). It represents the skeleton \( S \) of the same polygon, drawn over the square grid formed by plotting the critical points along the axes. Note that the lines which form the grid are added for reference only and are not part of the skeleton. The thick grey and black lines in the figure correspond to the vertical and horizontal walls of \( S \). For convenience we assume that the configuration \((a_0, a_0)\) is in the upper left corner of the grid. This results in a slight shift along the diagonal of the SOD (Figure 2.5(b)) relative to the VOD (Figure 2.5(a)).

\begin{definition}
A \textbf{relaxed path} in \( X \) is a continuous rectilinear path which does not cross the diagonal \( D \) or a horizontal wall and does not cross a vertical wall from left to right. A \textbf{relaxed path} which starts immediately above \( D \) and ends immediately below \( D \) is a \textbf{winning relaxed path}.
\end{definition}

Every legal path is also relaxed, thus every winning legal path is identical to a winning relaxed one. In the next section we show that a winning relaxed path can be transformed into a winning legal path.

\textbf{Figure 2.5} Skeleton (a) and square grid (b) of the polygon from Figure 2.1
2.3.3 Correspondence between winning legal paths and winning relaxed paths

In general, $X_v$ consists of a finite number of maximal connected regions. These are called conservative regions, because any path within a region preserves the left invariant. For example, there are three (two small and one large) conservative regions in Figure 2.5(a).

![Figure 2.6 Horizontal separation of conservative regions in the proof of Lemma 2.10.](image)

The following lemma states that conservative regions which do not overlap horizontally are separated by horizontal walls.

**Lemma 2.10.** Let $R$ be a conservative region, and let the configuration $(p_1, q_2) \in X$ be a local minimum of the boundary of $R$. Let $q_1, q_4 \in \partial P$ be such that $q_2 \in (q_1, q_4)$, and such that $c_1 = (p_1, q_1)$ and $c_2 = (p_1, q_4)$ are configurations in $X_n$ that do not lie on the boundary of $R$, see Figure 2.6. Let $Q$ be a sufficiently small rectangle (possibly a segment) with an upper edge $c_1 c_2$, such that all interior points of $Q$ lie in $X_n$. For any $p_3 \in \partial P$ and $q_3 \in (q_1, q_4)$, if $(p_3, q_3) \in X_v - Q$, then there is a horizontal wall which separates $p_1$ and $p_3$, i.e., there exists $p_2 \in (p_1, p_3)$ such that the horizontal segment between $(p_2, q_1)$ and $(p_2, q_4)$ is a part of a horizontal wall.

**Proof:** The proof relies on an observation of Guibas et al [15] that when the pursuer moves along the boundary, an interval of visible points appears/disappears exactly when a point on a bitangent line is crossed. Note that in the lemma, the existence of a local minimum $(p_1, q_2)$ for the conservative region $R$ means that when a pursuer is moving in the positive direction along the boundary, a visible interval neighboring point $q_2$ disappears exactly when the pursuer crosses point $p_1$. Therefore, $p_1$ must be a point where a bitangent intersects $\partial P$. Assume that the pursuer continues to move clockwise on the

---

5The conservative regions in our paper have a meaning close to the conservative cells defined by Guibas et al [15].

6These are the “green segments” defined in [15].
boundary past point \( p_1 \). How soon can point \( q_3 \) become visible? No point \( x \in (q_1, q_4) \), including \( q_3 \), is visible before the pursuer crosses another point \( p'_1 \) (shown in Figure 2.6) of the bitangent. Since there is a point \( p_2 \in A \cup B \) (the set of all shelters) such that \( p_2 \in (p_1, p'_1) \) and \( (q_1, q_4) \subset H(p_2) \), it follows that there is a corresponding horizontal wall which includes the segment between the configurations \( (p_2, q_1) \) and \( (p_2, q_4) \).

Since describing the exact shape of the conservative regions is rather complicated, we would like to enclose every region in a rectilinear boundary which will be easier to construct and explore.

**Definition 2.11.** Given a configuration \( c \in X - S \), the set of all configurations reachable from \( c \) without crossing the skeleton is called a tile (of \( c \)).

Note that every conservative region of \( X \) is contained in some tile. If a tile does not contain a conservative region, it is an empty tile. Otherwise it is nonempty.

**Lemma 2.12.** Every tile contains at most one conservative region. Thus, for a nonempty tile \( T \), if \( c_0, c_1 \in X_v \cap T \), then there is a path between \( c_0 \) and \( c_1 \) entirely within \( X_v \cap T \).

**Proof:** The claim is trivial for an empty tile, so assume that \( T \) is nonempty. Let \( c_0, c_1 \) be two configurations in \( X_v \cap T \). Since \( c_0, c_1 \in T \), there is a finite rectilinear path \( \pi \) within \( T \) from \( c_0 \) to \( c_1 \). We will prove by induction on \( k \), the number of segments in \( \pi \), that there is a rectilinear path from \( c_0 \) to \( c_1 \) entirely within \( X_v \cap T \), i.e., along feasible configurations.

The basis of the induction includes two cases: \( k = 1 \) and \( k = 2 \). The case for \( k = 1 \) follows immediately from Lemma 2.8. The case for \( k = 2 \) follows from Lemma 2.10. Assume that the statement is true for all paths of length less than \( k \), \( k \geq 3 \).

Consider a rectilinear path \( \pi = (c_0 = \pi_0, \pi_1, \ldots, \pi_k = c_1) \) within \( T \). The inductive step includes separate arguments for monotone\(^7\) and nonmonotone rectilinear paths.

(a) Assume that the path \( \pi \) is monotone. Without loss of generality, assume that the segments \( \overline{\pi_0 \pi_1}, \overline{\pi_1 \pi_2}, \text{and } \overline{\pi_2 \pi_3} \) are left-to-right, upward, and left-to-right, respectively, see Figure 2.7. (The rest of the cases are similar.) Let us start to translate the segment \( \overline{\pi_1 \pi_2} \) horizontally from left to right to the segment \( \overline{\pi_2' \pi_3'} \), where one of the following two events will occur:

- \( \pi_2' = \pi_3 \), see Figure 2.7(a).

Then the path \( (c_0 = \pi_0, \pi_1', \pi_2' = \pi_3, \ldots, \pi_k = c_1) \) consists of less than \( k \) segments so the induction hypothesis applies.

\(^7\)We define a "monotone rectilinear path" as a rectilinear path, whose projections on the horizontal and vertical axes are monotone.
Figure 2.7 Inductive hypothesis for a monotone path, Lemma 2.12

- \( \overrightarrow{\pi_1 \pi_2} \) will cross the tip of a horizontal or vertical wall, see Figure 2.7(b).

That is, there exists a point \( x \in \mathcal{X}_v \cap \overrightarrow{\pi_1 \pi_2} \). Then we can apply the inductive hypothesis to the paths \( (c_0 = \pi_0, \pi_1, x) \) and \( (x, \pi_2, \pi_3, \ldots, \pi_k = c_1) \).

We have shown that, if the path has \( k \) segments and is monotone, the statement holds.

Figure 2.8 Inductive hypothesis for a nonmonotone path, Lemma 2.12

(b) Next assume that the path \( \pi \) is not monotone. This implies, that there exists \( i \) such that the segments \( \overrightarrow{\pi_{i-1} \pi_i} \) and \( \overrightarrow{\pi_{i+1} \pi_{i+2}} \) are in opposite directions.

Without loss of generality, assume that \( \overrightarrow{\pi_{i-1} \pi_i} \), \( \overrightarrow{\pi_i \pi_{i+1}} \) and \( \overrightarrow{\pi_{i+1} \pi_{i+2}} \) are upward, left-to-right and downward, respectively, see Figure 2.8. (The rest of the cases are similar.) Let us start to
translate the segment \( \pi_i \pi_{i+1} \) downward to the segment \( \pi'_i \pi'_{i+1} \), where one of the following three events will occur:

- \( \pi'_{i+1} = \pi_{i+2} \), see Figure 2.8(a).

Then the path \( \langle \pi_0, \ldots, \pi_{i-1}, \pi'_i, \pi_{i+1}, \ldots, \pi_k = c_1 \rangle \) has less than \( k \) segments so the induction hypothesis applies.

- \( \pi'_i = \pi_{i-1} \).

Then the path \( \langle \pi_0, \ldots, \pi_{i-1} = \pi'_i, \pi_{i+1}, \pi_{i+2}, \ldots, \pi_k = c_1 \rangle \) has less than \( k \) segments so the induction hypothesis applies.

- \( \pi'_i \pi'_{i+1} \) will cross the tip of a vertical wall, see Figure 2.8(b).

That is, there exists a point \( x \in X \cap \pi'_i \pi'_{i+1} \). Then we can apply the inductive hypothesis to the paths \( \langle \pi_0, \ldots, \pi_{i-1}, \pi'_i, x \rangle \) and \( \langle x, \pi'_{i+1}, \pi_{i+2}, \ldots, \pi_k = c_1 \rangle \).

We have shown that, if the path has \( k \) segments and is not monotone, the statement holds.

Combining (a) and (b) completes the proof of the lemma. \( \square \)

Let \( T_0 \) and \( T_1 \) be two nonempty tiles, and let \( p, q_0, q_1 \) be points in \( \partial P \), such that \( q_0 \in (p, q_1) \) and the configurations \( d_0 = (p, q_0) \) and \( d_1 = (p, q_1) \) are interior points in \( T_0 \) and \( T_1 \), respectively, see Figure 2.9. If the line segment \( d_0d_1 \) crosses neither the diagonal nor any nonempty tiles other than \( T_0 \) and \( T_1 \), we say that \( T_0 \) is a left neighbor of \( T_1 \). The idea behind the relation "\( T_0 \) is a left neighbor of \( T_1 \)" is to represent a useful recontamination move of the pursuer from \( T_1 \) to \( T_0 \). Thus without loss of generality we can assume that \( T_0 \) (respectively, \( T_1 \)) does not border the diagonal from above (respectively, from below). We can make this assumption because if there is recontamination to \( T_0 \) and \( T_0 \) borders the diagonal from above, then we can ignore all the previous moves, since \( T_0 \) is reachable directly from the diagonal. Similarly, if there is recontamination from \( T_1 \) and \( T_1 \) borders the diagonal from below, then we can ignore the rest of the moves, since the diagonal is reachable directly from \( T_1 \).

The following lemma gives an important property of neighboring nonempty tiles.

**Lemma 2.13.** If \( T_0 \) and \( T_1 \) are two nonempty tiles such that \( T_0 \) is a left neighbor of \( T_1 \), then the corresponding conservative regions, \( R_0 \) and \( R_1 \), overlap horizontally, i.e., there exist \( c_0 = (p, q_0) \in R_0 \) and \( c_1 = (p, q_1) \in R_1 \). Moreover, the segment \( \overline{c_0c_1} \) does not cross any conservative regions other than \( R_0 \) and \( R_1 \).

**Proof:** Note that if one of the two tiles \( T_i \) is unbounded (on the torus), then the corresponding conservative region \( R_i \) is unbounded and it overlaps horizontally with the whole vertical axis; this
implies the claim. Therefore, it suffices to consider the case in which both tiles are bounded from above and below by horizontal walls. Let $L_0 = \langle l_0, \cdot \rangle$ (respectively, $L_1 = \langle l_1, \cdot \rangle$) be the lowest (respectively, highest) horizontal wall which borders $T_0$ (respectively, $T_1$). Similarly, let $m_0 = \langle m_0', m_0'' \rangle$ (respectively, $m_1 = \langle m_1', m_1'' \rangle$) be a global minimum (respectively, maximum) on the boundary of the conservative region $R_0$ (respectively, $R_1$), see Figure 2.9.

Since $T_0$ is a left neighbor of $T_1$, there exists a horizontal segment $d_0d_1$, $d_i \in T_i$, which crosses $T_0$, $T_1$ and possibly some empty tiles in between. Without loss of generality, assume that $d_0d_1$ lies between the walls $L_1$ and $L_0$. (The case in which $d_0d_1$ lies between the walls $L_0$ and $L_1$ is similar, we just have to redefine the walls $L_0$ and $L_1$ to be the highest and lowest horizontal walls of the corresponding regions.) If $m_0' \notin \langle l_1, l_0 \rangle$ or $m_1' \notin \langle l_1, l_0 \rangle$, we can apply Lemma 2.7 to show that there are nonempty tiles between $T_0$ and $T_1$. This would contradict the hypothesis that $T_0$ and $T_1$ are neighboring tiles. It follows that $m_0', m_1' \in \langle l_1, l_0 \rangle$.

Finally, we argue that the conservative regions $R_0$ and $R_1$ overlap horizontally. If they do not, then $m_0' \notin \langle l_1, m_1' \rangle$. We can apply Lemma 2.10 by choosing $m_0$ for the local minimum in the lemma, and any point in $X_\nu$ sufficiently close to $m_1$ for the configuration in $X_\nu$; the lemma implies that there is another horizontal wall which crosses the rectangle with left upper corner $m_0$ and lower right corner $m_1$. Thus a horizontal wall lies between the walls $L_1$ and $L_0$, so either $L_0$ is not a bordering wall of $T_0$, or $L_1$ is not a bordering wall of $T_1$. This gives a contradiction, implying that $m_0' \notin \langle l_1, m_1' \rangle$. It follows that $R_0$ and $R_1$ overlap horizontally, i.e., there exist $c_0 = \langle p, q_0 \rangle \in R_0$ and $c_1 = \langle p, q_1 \rangle \in R_1$, such that the horizontal segment $\overline{c_0c_1}$ does not cross any conservative regions other than $R_0$ and $R_1$. ■

Every relaxed path $\pi$ which starts and ends at a nonempty tile can be partitioned into a sequence...
of subpaths $\tau_0, \rho_1, \tau_1, \ldots, \rho_k, \tau_k$ such that every subpath $\tau_i$ lies completely inside a nonempty tile and every subpath $\rho_i$ starts in the tile of $\tau_{i-1}$, ends in the tile of $\tau_i$, and crosses no other nonempty tiles. On the other hand, a legal path $p$ can be partitioned into a concatenation of subpaths $t_0, r_1, t_1, \ldots, r_k, t_k$ where each $t_i$ is a legal path inside the same conservative region and $r_i$ is a horizontal path which starts from the conservative region of $t_{i-1}$, ends in the conservative region of $t_i$, and does not cross other conservative regions. Intuitively, a path $t_i$ corresponds to a sequence of the four reversible atomic moves of the pursuer as defined in section 2.2.3.3. A horizontal segment $r_i$ (always right-to-left) corresponds to a recontamination move: stationary pursuer, rotating the flashlight counterclockwise across an interval of invisible points.

We will show that we can transform every relaxed path $\pi$ into a legal path $p$, by replacing every subpath $\tau_i$ and $\rho_i$ with legal subpaths $t_i$ and $r_i$. A formal description follows. For convenience, we use $s(\pi)$ and $f(\pi)$ do denote the start and the end of a path $\pi$ (same for path $p$).

**Proposition 2.14.** Every relaxed path $\pi = (\tau_0, \rho_1, \tau_1, \ldots, \rho_k, \tau_k)$ with $s(\pi), f(\pi) \in X_v$ can be transformed into a legal path $p = (t_0, r_1, t_1, \ldots, r_k, t_k)$, where $\tau_i, \rho_i, t_i$ and $r_i$ are defined as above and $s(\pi) = s(p)$ and $f(\pi) = f(p)$.

**Proof:** The proof is by induction on $k$, the number of times the path $\pi$ switches between nonempty tiles.

[Basis, $k = 0$.] If $\pi = \tau_0$, this implies that $\pi$ starts and ends in the same conservative region and never leaves its original tile. From Lemma 2.12 it follows that there exists a path $p = t_0$, such that
\[ s(p) = s(\pi), \quad f(p) = f(\pi), \quad \text{and} \quad t_0 \text{ lies entirely within } X_p. \] Thus \( p \) is a legal path.

[Hypothesis, \( k < n \)]. Assume that the proposition is true for all \( k < n \).

[Inductive step, \( k = n \)]. Let \( \pi = (\tau_0, \rho_1, \tau_1, \ldots, \rho_n, \tau_n) \) be as described in the proposition. We have to show that it can be transformed into a legal path \( p = (t_0, t_1, \ldots, t_n, t_\pi) \). Let \( T_0 \) and \( T_1 \) be the first two nonempty tiles that \( \pi \) visits, i.e., \( \tau_0 \subset T_0 \) and \( \tau_1 \subset T_1 \). From Lemma 2.13 it follows that there exists a pair of configurations \( c_0 \in X_\nu \cap T_0 \) and \( c_1 \in X_\nu \cap T_1 \) such that \( r_0 = c_0c_1 \) is a horizontal segment in \( X \), see Figure 2.10. Also, by the assumption, we have \( s(\tau_0) = s(\pi) \in T_0 \cap X_\nu \), and since \( c_0 \in T_0 \cap X_\nu \), from Lemma 2.12 it follows that there exists a legal path \( t_0 \subset T_0 \cap X_\nu \) with \( s(t_0) = s(\tau_0) \) and \( f(t_0) = c_0 \). Finally, there is a path \( r'_1 \subset T_1 \) such that \( s(r'_1) = c_1 \) and \( f(r'_1) = f(\tau_1) = s(\rho_2) \).

We apply the inductive hypothesis to the relaxed path \( r' = (r'_1, \rho_2, \tau_2, \ldots, \rho_n, \tau_n) \) to get the legal path \( p' = (t_1, r_2, t_2, \ldots, r_n, t_n) \). We concatenate \( (t_0, r_0) \) and \( p' \) into the legal path \( p = (t_0, r_0, t_1, r_2, t_2, \ldots, r_n, t_n) \) thus proving the inductive step.\)

We summarize the discussion of the section in the next theorem.

**Theorem 2.15.** (i) Every winning legal path is a winning relaxed path. (ii) Every winning relaxed path can be transformed into a winning legal path.

**Proof:** Part (i) follows immediately from the definitions of legal and relaxed path. The proof of (ii) is based on Proposition 2.14 and the fact that for every nonempty tile \( T \) which borders the diagonal, the corresponding conservative region \( R \) also borders the diagonal.

The theorem shows that clearing \( P \) is reducible to finding a relaxed path in its corresponding SOD.

We take up the latter problem in the next section.

### 2.4 Finding a Solution

#### 2.4.1 Finding a relaxed path in time \( O(m^2 + m \log n + n) \)

The grid in Figure 2.5(b) is useful to illustrate compactly the restrictions imposed by the skeleton. Intuitively, we can think about a winning relaxed path as a path in a maze with horizontal and (one-way) vertical walls. For clarity of exposition we will present the search for a winning relaxed path in a directed graph \( \mathcal{G}_P \) that represents (the restrictions on) the connectivity between neighboring squares of the grid. Let \( (r_0, r_1, \ldots, r_{4m-1}) \) be the sequence of all the critical points in \( A, A', B, \) and \( B' \), as defined in Section 2.3.1, such that \( r_0 = a_0 \) and the elements of the sequence are ordered in the positive direction along \( \partial P \).

\( V(\mathcal{G}_P) \), the set of vertices of the graph, consists of:
non-diagonal vertices, one for each square away from the diagonal; e.g., the square labeled with A in Figure 2.5(b),

starting vertices, one for each half-square (triangle) immediately above the diagonal; e.g., the half-square labeled with B in Figure 2.5(b), and

goal vertices, one for each half-square (triangle) immediately below the diagonal; e.g., the half-square labeled with C in Figure 2.5(b).

We define \( E(G_P) \), the edges in the graph, as the largest subset of \( V(G_P) \times V(G_P) \) satisfying the following constraints:

1. there can be edges only between neighboring squares or half-squares of the grid,
2. there is no edge across the diagonal,
3. there is no edge across a horizontal wall,
4. there is no edge from left to right across a vertical wall, and
5. there is no edge from right to left across a vertical wall \( \alpha^v \) (respectively, \( \beta^v \)) if there are no horizontal walls crossing \( \alpha^v \) (respectively, \( \beta^v \)) between the edge and the tip of \( \alpha^v \) (respectively, \( \beta^v \)).

Rule (e1) ensures that a path in the graph represents a continuous relaxed path. Rules (e2), (e3) and (e4) are required by the definition of a relaxed path. Finally, rule (e5) guarantees that a relaxed path avoids unnecessary recontamination. i.e., we disallow a recontamination move between two squares if there is a path between them which does not require recontamination. For example, there is no edge from the square labeled E to the square labeled D in Figure 2.5(b).

Finding a winning relaxed path is equivalent to finding a path in the graph \( G_P \) from a starting to a goal vertex. Finding a path in \( G_P \) can be done by a breadth-first search and it takes time linear in the size of the graph.

**Theorem 2.16.** There is an algorithm that, given a simple polygon \( P \) with \( n \) edges and \( m \) concave regions, decides whether \( P \) can be cleared by a 1-searcher, and if so, outputs a winning relaxed path in time \( O(n + m \log n + m^2) \).

**Proof:** An outline of an algorithm which, given a simple polygon, finds a winning relaxed path if one exists, is presented in Figure 2.11. The correctness of the algorithm follows immediately from Theorem 2.15 and the definition of the graph \( G_P \).
FIND_RELAXED_PATH()
1 Input a polygon as a sequence of points \((p_0, p_1, \ldots, p_{m-1})\).
2 Determine the shelter points \(\{a_0, \ldots, a_{m-1}\}\) and \(\{b_0, \ldots, b_{m-1}\}\).
3 Use ray shooting to find the threshold points \(a'_i, b'_i\) for \(i \in \mathbb{Z}_m\).
4 Sort the critical points into a sequence \((r_0, r_1, \ldots, r_{m-1})\).
5 Construct the graph \(G_P\).
6 Using BFS on \(G_P\) find a path from a starting to a goal vertex.
7 Output the path or a message if such a path does not exist.

Figure 2.11 Outline of the algorithm for finding a winning relaxed path

We have to show that the algorithm runs in time \(O(n + m \log n + m^2)\). Steps (1) and (2) of the algorithm take time \(\Theta(n)\). To perform step (3) we apply from Lemma 2.5, which states that after a preprocessing of \(\Theta(n)\), we need to perform \(2m\) ray shooting queries, each of which can be performed in time \(O(\log n)\), thus the total time for step (3) is \(O(m \log n + n)\). Step (4) takes time \(O(m \log m)\). Steps (5) and (6) can be completed in time \(O(m^2)\) since the graph has \(m^2\) vertices and the outdegree of each vertex is bounded by 4.

Since the number of concave regions cannot exceed the number of edges in the polygon, the number of critical points is \(4m\), which is \(O(n)\). Thus the total running time is \(O(n + m \log n + m \log m + m^2) = O(n + m \log n + m^2)\). Note that in the worst case, when \(m = \Theta(n)\), we have a \(O(n^2)\) algorithm, but when \(m = O(\sqrt{n})\), we have a linear time algorithm.

2.4.2 Transforming a relaxed path into a legal path

Note that the search in \(G_P\) will find a relaxed, but not necessarily legal, path. In time \(O(m^2 + m \log n + n)\) we can transform a winning relaxed path \(\tau\) into a winning legal path containing \(O(m^2)\) atomic instructions. We do this by a procedure similar to a sequence of iterative applications of the induction step in Proposition 2.14. The procedure can be divided into two stages:

- First we find the places where recontamination occurs in the relaxed path, i.e., where \(\tau\) crosses a vertical wall from right to left. For each recontamination in \(\tau\) we have to identify a pair of configurations \(c_0, c_1 \in X_\tau\) representing the corresponding atomic recontamination move from \(c_0\) to \(c_1\), see Lemma 2.13. Such a pair can be found in time \(O(n_0 + n_1 + \log n + m)\), where \(n_0\) and \(n_1\) are the number of vertices in the respective concave regions associated with the recontamination move. In order to compute the total time for finding all the atomic recontamination moves, we note that there can be at most two recontamination moves across a given concave region (one per each hiding place in a concave region) and thus a total of at most \(O(m)\) recontamination moves.
Since each vertex of $P$ is counted at most twice, the total time is $O(n) + O(m) \cdot O(\log n + m) = O(n + m \log n + m^2)$.

- Once we have broken the relaxed path $r$ into subpaths $r_1, \ldots, r_k$, which do not contain any recontamination moves, we have to transform each subpath $r_i$ into a sequence of the other four moves. We do this by partitioning each $r_i$ into maximal monotone subpaths $r_{i,1}, \ldots, r_{i,\ell}$. Then each monotone subpath $r_{i,j}$ can be divided into a sequence of the four atomic moves by iteratively finding the tip of a wall closest to $s(r_{i,j})$ (vertically or horizontally in the direction of $f(r_{i,j})$).

Every single step described above takes time linear in the number of segments of the corresponding subpath, so transforming all relaxed subpaths $r_i$ into sequences of atomic moves can be done in time $O(m^2)$.

Thus the total time for outputting a description of a winning path consisting of $O(m^2)$ atomic instructions is $O(n + m \log n + m^2)$. Note that in the worst case, when $m = \Theta(n)$ this is equivalent to $\Theta(n^2)$. However, in the cases in which $m = o(n)$ (e.g., in the case of curved planar environments [25]), our algorithm is considerably faster than $O(n^2)$.

On the other hand, if we need a more detailed description of the sequence of vertices visited by the pursuer or illuminated by the flashlight, this can be done in time $O(n^2)$ by computing in $O(n)$ time the visibility polygon for all the vertices and critical points as identified by the algorithm.

### 2.5 Implementation and Comparison with Previous Work

The single-pursuer algorithm was implemented using GNU C++ and the Library of Efficient Data Types and Algorithms (LEDA), and experiments were performed on a Pentium III 500Mhz PC running Linux. The algorithm was determined to be efficient enough for practical use in real environments. Figure 2.12 provides a sequence of snapshots for a solution. Note that the position of the pursuer on the boundary is designated with a small white circle.

The polygon in Figure 2.12 is a simplified version of one from [49] and it is of interest beyond a mere illustration of the algorithm implementation. It represents a polygon which requires a recontamination in order to be searched successfully. The recontamination happens between frames (d) and (e): the endpoint of the flashlight jumps over an interval of $\partial P$, thus some of the area which was already cleared is contaminated again after the pursuer rotates her flashlight counterclockwise.

It is important to note that a schedule for a 1-searcher which contains a recontamination move cannot be simulated by a corresponding search schedule with two guards [21, 17]. The reason is that
the two guards cannot maintain visibility and stay on \( \partial P \) while attempting to simulate a jump of the flashlight. Therefore, there are polygons that can be cleared by a 1-searcher but cannot be cleared by two guards. Since every schedule for two guards is a schedule for a 1-searcher as well, it follows that the set of polygons that can be cleared by a 1-searcher is a strict superset of the polygons that can be cleared by two guards. Also, note that while the polygon in Figure 2.12 can be cleared by a chain of three guards using an algorithm by Efrat et al [12] (this is generalization of the two guards problem to a chain of \( k \) guards), this is not equivalent to finding a solution for a 1-searcher.

Similarly, it is not hard to show that every point in the polygon in Figure 2.12 is contaminated at some time during any successful 1-searcher schedule. Thus there is no point \( d \) such that the room \((P,d)\) can be cleared by a 1-searcher as described in the \( O(n^2) \) algorithm of Lee et al [30].

### 2.6 Conclusion

In this paper we have presented an \( O(m^2 + m \log n + n) \) algorithm which, given a simple polygon with \( n \) edges and \( m \) concave regions, decides whether the polygon can be cleared by a 1-searcher and if so, outputs a search schedule. The algorithm is a nontrivial generalization of the two-guard search algorithm and solves a (rather longstanding) problem left open in [49]. The most interesting extension
of the result in the current paper would be an algorithm which, given a polygon and an integer $k$, decides whether the polygon can be cleared by $k$ 1-searchers, and ideally, returns a search schedule. Note that the problem does not impose any restrictions on the mutual visibility between the 1-searchers which, we showed in Section 2.5, allows clearing of a strictly greater set of polygons as compared to a chain of $k + 1$ guards. Finally, while the proposed problem for $k$ 1-searchers is NP-hard for polygonal regions which contain holes [54], little is known about the complexity of the problem for simple polygons.
3 CLEARING A POLYGON WITH TWO 1-SEARCHERS

A paper submitted to the International Journal of Computational Geometry and Applications
Steven M. LaValle, Borislav H. Simov and Giora Slutzki

Abstract

We present an algorithm for a pair of pursuers, each with one flashlight, searching for an unpredictable, moving target in a 2D environment (simple polygon). Given a polygon with \( n \) edges, the algorithm decides in time \( O(n^4) \) whether it can be cleared by the pursuers and if so, constructs a search schedule. The pursuers are allowed to move on the boundary and in the interior of the polygon. They are not required to maintain mutual visibility throughout the pursuit.

3.1 Introduction

Consider the following scenario: in a (dark, doorless) polygonal region there are three moving objects (represented as points). Two of them, called the pursuers (also known as 1-searchers), have the task to find the third, called the evader. The evader can move arbitrarily fast, and his movements are unpredictable by the pursuers. Each pursuer is equipped with a flashlight and can see the evader only along the illuminated line segment emitted by the flashlight. The pursuers have perfect knowledge about each other's location. They plan their moves in cooperation and are not required to maintain mutual visibility at all times. The pursuers win if they illuminate the evader with a flashlight. If there is a movement strategy of the pursuers whereby they win regardless of the strategy employed by the evader, we say that the polygon can be cleared by two 1-searchers. In this paper we present an algorithm which, given a polygon with \( n \) edges, and \( m \) concave regions decides in time \( O(n^2 + nm^2 + m^4) \) whether it can be cleared by the two 1-searchers, and if so, constructs a search schedule.

---

1Reprinted with permission of World Scientific Publishing Co.
2An earlier version of this paper was presented at the 2002 IEEE International Conference on Robotics and Automation [43].
The scenario above is a typical problem in pursuit-evasion, a field of recent interest in both robotics and computational geometry. The basic task in pursuit-evasion is to compute motion strategies for one or more pursuers to guarantee that unpredictable evaders will be detected. A key difficulty which makes the problem more challenging than basic exploration is that the evaders can sneak back to places already explored. Efficient algorithms that compute these strategies can be embedded in a variety of robotics systems to locate other robots and people. They can aid mobile surveillance systems that detect intruders using sonars, lasers, or cameras. Mobile robots can be used by special forces in high-risk military operations to systematically search a building in enemy territory before it is declared safe for entry.

The rest of the paper is organized as follows. Section 3.1.1 gives a brief overview and history of the visibility-based pursuit-evasion problem. We introduce the notation and provide some basic definitions in Section 3.1.2. Two formal models of the pursuit as a search in a continuous information space are presented in Section 3.1.3 and Section 3.2. A discrete representation of the pursuit as a search in a finite graph is presented in Section 3.3. In Section 3.4 we present an algorithm which, given a polygon with \( n \) edges and \( m \) concave regions, decides in \( O(n^2 + nm^2 + m^4) \) time whether the polygon can be cleared by two 1-searchers and if so, constructs a winning search schedule. We note that the representations in Sections 3.1.3 and 3.2 are given only as a conceptual framework and are not directly used by the algorithm in Section 3.4. Section 3.5 concludes the paper with a summary and directions for future research.

### 3.1.1 Related work

Pursuit-evasion in the plane was introduced by Suzuki and Yamashita [49]. They considered a single pursuer looking for an evader inside a simple polygon. They defined different kinds of pursuers depending on the number of flashlights that the pursuer is equipped with, e.g., a 1-searcher has one flashlight, a \( k \)-searcher has \( k \) flashlights, and an \( \infty \)-searcher has 360°-vision. This naturally defines a pursuit-evasion problem for each class of searchers. Independently of [49], Icking and Klein [21] defined the "two guard walkability problem", which is a search problem for two guards whose starting and goal position are given, and who move on the boundary of a polygon while maintaining mutual visibility. A solution to the two guards problem was provided in [21], followed by improvements in [17, 52]. While [49, 10, 30, 31, 27] presented polynomial solutions for deciding searchability of special classes of polygons, the general case single pursuer problem was open for quite a while.

Recently, the authors provided a \( O(n^3) \) solution for a single 1-searcher in a polygon [42], a result
which they later improved to $O(n + m \log n + m^2)$ [28]. Park et al [35] presented a polynomial solution for the case of a single 2-searcher and proved that adding more flashlights to a single pursuer does not increase the class of the polygons she can clear. Note that the set of polygons that can be cleared by a single 2-searcher is a proper subset of the set of polygons that can be cleared by two 1-searchers. Figure 3.4(a) presents an example of a polygon which can be cleared by two 1-searchers, yet cannot be cleared by a single 2-searcher.

Guibas et al [15] extended the pursuit-evasion problem to one in which multiple pursuers collaborate in order to clear a polygonal region. They showed that determining the minimal number of pursuers needed to clear a polygonal region with holes allowed is an NP-hard problem. It is not known whether the same problem defined over simple polygons without holes is also NP-hard.

Efrat et al [12] considered pursuit-evasion by a chain of $k$ guards as a generalization of the search with two guards. Their pursuit is subject to the restriction that the first and the $k$-th guards always move on the boundary while guard $i$, $1 < i < k$ moves in the interior of the polygon and maintains visibility with her neighbors, guards $i - 1$ and $i + 1$. Efrat et al [12] gave a polynomial algorithm for the $k$ guards problem. Note that the pursuit with two 1-searchers is not a special case of the $k$ guards pursuit since (i) the 1-searchers are not required to maintain visibility all the time, and (ii) for each 1-searcher, the endpoint of the ray of light emitted by her flashlight does not have to move continuously along the boundary of the polygon.

Suzuki et al [48] provided a polynomial-time solution for a version of the pursuit in which a single $\infty$-searcher is restricted to the boundary of the polygonal region. Vidal et al [53] addressed a version of the pursuit-evasion problem that uses probabilistic models.

3.1.2 Notation and preliminaries

In the rest of the paper "pursuer" will be synonymous to a 1-searcher, unless otherwise specified. Also, all polygons are assumed to be simple. The boundary of a polygon $P$ is denoted by $\partial P$ and we assume that $\partial P \subseteq P$ and that $\partial P$ is oriented in the clockwise (also called positive) direction. For two distinct points $a, c \in \partial P$, we write $\partial P(a, c)$ to denote the open interval of all points $b \in \partial P$ such that when starting after $a$ in positive direction along $\partial P$, $b$ is reached before $c$. We write $a < b < c$, if $b \in \partial P(a, c)$, and also use the notation $\partial P[a, c]$, $\partial P[a, c)$ and $\partial P(a, c]$ for the closed and half-closed intervals on $\partial P$.

Let $p_0, p_1, \ldots, p_{n-1}$ denote the vertices on $P$ ordered in the positive direction. The edges of $\partial P$ are $e_0, e_1, \ldots, e_{n-1}$, where edge $e_i$ has endpoints $p_i$ and $p_{i+1}$, and the indices are computed modulo $n$. 
e.g., \( p_n = p_0 \). Vertex \( p_i \in \partial P \) is a reflex vertex if the angle formed by incident edges \( e_{i-1} \) and \( e_i \), in the interior of \( P \), is greater than 180° (i.e., points \( p_{i-1}, p_i \), and \( p_{i+1} \), form a left turn). Otherwise, \( p_i \) is a non-reflex vertex.

We use a standard definition of visibility. For points \( c, d \in P \) we say that \( d \) is visible from \( c \), if every interior point of the line segment \( \overline{cd} \) lies in \( P - \partial P \). Obviously, if one point is visible from another, then the two are mutually visible. Note that no two points on the same edge of \( P \) are mutually visible.

![Figure 3.1 The two kinds of gap edges: (a) left, (b) right.](image)

Let \( r \in P \), \( q^1, q^2 \in \partial P \) be three colinear points, see Figure 3.1(a). We say that the pair \( (q^1, q^2) \) forms a left gap edge relative to \( r \) if:

- \( q^1 \) is visible from \( r \),
- no point in \( (q^1, q^2) \) is visible from \( r \),
- every open interval which contains \( q^2 \) also contains a point visible from \( r \).

Note that we do not require that \( q^1 \) and \( q^2 \) are mutually visible.

Similarly, see Figure 3.1(b), the pair \( (q^2, q^1) \) forms a right gap edge relative to \( r \) if:

- \( q^2 \) is visible from \( r \),
- no point in \( [q^1, q^2] \) is visible from \( r \),
- every open interval which contains \( q^1 \) also contains a point visible from \( r \).

Clearly, the order of \( r, q^1 \) and \( q^2 \) is important in the definition of a gap edge. However, from now on we will be a bit more casual about the order, if it can be inferred from the context.
Consider \( q^0, q^1, q^2, q^3 \in \partial P \) ordered in positive direction and \( p \in P \). Suppose that \( r, q^1 \) and \( q^2 \) form a (left or right) gap edge and \( q^0 \) and \( q^3 \) are sufficiently close to \( q^1 \) and \( q^2 \) respectively, so that all the points in \( (q^0, q^1) \cup (q^2, q^3) \) are visible from \( r \), see Figure 3.1, (a) or (b). Suppose a pursuer is located at \( r \) and her \textbf{lightpoint} (the point of the boundary illuminated by her flashlight) is at \( q^0 \). If the pursuer rotates the flashlight clockwise, the lightpoint moves continuously over \( \partial P \) before it reaches \( q^1 \). At that moment the lightpoint jumps from \( q^1 \) to \( q^2 \). After \( q^2 \) the lightpoint moves continuously to \( q^3 \). We call this a \textbf{lightpoint jump} from \( (r, q^1) \) to \( (r, q^2) \). The reverse move, from \( (r, q^2) \) to \( (r, q^1) \), is also a lightpoint jump. We note that the lightpoint jumps are the only possible discontinuities in the location of the lightpoint.

\[\text{3.1.3 Semantics of the pursuit}\]

For \( i = 0, 1 \) and mutually visible points \( r_i \in P, q_i \in \partial P \), we say that pursuer \( i \) is in \textbf{configuration} \( (r_i, q_i) \) if the pursuer is located at \( r_i \) and her lightpoint is at \( q_i \). Naturally then, the positions of both pursuers can be encoded as a pair of configurations, \((r_0, q_0)\) and \((r_1, q_1)\).

The segments \( r_0 q_0 \) and \( r_1 q_1 \) partition \( P \) into a number of connected components. We call each component a \textbf{contamination region}, consisting of all points of \( P \) which are connected by a path within \( P \) not crossed by a ray of light. \(^3\) For example, in Figure 3.2(a) there are two contamination regions, \( C_0 \) and \( C_1 \). For stationary pursuers, an evader can move undetected to every point within a contamination region, hence all the points in a region have the same contamination status. From now on we will simply refer to the region itself as being \textbf{contaminated} if it may contain the evader, or as \textbf{clear} otherwise.

We can now define the motion of the pursuers as a trajectory in the space \( P \times \partial P \times P \times \partial P \), parametrized over time. Without loss of generality we can assume that the pursuit starts at time \( t = 0 \) with the assumption that the polygon is contaminated. For \( i = 0, 1 \), define the functions \( r_i : [0, \infty) \to P \), and \( q_i : [0, \infty) \to \partial P \), such that at time \( t \) the \( i \)-th pursuer is in configuration \( (r_i(t), q_i(t)) \). For \( i = 0, 1 \), the functions \( r_i \) and \( q_i \) are subject to some additional constraints stemming from the semantics of the pursuit:

- At any time \( t \in [0, \infty) \), \( r_i(t) \) and \( q_i(t) \) are mutually visible.
- The position of the pursuer, \( r_i(t) \), is a continuous function.

\(^3\)To avoid tedious technicalities, we exclude from the partition the points lying on the segments \( r_0 q_0 \) and \( r_1 q_1 \).
• The lightpoint, \( q_i(t) \), is a piecewise continuous function with discontinuities corresponding to the lightpoint jumps defined earlier.

We define a schedule to be a 4-tuple of functions, \((r_0(t), q_0(t), r_1(t), q_1(t))\), satisfying the above constraints. Starting with a contaminated polygon, a given schedule implicitly determines the contamination status of the polygon. That is, at any time \( t \geq 0 \), there is a well-defined set of points in \( P \) which are clear. For a fixed polygon, if a schedule starts at time \( t = 0 \) with a contaminated polygon and at some time \( t = T \), the polygon is clear, the schedule is called a winning schedule.

**Definition 3.1.** A polygon \( P \) can be cleared by two 1-searchers if there exists a winning schedule for \( P \).

### 3.2 Canonical Pursuit

In Section 3.1.2 we defined a model of the pursuit, called a schedule. Its main advantage is its simplicity, i.e., it explicitly represents the motions of the pursuers. However, it has certain shortcomings. First, a snapshot of the pursuit at a fixed moment of time \( t \), or equivalently, the values \((r_0(t), q_0(t), r_1(t), q_1(t))\) alone, do not give us information about the complete status of the pursuit. We need the past trajectory of the four functions to determine the clear and contaminated areas of the polygon.

In this section we consider an equivalent model of pursuit, in which the two 1-searchers stay on the boundary most of the time. While the schedule defined in Section 3.1.3 corresponds to a trajectory in \((P \times \partial P)^2\), the schedule we define in Section 3.2.2 is similar to a trajectory in a smaller space, \((\partial P)^4\).

In addition, every snapshot of the new representation will give us the complete status of the pursuit: it contains both the pursuers' positions and the contaminated regions of the polygon.

#### 3.2.1 Canonical configurations

We first show how, without causing contamination, we can map an arbitrary configuration to one in which the pursuer is on the boundary. For \( i = 0,1 \), suppose pursuer \( i \) is at point \( r_i \in P - \partial P \), directing the beam at point \( q_i \in \partial P \), see Figure 3.2(b). Shoot a ray starting from \( r_i \) in the direction opposite of \( q_i \) and let \( p_i \in \partial P \) be the first boundary point hit by the ray. We say that \((p_i, q_i)\) is the canonical configuration corresponding to the configuration \((r_i, q_i)\). If \( r_i \in \partial P \), then \((r_i, q_i)\) maps to itself.

An interesting property of the mapping is that the contamination regions induced by the segments \( \overline{p_i q_i} \) are a refinement of the regions induced by the segments \( \overline{r_i q_i} \). For example, consider Figure 3.2(b) and assume the pursuers are in configurations of \((r_i, q_i), i = 0,1 \). If we define the area \( C_i = C_i^1 \cup C_i^2 \cup C_i^3 \),
then \( r_0q_0 \) and \( r_1q_1 \) partition \( P \) into just two contamination regions, \( C_0 \) and \( C_1 \). On the other hand, for the corresponding canonical configurations, \( (p_i, q_i), i = 0, 1 \), there are four contamination regions: \( C_0, C_0^i, C_1^i \) and \( C_1^i \). The contamination regions \( C_0, C_0^i, C_1^i \) and \( C_1^i \) are a refinement of the regions \( C_0 \) and \( C_1 \), which leads us to the following observation.

**Observation 3.2.** At any single moment, we can replace a pair of pursuers located in the interior of \( P \) with the corresponding canonical pair without causing additional contamination.

We showed how to map an arbitrary configuration into a corresponding canonical one. A natural continuation would be to use the same transformation to map an arbitrary trajectory of a pursuer from \( P \times \partial P \) onto \( \partial P \times \partial P \). Most of the time the continuous motion of \( r_i(t) \in P \) translates into a continuous motion of \( p_i(t) \in \partial P \), see Figure 3.3(a). However, there are exceptions: \( p_i \) may not be a continuous function on \( \partial P \), as seen in the example in Figure 3.3(b). For simplicity, assume that pursuer 0 is stationary at point \( r_0 \) with lightpoint \( q_0 \). Pursuer 1 moves over a continuous path from \( r_1^i \) to \( r_1^f \), which is projected as a piecewise continuous path on \( \partial P \) from \( p_1^i \) to \( p_1^f \). The move of the first pursuer can be divided into two parts. The first part, from \( r_1^i \) to \( r_1^2 \), is projected into the continuous path from \( p_1^i \) to \( p_1^2 \). The second part, from and excluding \( r_1^2 \) to \( r_1^f \), is projected into the continuous path from and excluding \( p_1^2 \) to \( p_1^f \). Note that the jump from \( p_1^2 \) to \( p_1^f \) represents a discontinuity in \( p_1(t) \), the projection of \( r_1(t) \) on \( \partial P \), and this jump cannot be simulated by a pursuer moving solely over the boundary.

The solution is to allow the pursuer to move along the segment \( \overline{p_1^2p_1^f} \). Thus the continuous motion from \( r_1^i \) to \( r_1^f \) can be represented as a continuous motion from \( p_1^i \) to \( p_1^2 \) along \( \partial P \), followed by a continuous motion from \( p_1^2 \) to \( p_1^f \) inside \( P \), followed by a continuous motion from \( p_1^f \) to \( p_1^i \) along \( \partial P \).

We call the move from \( (p_1^i, q_1) \) to \( (p_1^f, q_1) \) a pursuer jump. Just like in the case of the lightpoint
jump, the reverse move, from \((p_1^2, q_1)\) to \((p_1^2, q_1)\), is also considered a pursuer jump. Note that in reality the physical location of the pursuer is still continuous but at least for a moment the pursuer had left the boundary. From now on this will be the only circumstance in which the pursuers will leave \(\partial P\).

### 3.2.2 Canonical schedule

Our next goal is to define a pursuer schedule based on canonical configurations. Recall that we defined canonical configurations as \textit{directed} line segments of mutually visible points on the boundary. However a lot of the discussion in the rest of the paper will deal with visibility, which is a symmetric relation. Therefore, we do not base the new schedules solely on the original definition of canonical configurations, but we also incorporate an alternative interpretation of canonical configurations as \textit{undirected} line segments.

Let \(x_0, y_0, x_1, y_1 \in \partial P\), such that \(x_0 < x_1 < y_1\), and for \(i \in \{0, 1\}\), \(x_i\) and \(y_i\) are mutually visible. Let the \textit{order} (of \(y_0\)), \(k\), denote the number of points from \(\{x_1, y_1\}\) between \(x_0\) and \(y_0\), or:

\[
k = |\{x_1, y_1\} \cap \partial P(x_0, y_0)| = \begin{cases} 
0, & \text{if } x_0 < y_0 < x_1 < y_1 \\
1, & \text{if } x_0 < x_1 < y_0 < y_1 \\
2, & \text{if } x_0 < x_1 < y_1 < y_0
\end{cases}
\]

We refer to \((x_0, y_0, x_1, y_1, k)\) as a \textit{visibility tuple}. Due to the circularity of \(\partial P\), each tuple has three more equivalent tuples: \((y_0, x_1, y_1, x_0, k')\), \((x_1, y_1, x_0, y_0, k'')\), and \((y_1, x_0, y_0, x_1, k''')\). The set of all visibility tuples is defined as the \textit{visibility space}, \(\mathcal{I}_v\).
Suppose \((p_0, q_0)\) and \((p_1, q_1)\) are two configurations with the property that the light segments \(p_0q_0\) and \(p_1q_1\) are a permutation of the light segments \(x_0y_0\) and \(x_1y_1\). The visibility tuple contains exactly the same visibility information as the pair of configurations \((p_0, q_0), (p_1, q_1)\). At the same time, the visibility tuple has the advantage that the order of the points \(x_0, y_0, x_1, y_1\) is completely determined by their position in the tuple and \(k\).

Let \(\mathcal{X}_v\) be the set of all pairs of mutually visible points \((x, y) \in \partial P \times \partial P\) such that \(x < y\). The set \(\mathcal{X}_v\) is part of the visibility obstruction diagram (VOD) defined in [42], as a graphical representation of the visibility relation between pairs of points from \(\partial P\). The only difference is that, since visibility is a symmetric relation, without loss of generality, in this paper we have restricted \(\mathcal{X}_v\) to points above the diagonal. For example consider, the polygon in Figure 3.4(a). The corresponding VOD \(\mathcal{X}_v\) is shown as the set of points in the white regions in Figure 3.4(b). If for the moment we disregard \(k\), then we can think about a visibility tuple merely as two points in \(\mathcal{X}_v\). A visibility tuple \((8, 27, 9, 26, 2)\) denotes that the two light segments are \(8, 27\) and \(9, 26\) but does not determine the position of the pursuers, i.e., pursuer 0 can be either at point 8 or at point 27 of the polygon in Figure 3.4(a). Points \((8, 27)\) and \((9, 26)\) from the set \(\mathcal{X}_v\) shown in Figure 3.4(b) represent the positions of the two light segments \(8, 27\) and \(9, 26\).

We have replaced a pair of configurations \((p_0, q_0)\) and \((p_1, q_1)\), i.e., two directed segments, with a visibility tuple \((x_0, y_0, x_1, y_1, k)\), which no longer stores the direction of the segments, or equivalently, the location of the pursuer. To record this additional information, we need one extra bit for each segment. For \(i \in \{0, 1\}\), define the direction bit \(d_i\) for pursuer \(i\), such that \(d_i \in \{0, 1\}\), and \(d_i = 0\) when \(x_i\) corresponds to a pursuer location, i.e., when \(x_i \in \{p_0, p_1\}\).

As we mentioned before, one drawback of the schedule defined in Section 3.1.2 is that it does not explicitly show the status of the contamination regions. We fix this as follows. Since the segments \(p_iq_i\), or equivalently, \(x_iy_i\), \(i = 0, 1\), divide the boundary into four regions, four bits will be sufficient to record the contamination status at a given moment of time. Let us number the intervals from 0 to 3 in positive direction, starting from the interval beginning at \(x_0\). For \(i = 0, 3\), define contamination bit \(b_i \in \{0, 1\}\), to be the contamination status of interval \(i\), with \(b_i = 0\) when the interval is clear. We refer to \((d_0, d_1, b_0, b_1, b_2, b_3)\) as a bits tuple. The set of all bits tuples is defined as the bits space, \(\mathcal{I}_o\).

Consider a fixed moment during a pursuit, and suppose that the two pursuers are in configurations \((p_i, q_i), i = 0, 1\). Let \(k, x_i, y_i, d_i, i = 0, 1, \text{ and } b_j, j = 0, 3\) be as defined above. Define the canonical
Figure 3.4  Visibility obstruction diagram (b) for the polygon in (a). The white regions in (b) correspond to the set $X_v$ of the mutually visible pairs of points in $\partial P$. 
information state for the pursuit to be the concatenation of the visibility and the bits tuples:

\[(x_0, y_0, x_1, y_1, k, d_0, d_1, b_0, b_1, b_2, b_3)\]

Note that the information state gives us explicitly a full snapshot of the pursuit at that moment. Consider again the polygon in Figure 3.4(a) and its corresponding VOD, Figure 3.4(b). A canonical information state \((8, 27, 9, 26, 2, 1, 0, 0, 1, 0, 1)\) denotes that the two pursuers are at points 27 and 9, illuminating points 8 and 26 respectively. The intervals \(\partial P(9, 26)\) and \(\partial P(27, 8)\) are contaminated. The intervals \(\partial P(8, 9)\) and \(\partial P(26, 27)\) correspond to the same region which is clear.

A canonical information state is a start if all the contamination bits are 1, corresponding to a contaminated polygon. A canonical information state is a goal if the contamination bits are 0, i.e., the polygon is clear. The set of all canonical information states is the canonical information space, \(I\).

Observe that \(I = I_v \times I_b\), i.e., \(I\) is a product of the infinite space \(I_v\) and the finite space \(I_b\).

After the definition of information states and the introduction of the pursuer jump we can define a canonical schedule to be a piecewise continuous trajectory in the corresponding canonical information space, \(I\), parametrized over time. It is quite similar to a schedule with the restriction that the pursuers move almost always on the boundary and enter the interior only during the pursuer jumps. We can view a canonical schedule as a function \(I : [0, \infty) \rightarrow I\), where \(I(t) = (x_0, y_0, x_1, y_1, k, d_0, d_1, b_0, b_1, b_2, b_3)\) encodes the information state at time \(t\). Just like in previous definition, a canonical schedule has to also satisfy some conditions derived from the semantics of the pursuit:

- The functions \(x_i(t)\), \(y_i(t)\) are piecewise continuous in \(\partial P\), with discontinuities corresponding to jumps, pursuer or lightpoint. Note that the direction bit \(d_i\) is required to determine the type of a jump for \(x_i(t)\) or \(y_i(t)\).
- At time 0 the canonical schedule is in a starting canonical information state.
- The direction bits \(d_i\), the contamination bits \(b_j\) and the order \(k\) change only during a jump or during a change in the relative order of \(x_0, y_0, x_1, y_1\). (We provide a detailed explanation of the types of changes to the information states in Section 3.2.3.)

A canonical schedule which at some time is in a goal state is called a winning canonical schedule.

In the rest of this section we show that winning schedules are equivalent to the original schedules. That is, we do not reduce the power of the pursuers by restricting them to moving on the boundary most of the time.
Lemma 3.3. A polygon can be cleared by two 1-searchers, if and only if there exists a winning canonical schedule.

Proof. First, consider the reverse direction. Note that if we ignore the details of the representation, every canonical schedule is just a more restricted version of a schedule. So if there exists a winning canonical schedule, then there also exists a winning schedule, therefore the polygon can be cleared by two 1-searchers.

For the forward direction, assume that the polygon can be cleared, so there exists a winning schedule. Consider its corresponding canonical schedule. From Observation 3.2, at any time, the contaminated regions in the canonical schedule are a subset of the contaminated regions in the original winning schedule. It follows that the canonical schedule is a winning one. •

3.2.3 Changes to the canonical information states

In this section we discuss the ways contamination bits change. This will allow us to define a finite set of elementary moves, so that a canonical schedule can be considered a sequence of these elementary moves.

From the definition of a canonical schedule, it follows that it is a piecewise continuous function in $I$. If we consider every element of the tuple $I(t)$ as a function of time, for most of the duration of the pursuit, it is a continuous function. Only at discrete moments of time, there are relative order changes or discontinuities in $x_0$, $y_0$, $x_1$, or $y_1$. These conditions trigger corresponding changes in the other elements of the tuple, the direction bits $d_i$, the contamination bits $b_j$ and the order $k$. We identify the different types of changes to the information state and define those as elementary moves. For a given schedule, let $0 < t_1 < t_2 < \ldots$ be the points of time at which either a jump or a change of order occurs. We call each portion $(t_{i-1}, t_i)$ of the schedule a type 1 move. Every type 1 move corresponds to a continuous path in the canonical information space, $I$. On the other hand, the change occurring at each $t_i$ corresponds to a jump in $I$. Details about the different types of elementary moves are provided in the rest of the section.

We assume that at the begging of the move, the information state is

$$I' = (x_0', y_0', x_1', y_1', k', d_0', d_1', b_0', b_1', b_2', b_3')$$,
and at the end of the move the information state is

\[ I'' = (x''_0, y''_0, x''_1, y''_1, k'', d''_0, d''_1, b''_0, b''_1, b''_2, b''_3) \]

In the following paragraphs we denote the move from \( I' \) to \( I'' \) as \( (I' \Rightarrow I'') \) and we describe the move by defining the value of \( I'' \) as a function of \( I' \). Also, we note that the reverse move, from \( I'' \) to \( I' \), is also an elementary move. We denote it as \( (I' \Leftarrow I'') \) and we describe it by defining the value of \( I' \) as a function of \( I'' \). For simplicity, if any of the \( x_i \)'s or the \( y_i \)'s are the same in \( I' \) and in \( I'' \), we omit the ' symbol. An exhaustive list of all the elementary moves follows:

**Type 0: change of choice of \( x_0 \)**

This elementary move is purely technical. It represents no real change in the canonical information state, merely a permutation of the positions of the pursuers and the bits. Recall that in the definition of a canonical information state in Section 3.2.2 we chose \( x_0 \) arbitrarily out of \( \{p_0, q_0, p_1, q_1 \} \). This implies that depending on the choice of \( x_0 \), there are four different canonical information states which represent the same status of the pursuit. In order to account for that fact, we define a move which switches \( x_0 \) to be the next point in positive direction out of \( x_1, y_0 \). (Note that we do not have to consider \( y_1 \) as a next point since it is always after \( x_1 \).) Following is an example for a type 0 move from a canonical configuration with \( k = 0 \).

\[
\begin{array}{c|c|c}
| x' & b' & b' |
\hline
b' & b' & b' \\
| x' & y' & x' |
\end{array}
\quad
\begin{array}{c|c|c}
| x'' & b'' & b'' |
\hline
b'' & b'' & b'' \\
| x'' & y'' & x'' |
\end{array}
\]

Let \( I' \) be the information state before the switch. Depending on the value of \( k' \), after the switch the
value of \( I'' \) is as follows:

\[
(I' \Rightarrow I'') : \quad I'' = \begin{cases} 
(y_0', x_0', x_1', y_1', 1 - d_0, \ d_1', \ 2, b_0', b_1', b_2, b_3'), \ & \text{if } k' = 0 \\
(x_1', y_1', y_0', x_0', \ d_1', \ 1 - d_0, 1, b_0', b_1', b_2, b_3'), \ & \text{if } k' = 1 \\
(x_1', y_1', y_0', x_0', \ d_1', \ 1 - d_0, 0, b_0', b_1', b_2, b_3'), \ & \text{if } k' = 2
\end{cases}
\]

**Type 1: no jumps, no change in the relative order**

This is a continuous move which represents almost all of the time of the canonical schedule. Throughout the duration of the move there are no jumps. Both the relative order of \( x_0, y_0, x_1, y_1 \) and the values of the bits are preserved. The next figure provides an example for a type 1 move with \( k = 1 \). The move is similar for the other other values of \( k \).

![Type 1 Move Example](image)

Apart from the \( x \)'s and the \( y \)'s, the information states do not change:

\[
(I' \Rightarrow I'') : \quad k'' \leftarrow k', b_j' \leftarrow b_j', j \in Z_3
\]

\[
(I' \Leftarrow I'') : \quad k' \leftarrow k'', b_j'' \leftarrow b_j'', j \in Z_3
\]

A type 1 move followed immediately by its own reverse, leaves the contamination bits intact, regardless of their initial values.

**Type 2: clearing a corner**

The move occurs at a moment in which one of the light segments, \( x_iy_i \), becomes arbitrarily small. i.e., during the move points \( x_i \) and \( y_i \) merge at (or more precisely, converge arbitrarily close on both sides of) a non-reflex vertex, \( p \).

Since the type 0 moves allows us to reorder the points, without loss of generality, we can assume that \( x_1 \) and \( y_1 \) merge at \( p \) and \( p \in \partial P(x_1, y_1) \), as shown in the example.
Since the underlying contamination region can be made arbitrarily small, we assume that after the move the region incident to $p$ is clear. The forward and the reverse move are identical, so we only describe one:

$$(I' \Rightarrow I''): k'' = k', b'_j \leftarrow 0, b''_j \leftarrow b'_j, j \in \{0, 1, 3\}$$

**Type 3: point merge move**

The move occurs at a moment in which two endpoints of different light segments merge into a single one and subsequently their relative order changes. As mentioned before, the type 0 move allows us without loss of generality to choose $x_0$ such that $k' = 0$ and the merge is between $y_0$ and $x_1$. In the forward direction, the move represents a change in the relative order from $x_0 < y_0 < x'_1 < y_1$ to $x_0 < x'_1 < y_0 < y_1$, i.e., $y_0$ and $x_1$ switch positions.

At the moment in which $y_0 = x_1$ there is a change in the contamination regions. There were three regions prior to that moment and four regions after that. The newly created region is clear. The tuples
change as follows:

\[(I' \Rightarrow I'') : \quad k'' = 1, \quad b'_i' \leftarrow 0, \quad b''_j \leftarrow b'_j, j \in \{0, 2, 3\}\]

\[(I' \Leftarrow I'') : \quad k' = 0, \quad b'_i \leftarrow b''_j, \quad b''_j \leftarrow b'_j, j \in \{0, 2, 3\}\]

**Type 4: lightpoint jump**

Next we describe moves of type 4 and type 5. They are both similar in the sense that one of the pursuers makes a pursuer or lightpoint jump, while the other pursuer is stationary. The jump results in possible contamination of previously clear regions. (We call this recontamination.) There may also be a simultaneous change in the relative order of the points \(x_0, y_0, x_1, \) and \(x_1\).

Without loss of generality, we can choose \(x_0\) such that (i) pursuer 1 is stationary (ii) \(y'_0\) is an interior point of the line segment \(x_0y'_0\), (iii) there is a jump from \(y'_0\) to \(y''_0\). The jump is a lightpoint one if and only if \(d_0 = 0\). Otherwise, if \(d_0 = 1\), the jump is a pursuer one.

The way a jump will affect the values of \(k\) and \(b_j\) depends on the relative position of \(x_0, y'_0, y''_0, x_1\) and \(y_1\). One out of the 6 different orderings is not feasible, so there are 5 cases that we have to consider:

In the forward direction of this move, \((I' \Rightarrow I'')\), there is a single lightpoint jump, from \(y'_0\) to \(y''_0\),
i.e., \( d_0 = 0 \). The changes to the tuples are listed for each of the 5 possible cases:

(a) \((I' \Rightarrow I'')\): \(k'' \leftarrow 0, \quad b''_2 \leftarrow b''_1 \leftarrow b''_1 \lor b'_0, b''_j \leftarrow b'_j, j \in \{0, 2\}\)

\((I'' \Leftarrow I'')\): \(k' \leftarrow 0, \quad b'_0 \leftarrow b'_0 \lor b'_1, b'_j \leftarrow b'_j, j \in \{1, 2, 3\}\)

(b) \((I' \Rightarrow I'')\): \(k'' \leftarrow 1, \quad b''_2 \leftarrow b'_2 \lor b'_1, b''_j \leftarrow b'_j, j \in \{0, 1, 3\}\)

\((I'' \Leftarrow I'')\): \(k' \leftarrow 1, \quad b'_1 \leftarrow b'_1 \lor b'_2, b'_j \leftarrow b'_j, j \in \{0, 2, 3\}\)

(c) \((I' \Rightarrow I'')\): \(k'' \leftarrow 2, \quad b''_2 \leftarrow b'_2 \lor b'_1, b''_j \leftarrow b'_j, j \in \{0, 1, 2\}\)

\((I'' \Leftarrow I'')\): \(k' \leftarrow 2, \quad b'_0 \leftarrow b'_2 \lor b'_1, b'_j \leftarrow b'_j, j \in \{1, 3\}\)

(d) \((I' \Rightarrow I'')\): \(k'' \leftarrow 2, \quad b''_2 \leftarrow b''_1 \leftarrow b'_2 \lor b'_1, b''_j \leftarrow b'_j, j \in \{0, 1, 3\}\)

\((I'' \Leftarrow I'')\): \(k' \leftarrow 0, \quad b'_2 \leftarrow b'_0 \lor b'_1, b'_j \leftarrow b'_j, j \in \{2\}\)

(e) \((I' \Rightarrow I'')\): \(k'' \leftarrow 2, \quad b''_2 \leftarrow b''_1 \leftarrow b'_2 \lor b'_1, b''_0 \leftarrow b'_0 \leftarrow b'_0, b''_j \leftarrow b'_j, j \in \{0, 1, 2\}\)

\((I'' \Leftarrow I'')\): \(k' \leftarrow 1, \quad b'_2 \leftarrow b'_0 \lor b'_2, b'_j \leftarrow b'_j \lor b'_2, b_2 \leftarrow b''_2\)

Type 5: pursuer jump

In the forward direction of this move, \((I' \Rightarrow I'')\), there is a single pursuer, from \(y'_0\) to \(y''_0\), i.e., \(d_0 = 1\).

The changes to the tuples are listed for each of the 5 possible cases:

(a) \((I' \Rightarrow I'')\): \(k'' \leftarrow 0, \quad b''_2 \leftarrow b''_1 \leftarrow b''_1 \lor b'_1, b''_j \leftarrow b'_j, j \in \{0, 2\}\)

\((I'' \Leftarrow I'')\): \(k' \leftarrow 0, \quad b'_0 \leftarrow b'_1 \leftarrow b'_1 \lor b'_2, b'_j \leftarrow b'_j, j \in \{1, 2, 3\}\)

(b) \((I' \Rightarrow I'')\): \(k'' \leftarrow 1, \quad b''_2 \leftarrow b''_1 \leftarrow b'_2 \lor b'_1, b''_j \leftarrow b'_j, j \in \{0, 3\}\)

\((I'' \Leftarrow I'')\): \(k' \leftarrow 1, \quad b'_1 \leftarrow b'_2 \leftarrow b'_1 \lor b'_2, b'_j \leftarrow b'_j, j \in \{0, 3\}\)

(c) \((I' \Rightarrow I'')\): \(k'' \leftarrow 2, \quad b''_2 \leftarrow b''_1 \leftarrow b'_2 \lor b'_1, b''_j \leftarrow b'_j, j \in \{0, 3\}\)

\((I'' \Leftarrow I'')\): \(k' \leftarrow 2, \quad b'_0 \leftarrow b'_2 \leftarrow b'_0 \lor b'_2, b'_j \leftarrow b'_j, j \in \{0, 3\}\)

(d) \((I' \Rightarrow I'')\): \(k'' \leftarrow 2, \quad b''_2 \leftarrow b''_1 \leftarrow b'_2 \lor b'_1, b''_j \leftarrow b'_j, j \in \{0, 3\}\)

\((I'' \Leftarrow I'')\): \(k' \leftarrow 0, \quad b'_0 \leftarrow b'_2 \leftarrow b'_0 \lor b'_2, b'_j \leftarrow b'_j, j \in \{1, 3\}\)

(e) \((I' \Rightarrow I'')\): \(k'' \leftarrow 2, \quad b''_2 \leftarrow b''_1 \leftarrow b'_0 \lor b'_2, b''_j \leftarrow b'_j, j \in \{0, 1, 2\}\)

\((I'' \Leftarrow I'')\): \(k' \leftarrow 1, \quad b'_1 \leftarrow b'_0 \lor b'_2, b'_0 \leftarrow b'_2 \lor b'_0, b'_j \leftarrow b'_j, j \in \{0, 1, 2\}\)

3.3 Finite Representation

In the previous section we defined canonical configurations as a simpler model for the two pursuer problem. Yet the canonical information space is still an infinite one, which makes an exhaustive search for a winning canonical schedule infeasible. In this section we will introduce an equivalent, finite representation of the search space and we will show how to find a winning strategy by an exhaustive
3.3.1 VHC boxes

Let $X = (x, x)$ and $Y = (y, y)$ be two open intervals from $\partial P$ such that $x < x < y$ and $x < y < y$. Let $B$ be the region (most often a rectangular box) defined as \{(x, y) | x \in X, y \in Y, x < x < y\}. Suppose $C = B \cap X \neq \emptyset$. We define $B$ as a vertically and horizontally (VHC) convex box, and also we define $C$ as a VHC core, if both conditions are true:

- for every $x \in X$, the set $\{y | (x, y) \in C\}$ is a single connected nonempty interval. This condition corresponds to horizontal convexity.
- for every $y \in Y$, the set $\{x | (x, y) \in C\}$ is a single connected nonempty interval. This condition corresponds to vertical convexity.

We define eight parameters or extreme points, which will be sufficient to represent $C$. Let $\overline{x}$ be the lower boundary of the interval $X$. If $(\overline{x}, y^1)$ and $(\overline{x}, y^2)$ are the endpoints of the segment $\{(x, y) | (x, y) \in C\}$, we define them as the extreme points for $\overline{x}$. Similarly, there are three other pairs of extreme points, two points for each $x, y$ and $\overline{y}$. For example of a VHC box, VHC core, and corresponding parameters, refer to Figure 3.5. In the rest of the paper, we will not be interested in the precise shape of $C$. Instead, the points $\overline{x}, x, \overline{x}, y, x, x, y^1, y^2, y^3, y^4$ will be sufficient to record the extremums of $C$. Note that the extreme points need not be distinct and it is possible for a box to have as little as two unique extreme points.

For $i = 0, 1$, let $X_i = (x_i, x_i)$, and $Y_i = (y_i, y_i)$ be open intervals on the boundary $\partial P$, defining the VHC box $B(X_i, Y_i)$. Consider all the intersections of an interval from $\{X_0, Y_0\}$ with an interval from
\{X_1, Y_1\}. If at most one of the intersections is nonempty, we say that the two boxes are independent. Otherwise, we say that the boxes are dependent.

The next lemma allows us to determine in constant time whether a particular visibility tuple is feasible.

**Lemma 3.4.** Let \(X_0, Y_0, X_1,\) and \(Y_1\) be open intervals on the boundary \(\partial P\), defining VHC boxes \(B(X_0, Y_0)\) and \(B(X_1, Y_1)\). Given an integer \(k \in \{0, 1, 2\}\) and the parameters of the two boxes, define the condition

\[
\exists (x_i, y_i) \in C(X_i, Y_i), 0 \leq i \leq 1 : I_v = (x_0, y_0, x_1, y_1, k) \in I_v.
\]

(3.1)

The condition above can be evaluated in \(O(1)\) time. If existing, a tuple \(I_v\) can be constructed in time \(O(1)\) if the boxes are independent and in time \(O(n)\) if the boxes are dependent.

**Proof.** The proof is presented in Appendix A. \(\square\)

### 3.3.2 Concave regions and red intervals

A maximal nonempty subinterval of \(\partial P\) of the form \(C = (p_i, p_{i+1})\), where all the vertices \(p_{i+1}, \ldots, p_j\) are reflex vertices, forms a concave region. Obviously, two concave regions cannot overlap and must be separated by non-concave regions (each of which contains at least one non-reflex vertex). Let the \(k\)-th concave region of \(P\) be \((p_i, p_{j+1})\). We define points \(a_k\) and \(b_k\) as the midpoints of the edges \(e_i\) and \(e_j\), respectively, and refer to them as critical points.\(^4\) For example, the polygon in Figure 3.4(a) has two concave regions, \((5, 10)\) and \((22, 29)\). The midpoints of the edges \((5, 6), (9, 10), (22, 23)\) and \((28, 29)\) are the critical points for the polygon.

The critical points of a polygon \(P\) divide \(\partial P\) into intervals \(R_0, R_1, \ldots, R_{2m-1}\), where \(m\) is the number of concave regions in the polygon. The critical points define a grid of vertical and horizontal lines over \(X\) which partition \(X_v\) into multiple maximal connected subsets. The next lemma establishes a connection between grid elements and VHC cores.

**Lemma 3.5.** Consider the grid over \(X\) defined by the critical points and let \(A\) be an arbitrary grid element. If \(C\) is the intersection of the interior of \(X_v\) with the interior of \(A\), then \(C\) is either empty or it is a VHC core.

**Proof.** The proof follows directly from the lemmas in Section 3.2 of [29]. \(\square\)

\(^4\)The terms concave regions and critical points of \(\partial P\) have been defined previously, see [29] for more details.
Later in the paper we will show that for each $j_0, j_1 \in \mathbb{Z}_{2m}$, there is at most one maximal VHC box $B(X,Y)$, such that $X \subseteq R_{j_0}$ and $Y \subseteq R_{j_1}$. We will describe how given $j_0$ and $j_1$ we can compute the extreme points of $B(X,Y)$ and vice-versa. This will allow us, instead of considering the rather complex shape of $X_v$, to work with the grid elements each of which is by itself has a simple shape as guaranteed by the vertical and horizontal convexity property.

### 3.3.3 Equivalence classes in $\mathcal{I}$ and finite search schedule

In Section 3.1.3 we defined contamination regions in order to capture equivalence of positions of the evader. In this section we explore similarities between mutual positions of the two pursuers in order to group together equivalent visibility tuples and also equivalent canonical information states.

We define a binary relation on the visibility tuples. Let $P$ be a polygon with a corresponding partition of $\partial P$ into red intervals. Consider a pair of visibility tuples $I_{i} = (c_{i}^{y}, x_{i}^{y}, y_{i}^{y}, f_{i}), i \in \{0,1\}$.

We say that $I_{0}^i$ is similar to $I_{1}^i$ if there exist red intervals $R_{j_0}, R_{j_1}, R_{j_2},$ and $R_{j_3}$, such that, for $i \in \{0,1\}, x_{0}^{i} \in R_{j_0}, y_{1}^{i} \in R_{j_1}, x_{1}^{i} \in R_{j_2}, y_{1}^{i} \in R_{j_3}$. Clearly, “similar” is an equivalence relation so from now on we just say that $I_{0}^i$ and $I_{1}^i$ are similar. We denote the equivalence class containing $I_{0}^0$ and $I_{1}^0$ as $(j_0, j_1, j_2, j_3, k)$. The relation partitions $\mathcal{I}_v$ into $O(m^4)$ equivalence classes.

The “similar” relation on visibility tuples can be extended to a relation on canonical information states. Let $I_{0}^i$ and $I_{1}^i$ be two canonical information states, such that for $i \in \{0,1\}, I_{i}$ is a concatenation of a visibility tuple $I_{v}^i$ and a bits tuple $I_{b}$. We say that $I_{0}^i$ and $I_{1}^i$ are similar if $I_{v}^0$ and $I_{v}^1$ are similar. The “similar” relation over informations states is also an equivalence relation. It partitions the information space $\mathcal{I}$ into $O(m^4)$ equivalence classes of the form $(j_{0}, j_{1}, j_{2}, j_{3}, k, d_{0}, d_{1}, b_{0}, b_{1}, b_{2}, b_{3})$, where $j_{0}, j_{1}, j_{2}, j_{3},$ and $k$ are as described above, while $d_{0}, d_{1}$ and $b_{0}, b_{1}, b_{2}, b_{3},$ are the direction and contamination bits, correspondingly.

**Lemma 3.6.** Let $I_{0}^i$ and $I_{1}^i$ be similar canonical information states, contained in the equivalence class $v$. It follows that there is a type 1 move from $I_{0}^0$ to $I_{1}^1$ entirely over canonical information states in $v$.

**Proof.** Follows from the definition of VHC boxes. \qed

We define the directed information state graph $G = (V,E)$ to capture the equivalence classes of the similar relation over $\mathcal{I}$. The set of edges of $G$ is $V(G) \subset (\mathbb{Z}_{2m})^4 \times \mathbb{Z}_3 \times (\mathbb{Z}_2)^6$. A tuple
$v = (j_0, j_1, j_2, j_3, k, d_0, d_1, b_0, b_1, b_2, b_3) \in V(G)$ if and only if $v$ is a (non-empty) equivalence class of the similar relation over $I$. The set of edges, $E(G)$, consists of all the pairs $(v^0, v^1)$, such that there is an elementary move from some canonical information state in $v^0$ to some canonical information state in $v^1$. Intuitively, we are replacing the canonical schedules (i.e., piecewise continuous trajectories in $I$) defined in Section 3.2.2, with corresponding finite paths in $G$.

Define $v \in V(G)$ to be a starting (respectively, goal) vertex if $v$ contains a starting (respectively, goal) canonical information state. A winning finite schedule is a path in $G$ from a starting to a goal vertex.

**Lemma 3.7.** For a polygon $P$, there exists a winning canonical schedule, if and only if there exists a winning finite schedule.

**Proof.** Let $I$ and $G$ be the canonical information space and the information state graph $G$ for the polygon $P$.

Suppose there exists a winning canonical schedule, i.e., there exists a path $\pi \subset I$ from a starting to a goal information state, such that $\pi$ consists of elementary moves. The path $\pi$ can be divided into subpaths $\pi_0, \pi_1, \ldots, \pi_k$ where each $\pi_i$ is a path within the same equivalence class $v^i \subset I$. $v^i \in V(G)$, $0 \leq i \leq k$. Since the path $\pi_0$ begins at a starting canonical state, then $v^0$ is a starting vertex. Also, for $1 \leq i \leq k$, the transition from $\pi_{i-1}$ to $\pi_i$ corresponds to an elementary move, so $(v^{i-1}, v^i) \in E(G)$. It follows that the corresponding path $v^0, v^1, \ldots, v^k$ is a finite schedule. Finally, $\pi_k$ ends at a goal canonical information state, so $v^k$ is a goal vertex and the path $v^0, v^1, \ldots, v^k$ is a winning finite schedule.

Now suppose that there exists a winning finite schedule $v^0, v^1, \ldots, v^k$. For $1 \leq j \leq k$, from $(v^{j-1}, v^j) \in E(G)$, it follows that there exist canonical information states $I''_{j-1} \in v^{j-1}$ and $I'_j \in v^j$ and also a path $\pi_j$ in $I$ which corresponds to an elementary move from $I''_{j-1}$ to $I'_j$. However, in general, for $1 \leq j \leq k - 1$, $I'_j \neq I''_j$, so simply concatenating the paths $\pi_1, \pi_2, \ldots, \pi_k$ will not yield a valid canonical schedule. On the other hand, since for $1 \leq j \leq k - 1$, $I'_j$ and $I''_j$ belong to the same equivalence class $v^j$, we can apply Lemma 3.6, which states that there is a (type 1) elementary move between the two states, thus there is a path $\rho_j$ from $I'_j$ to $I''_j$. Define the path $\pi \subset I$ as a concatenation of $\pi_1, \rho_1, \pi_2, \ldots, \pi_{k-1}, \rho_{k-1}, \pi_k$. The vertex $v^0$ is a starting vertex, thus $I''_0 \in v^0$ is a starting canonical information state, therefore $\pi$ is a canonical schedule. Finally, $v^k$ is a goal vertex, thus $I'_k \in v^k$ is a goal canonical information state, so $\pi$ is a winning canonical schedule.

Lemma 3.7 shows how to transform a winning canonical schedule into a winning finite schedule, however, it does not describe how to construct a winning canonical schedule, given only the polygon $P$. 

3.4 Algorithm for Finding a Finite Search Schedule

In the next section we provide an algorithm which, given \( P \), first constructs the graph \( G \) and then performs a breadth-first search in \( G \) to find a winning finite schedule. If for a given polygon \( P \) and graph \( G \) such a schedule exists, it will serve as a description of a winning strategy for the two pursuers. The running time of the algorithm is \( O(m^4 + m^2n + n^2) \) where \( n \) is the number of edges and \( m \) is the number of concave regions of the polygon.

In order to construct the graph \( G \), it is sufficient to construct the vertex set \( V(G) \) and the edge set \( E(G) \). In the next sections we provide details for both constructions.

3.4.1 Constructing \( V'(G) \)

In this section, we describe how to construct \( V'(G) \), given the parameters of all the \( O(m^2) \) VHC boxes defined by the critical points. Since the parameters of each box will be used in \( O(m^2) \) different computations, it is helpful to precompute them.

Suppose \( p \in \partial P \) is a critical point. By constructing the visibility polygon for \( p \), we can identify all pairs \((q_1, q_2)\) such that \((q_1, q_2)\) corresponding to a (left or right) gap edge relative to \( p \). We define configurations \((p, q_1), (p, q_2), (q_1, p), (q_2, p) \in \mathcal{X}\), to be critical gap configurations.

\[
\begin{array}{c}
\text{d'} \\
\text{c} \\
\text{x}
\end{array} \\
\begin{array}{c}
\text{d} \\
\text{c'}
\end{array}
\begin{array}{c}
\text{c'} \\
\text{d'}
\end{array} \\
\begin{array}{c}
\text{d'} \\
\text{c'}
\end{array}
\begin{array}{c}
\text{d} \\
\text{c'}
\end{array}
\begin{array}{c}
\text{d'} \\
\text{c'}
\end{array}
\begin{array}{c}
\text{d'} \\
\text{c'}
\end{array}
\]

Figure 3.6 Example of bitangents (shown as dashed lines) and bitangent points (shown as points \( c, c', d \) and \( d' \)).

Let \( c, d \in \partial P \) be mutually visible vertices and \( x \in P \) be an interior point of the segment \( \overline{cd} \). see Figure 3.6. If there exist points \( c', d' \in \partial P \) such that the pairs \((c, c')\) and \((d, d')\) form gap edges relative to \( x \), we say that \( c \) and \( d \) define a bitangent. We call \( c, d, c' \) and \( d' \) bitangent points. If \( p, q \in \{c, d, c', d'\} \) and \( p \neq q \) we define the pair \((p, q) \in \mathcal{X}\) as a bitangent configuration.

Lemma 3.8. For a polygon with \( n \) edges and \( m \) concave regions, the parameters of all VHC boxes defined by the corresponding red grid can be precomputed in a \( 2m \times 2m \) matrix in \( O(n^2) \) time. Thus,
given the matrix and $j_0, j_1 \in \mathbb{Z}_{2m}$, in constant time we can determine the red intervals, $R_{j_0}$ and $R_{j_1}$, and the parameters of the VHC box $B(X, Y)$, where $X \subseteq R_{j_0}$ and $Y \subseteq R_{j_1}$, if such a box exists.

Proof. The lemma summarizes the discussion so far: every extreme point of a VHC box is either a bitangent configuration or a critical gap configuration. By traversing the boundary, in $O(n)$ time we can construct a vector of size $2m$ which stores the critical points, i.e., the endpoints of the red intervals $R_{j_0} \ldots R_{j_{2m-1}}$. For each critical point in $O(n)$ time we can construct the corresponding visibility polygon which determines the critical gap configurations. So the time to determine the critical gap configurations for all critical points is $O(n + 2mn) = O(mn)$.

In order to determine the bitangent points, for every vertex $p_i$ of $P$ we can compute the visibility polygon of $p_i$ in $O(n)$ time. Since there are $n$ vertices, the total time to determine the bitangent configurations is $O(n^2)$. Thus the total time to determine the parameters of all VHC boxes is $O(mn + n^2) = O(n^2)$. □

Lemma 3.9. For a given polygon, if the parameters of all the boxes are precomputed, then $V(G)$, the vertex set of the information state graph $G$, can be constructed in time $O(m^4)$.

Proof. Let $v = (j_0, j_1, j_2, j_3, k, d_0, d_1, b_0, b_1, b_2, b_3)$ be a tuple in which $j_0, j_1, j_2, j_3 \in \mathbb{Z}_{2m}$ are the indices of red intervals $R_{j_0}, R_{j_1}, R_{j_2}, R_{j_3}$, while $k \in \mathbb{Z}_3$ corresponds to the order, and the $d_i$'s and $b_i$'s are the direction and contamination bits correspondingly. We show how to determine whether $v \in V(G)$.

If the parameters of each box are precomputed as described in Lemma 3.8, then in constant time we can construct VHC boxes $B(X_0, Y_0)$ and $B(X_1, Y_1)$, where $X_0 \subseteq R_{j_0}$, $Y_0 \subseteq R_{j_1}$, $X_1 \subseteq R_{j_2}$, $Y_1 \subseteq R_{j_3}$.

If any of the two boxes does not exist, then $v \not\in V(G)$, so assume that both boxes exist.

Note that $v \in V(G)$ does not depend on the values of the bits $d_i$ and $b_i$, but only on the mutual position of the parameters of the boxes, as well as on the value of $k \in \mathbb{Z}_3$. Therefore, using Lemma 3.4 we can determine the existence of a visibility tuple $I_v = (x_0, y_0, x_1, y_1, k) \in I_v$, where $(x_i, y_i) \in C(X_i, Y_i)$.

But this is equivalent to determining whether there exists a canonical information state $I = (x_0, y_0, x_1, y_1, k, d_0, d_1, b_0, b_1, b_2, b_3) \in I$ where $(x_i, y_i) \in C(X_i, Y_i)$, which itself is equivalent to determining whether $v \in V(G)$.

To construct the set $V(G)$, we consider all possible choices of $j_0, j_1, j_2, j_3 \in \mathbb{Z}_{2m}, k \in \mathbb{Z}_3, d_i, b_i \in \mathbb{Z}_2$. Ignoring $d_i$ and $b_i$ for each tuple $I_v$, we can evaluate Equation 3.1 from Lemma 3.4 in $O(1)$ time. Since the number of all such tuples is $O(m^4)$, the set $V(G)$ can be constructed in $O(m^4)$ time. □
3.4.2 Constructing \( E(G) \)

In order to construct the edge set, \( E(G) \), of the information state graph \( G \), we regard \( E(G) \) as a (not necessarily disjoint) union of sets of edges:

\[
E(G) = \bigcup_{i=0}^{5} E_i,
\]

where for \( 0 \leq i \leq 5 \), \( E_i \subseteq V(G) \times V(G) \) is set of edges, such that \((v^0, v^1) \in E_i \) if and only if there is a type \( i \) elementary move from a canonical information state in \( v^0 \) to a canonical information state in \( v^1 \).

In the rest of the section we will show how to construct each of the sets \( E_i \), \( 0 \leq i \leq 5 \).

3.4.2.1 Constructing \( E_0 \)

The edge corresponds the completely technical type 0 move, during which we change the reference point \( x_0 \) in a tuple. As a result of the move the order \( k \) may change, while the other elements in the tuple are merely permuted, since there is no real motion of the light segments, merely a relabeling of their endpoints.

For \( i = 0, 1 \), consider vertex \( v^i = (j_0, j_1, j_2, j_3, k, d_0, d_1, b_0, b_1, b_2, b_3) \in V(G) \), with corresponding VHC boxes \( B(X^0, Y^0) \) and \( B(X^1, Y^1) \), where \( X^0 \subseteq R_j, Y^0 \subseteq R_j \), and \( X^1 \subseteq R_j \), and \( Y^1 \subseteq R_j \). Pick an arbitrary tuple \( l^0 = (x_0^0, y_0^0, x_1^0, y_1^0, k^0, d_0^0, d_1^0, b_0^0, b_1^0, b_2^0, b_3^0) \), such that \((x_0^0, y_0^0) \in B(X_j^0, Y_j^0) \), \( l \in \{0, 1\} \) and \( k^0 \) corresponds to the order of \( x_0^0, y_0^0 \). As shown in Section 3.2.3, given \( l^0 \), in constant time we can build a second tuple \( l' = (x_0^1, y_0^1, x_1^1, y_1^1, k', d_0^1, d_1^1, b_0^1, b_1^1, b_2^1, b_3^1) \), which is the result of applying a type 0 move on \( l^0 \). Then

\[
(v^0, v^1) \in E_0 \iff l' \in v^1 \iff x_0^1 \in X_0^1 \land y_0^1 \in Y_0^1 \land x_1^1 \in X_1^1 \land y_1^1 \in Y_1^1 \land d_j^1 = d_j^0 \land b_i^1 = b_i^0 \quad (3.2)
\]

Clearly the condition in Equation 3.2 can be computed in constant time. In order to compute all the edges in \( E_0 \), we need to consider all the possible pairs of vertices \( v^0, v^1 \in V(G) \), so the total time is \( O(m^2 \cdot m^2) = O(m^4) \).

3.4.2.2 Constructing \( E_1 \)

Intuitively, a type 1 edge represents a move during which pursuer 0 moves between neighboring grid elements while pursuer 1 is stationary. There are no jumps and the relative order of the endpoints of the light segments does not change.
Consider distinct vertices $v^i = (j^i_0, j^i_1, j^i_2, j^i_3, k, d_0, d_1, b_0, b_1, b_2, b_3) \in V(G)$, $i = 0, 1$, and the corresponding VHC boxes $B(X^0_0, Y^0_0)$, $B(X^0_1, Y^0_1)$ and $B(X_1, Y_1)$, where $X^0_0 \subseteq R^0_1$, $Y^0_0 \subseteq R^0_2$, $X^0_1 \subseteq R^0_2$, $Y^0_1 \subseteq R^j_1$, $X_1 \subseteq R^1_1$, $Y_1 \subseteq R^1_2$, and $Y^1 \subseteq R^j_2$. Assume that $B(X^0_0, Y^0_0)$, $B(X^0_1, Y^0_1)$ share a (vertical or a horizontal) grid segment $s_1, s_4$. (To avoid tedious technicalities, i.e., a change in $k$, assume also that $j^0_0 \neq j^1_1$.)

Let $s_2, s_3$ be the maximal subsegment of $s_1, s_4$ whose interior lies entirely in $X$. If $s_2, s_3$ is not empty then there exists a VHC box $B(X, Y)$ which is an infinitely small open set which contains $s_2, s_3$.

**Observation 3.10.** There exists a type 1 move between canonical configurations from $v^0$ to $v^1$, i.e., $(v^0, v^1) \in E_1$, if and only if $k, B(X, Y)$ and $B(X_1, Y_1)$ satisfy Lemma 3.4.

**Proof.** Note that Lemma 3.4 is satisfied by some $I_v = (x_0, y_0, x_1, y_1, k) \in I_v$ if and only if there are points $(x^0_0, y^0_0) \in B(X^0_0, Y^0_0)$, sufficiently close to $(x_0, y_0)$, so that $I^i_v = (x^i_0, y^i_0, x_1, y_1, k) \in I_v$, $i = 0, 1$. This is equivalent to the existence of a type 1 move between canonical information states $I^i = (x^i_0, y^i_0, x_1, y_1, k, d_0, d_1, b_0, b_1, b_2, b_3) \in I$, $i = 0, 1$, i.e., equivalent to $(v^0, v^1) \in E_1$. □

It follows that determining whether $(v_0, v_1) \in E_1$ is equivalent to evaluating Equation 3.1 from Lemma 3.4. The condition can be verified and $I_v$ can be constructed in constant time for independent VHC boxes and in time $O(n)$ for dependent ones. There are $O(m^4)$ different independent boxes and $O(m^2)$ dependent ones, so to construct the edges in $E_1$ we need time $O(m^4 + m^2 n)$.

### 3.4.2.3 Constructing $E_2$

Intuitively, a type 2 edge represents a moment during which pursuer 0 clears the corner corresponding to a non-reflex vertex while pursuer 1 is stationary.

For $i = 0, 1$, consider $v^i = (j^i_0, j^i_1, j^i_2, j^i_3, k, d_0, d_1, b_0, b_1, b_2, b_3) \in V(G)$, with corresponding boxes $B(X_0, Y_0)$ and $B(X_1, Y_1)$, where $X_0 \subseteq R^0_0$, $Y_0 \subseteq R^0_1$, $X_1 \subseteq R^j_1$, $Y_1 \subseteq R^j_2$. If neither of $B(X_0, Y_0)$ or $B(X_1, Y_1)$ contains a non-reflex vertex, then $(v^0, v^1) \notin E_2$. So assume that there is a non-reflex vertex $p \in \partial P$ which belongs to one of the boxes. Let $p^-, p^+ \in \partial P$, $p \in (p^-, p^+)$ and also define the intervals $X = (p^-, p)$ and $Y = (p, p^+)$. Let $p^-, p^+$ be sufficiently small so that $B(X, Y) = C(X, Y) \subseteq X$. Without loss of generality, possibly after some relabeling, we can assume that $p \in B(X_1, Y_1)$ and neither of $X$ or $Y$ overlaps with $X_0$ or $Y_0$. Suppose that $(x_0, y_0)$ is an arbitrary point from $C(X_0, Y_0)$. If pursuer 0 is stationary at $(x_0, y_0)$ while pursuer 1 converges from $(p^-, p^+)$ to $(p, p)$, this corresponds to a type 2 move, as described in Section 3.2.3.
To determine whether \((v^0, v^1) \in E_2\), we just have to verify that the contamination bits \(b_0^i, b_1^i, b_2^i, b_3^i, i = 0, 1\), are consistent with the bit changes for a type 2 move. More precisely, let \(I^i = (x_0, y_0, p^-, p^+, k, d_0, d_1, b_0^i, b_1^i, b_2^i, b_3^i), i = 0, 1\). In constant time we can determine whether there is a type 2 move between \(I^0\) and \(I^1\). It follows that membership in \(E_2\) can be determined in constant time. There are \(O(n)\) possible values for \(p\), i.e., for the box \(B(X, Y)\) and \(O(m^2)\) possible values for \(B(X_0, Y_0)\), so the total time to construct \(E_2\) is \(O(nm^2 + m^4)\).

### 3.4.2.4 Constructing \(E_3\)

Intuitively an edge in \(E_3\) corresponds to a type 3 move between canonical information states. Each of the pursuers moves continuously but within the same VHC core. The order \(k\) changes and there is a possible change in the contamination bits.

For \(i = 0, 1\), consider \(v^i = (j_0, j_1, j_2, j_3, k, d_0, d_1, b_0^i, b_1^i, b_2^i, b_3^i) \in V(G)\). with corresponding boxes \(B(X_0, Y_0)\) and \(B(X_1, Y_1)\), where \(X_0 \subseteq R_{j_0}, Y_0 \subseteq R_{j_1}, X_1 \subseteq R_{j_2}, Y_1 \subseteq R_{j_3}\). If none of \(X_0\) or \(Y_0\) intersects with \(X_1\) or \(Y_1\), then there can be no overlap between the endpoints of the light segments, thus there can be no type 3 elementary move and \((v^0, v^1) \notin E_3\). So without loss of generality, possibly after some relabeling, we can assume that \(Y_0 \cap X_1 \neq \emptyset\). Choose \(x_0, y_0, y_1 \in \partial P\), such that \(y_0 \in Y_0 \cap X_1\), also \((x_0, y_0) \in C(\tilde{X}_0, \tilde{Y}_0), (y_0, y_1) \in C(\tilde{X}_1, \tilde{Y}_1)\). Since \(Y_0 \cap X_1\) is a nonempty open set there exists a sufficiently small interval \((x_1^i, x_0^i) \in Y_0 \cap X_1\), with the property that \(y_0 \in (x_1^i, x_0^i)\) and also that the entire segment between the points \((x_1^i, y_1)\) and \((x_0^i, y_1)\) lies in \(C(X_1, Y_1)\). If pursuer 0 is stationary at \((x_0, y_0)\), while pursuer 1 moves from \((x_0^i, y_1)\) over \((y_0, y_1)\) to \((x_1^i, y_1)\), this corresponds exactly to a type 3 move, as described in Section 3.2.3.

To determine whether \((v^0, v^1) \in E_3\), we just have to verify that the contamination bits \(b_0^i, b_1^i, b_2^i, b_3^i, i = 0, 1\), are consistent with the bit changes for a type 3 move. More precisely, let \(I^i = (x_0, y_0, x_1^i, y_1, k, d_0, d_1, b_0^i, b_1^i, b_2^i, b_3^i), i = 0, 1\). In constant time we can determine whether there is a type 3 move between \(I^0\) and \(I^1\). It follows that membership in \(E_3\) can be determined in constant time. There are \(O(m^4)\) possible values for \(v^0\) and \(v^1\), so the total time to construct \(E_3\) is \(O(m^4)\).

### 3.4.2.5 Constructing \(E_4\) and \(E_5\)

Intuitively the edges in \(E_4\) and \(E_5\) correspond to the type 4 and 5 jump elementary moves. For every elementary move, given the original canonical information state \(I^0\) and the type of the move we can construct the resulting canonical information state \(I^1\). Thus our main goal is to determine whether there is a feasible jump between the corresponding visibility tuples \(I^0_v\) and \(I^1_v\). If the latter jump exists,
we can determine in constant time whether there is a type 4 or type 5 jump between the corresponding information states $I^0$ and $I^1$.

![Figure 3.7](image)

Figure 3.7 (a) Tangent defined by points $a$, $b$ and $c$. (b) Corresponding configurations in $\mathcal{X}$. The points in $\mathcal{X}_0$ are white, the points in $\mathcal{X} - \mathcal{X}_0$ are grey.

Consider a tangent defined by the points $a, b, c \in \partial P$, see Figure 3.7(a). The tangent induces two corresponding jumps in $\mathcal{X}$: a jump from $(a, b)$ to $(a, c)$ is a jump over a left gap edge represented a horizontal segment and a jump from $(c, b)$ to $(c, a)$ is a jump over a right gap edge, represented as a vertical segment, see Figure 3.7(b). Just as in the definition of the type 4 and type 5 elementary moves, we will only discuss the jumps over left gap edges. The jumps over right gap edges are analogous, e.g., they can be constructed by considering a mirror image of the polygon.

Suppose pursuer 0 performs a jump over a gap edge from $(a, b)$ to $(a, c)$, such that $x_0 = a$, $y_0 = b$ and $y_1 = c$. The points $a, b, c \in \partial P$ induce a partition of $\mathcal{X}_0$ into six regions $(k^0, k^1)$, $0 \leq k^0 \leq k^1 \leq 2$, where every region $(k^0, k^1)$ contains all the points $(x^1, y^1)$ such that there exist $I^0_i = (x_0, y_0, x_1, y_1, k^i)$, $i = 0, 1$. Intuitively, the visibility tuples $I^0_0$ and $I^0_1$ denote a move in which pursuer 0 is making a jump from $x_0, y_0$ to $x_0, y_1$, while pursuer 1 is stationary at $(x_1, y_1)$. Region $(k^0, k^1)$ denotes all the positions of pursuer 1, such that the order of the $I^0_i$ is $k^0$ and the order $I^0_i$ is $k^1$, see Figure 3.7(b).

Clearly, if we fix $x_0, y_0$ and $y_1$ and we are also given the VHC box of $(x_1, y_1)$, in constant time we can determine the feasibility of a jump. However, since the number of possible positions for $x_0, y_0$ and $y_1$ is infinite, our next goal is to partition all the jump moves into a finite number of classes.

Consider red intervals $R_{j_0}$, $R_{j_1}$, $R_{j_1}$ and let $B(X_0, Y_0^0)$ and $B(X_0, Y_0^1)$ be the corresponding VHC boxes where $X_0 \subseteq R_{j_0}$, $Y_0^0 \subseteq R_{j_1}$, $Y_1^0 \subseteq R_{j_1}$. Let $(x_0^0, x_0^1) \subseteq X_0$ be a maximal interval with the property that for every $x^0 \in (x_0^0, x_0^1)$, there is a jump from $(x^0, y^0) \in C(X_0^0, Y_0^0)$ to $(x^0, y^1) \in C(X_0^1, Y_0^1)$. We
define \((x'_0, x''_0)\) to be a jump interval and we use it to group together equivalent jumps from \(B(X'_0, Y'_0)\) to \(B(X'_1, Y'_1)\). For two given boxes as defined above a jump interval \((x'_0, x''_0)\) may not exist or may not be unique, yet its boundaries always correspond to either bitangent or critical gap configurations. Therefore, since each such point can border at most two intervals and the number of both the bitangent configurations and the critical gap configurations is \(O(m^2)\), it follows that the jump intervals are also \(O(m^2)\).

Let \(v^i = (j_0, j'_1, j_2, j_3, k^1, d_0, d'_1, b_0, b'_1, b_2, b'_2)\), \(i = 0, 1\) be vertices in \(V(G)\) with corresponding boxes \(B(X_0, Y'_0)\), \(B(X_0, Y'_1)\) and \(B(X_1, Y'_1)\), where \(X_0 \subseteq R_{j_0}, Y'_0 \subseteq R_{j'_1}, Y'_1 \subseteq R_{j'_1}, X_1 \subseteq R_{j_2}, Y'_1 \subseteq R_{j_2}\). Assume that \((a', a'')\) is a jump interval for boxes \(B(X_0, Y'_0)\) and \(B(X_0, Y'_1)\). Suppose that \(b', b'', c'\) and \(c''\) are such that there is a jump from \((a', b')\) to \((a', c')\) and also a jump from \((a'', b'')\) to \((a'', c'')\) and let \(x_0 \in (a', a''), y'^o_0 \in (b', b'')\) and \(y'_1 \in (c', c'')\) by such that there is a jump from \((x_0, y'^o_0)\) to \((x_0, y'_1)\) as shown in Figure 3.8(a). Figure 3.8(b) illustrates the corresponding VOD \(X_v\). The white points correspond to the points in \(X_v\). The shaded points are the ones in \(X - X_v\). The trapezoid which is shaded darker represents all the different jumps for the given jump interval. The red lines corresponding to the boundaries of the intervals \(R_{j_0}, R_{j'_0}\) and \(R_{j'_1}\), shown as thicker lines in Figure 3.8(b), are part of the grid and divide \(X\) into regions labeled with X, A, B, C and D. Note that there are no points from \(X_v\) in a region labeled X, so \(B(X_1, Y'_1)\) has to be in one of the other four. If \(B(X_1, Y'_1)\) lies in a region labeled A or B, then we can determine in constant time whether there exist \(I'_0\) and \(I'_1\) and a corresponding type 4 (resp. type 5) move between them, and we can determine in constant time whether \((v^0, v^1) \in E_4\) (resp. \(\in E_5\)). On the other hand, if \(B(X_1, Y'_1)\) lies in a region labeled C or D, we need to construct a finite number of visibility polygons (similar to the proof of Lemma 3.4) to determine membership in \(E_4\) and \(E_5\).

What is the time needed to construct \(E_4\) and \(E_5\)? For a fixed jump interval the intervals \(R_{j_0}, R_{j'_0}\) and \(R_{j'_1}\) are fixed as well. There are \(O(m^2)\) possible choices for \(R_{j_2}\) and \(R_{j_3}\). Since there are \(O(m^2)\) boxes \(B(X_1, Y'_1)\) which lie in regions labeled A or B and each one take \(O(1)\) time, the total time for those is \(O(m^2)\). On the other hand, there are \(O(1)\) boxes which lie in a region labeled C or D and each one takes \(O(n)\) time, so the total time for the C and D regions is \(O(n)\). Thus, for a single jump interval the time is \(O(n + m^2)\). There are \(O(m^2)\) jump intervals so the total time to construct \(E_4\) and \(E_5\) is \(O(m^2(n + m^2)) = O(m^4 + nm^2)\).
Figure 3.8 Jump interval and partition of $\mathcal{X}$ into regions of different labels.
3.4.3 Finding a winning finite schedule

We can precompute the bitangent and critical gap configurations in $O(n^2)$ time. Given this precomputation, we can construct the graph $G$ in time $O(m^4 + m^2n)$. Finally, we can run breadth-first search in the graph to find a winning finite schedule. The size of the vertex set, $V(G)$, is $O(m^4)$. The size of the edge sets $E_0$, $E_1$, $E_2$ and $E_3$ is also $O(m^4)$ since every vertex in $V(G)$ has a constant outdegree for each of those edge sets. Finally, the size of the edge sets $E_4$ and $E_5$ is $O(m^4)$ as well, since there is a constant number of edges for every jump interval and box $B(X_1, Y_1)$. Thus, the size of $E(G)$ is $O(m^4)$; therefore, breadth-first search in $G$ will take $O(m^4)$ time.

It follows that determining the existence and constructing a winning finite schedule can be done in time $O(n^2 + nm^2 + m^4)$.

3.5 Conclusion

We presented a complete algorithm for a pair of pursuers, each with one rotating flashlight, searching for a moving target in a simple polygon. For a polygon with $n$ edges and $m$ concave regions, the algorithm in time $O(n^2 + nm^2 + m^4)$ decides whether it can be cleared by the pursuers, and if so, constructs a search schedule. The algorithm can be implemented and embedded on any moving devices with unidirectional vision (flashlights, lasers, or cameras). A natural direction for extending the current results is designing a similar algorithm for two pursuers with 360° vision. A more ambitious goal is to provide an algorithm for searching a polygon without holes using any number of pursuers. Another interesting problem is combining the results of our paper with the minimal sensing approach of Sachs et al [37], i.e., whether the two pursuers can find a winning strategy without prior knowledge of the shape of the polygon.
4 INTEGRATING SECURITY IN THE MAC LAYER OF WDM OPTICAL NETWORKS

A paper published in Photonic Network Communications 1 2

Borislav H. Simov, Jason P. Jue and Srini Tridandapani

Abstract

We introduce a new technique for providing security in a broadcast-and-select, wavelength-division-multiplexed (WDM) optical network. The approach provides privacy of communications by employing a novel challenge-response scheme and exploiting the tuning delay inherent in optical receivers. The proposed technique can be integrated easily into any existing WDM media-access-control (MAC) protocol that employs tunable receivers. The modified protocol would require every idle user, who is not scheduled to receive data, to tune in to a channel that does not contain sensitive data. A violation of the protocol can be detected with very high probability, and appropriate measures can be taken against the violator. The technique provides features that cannot be achieved with cryptography alone. Significant benefits of the proposed approach include the ability to detect security violations as they occur, and an efficient mechanism to provide privacy for multicast transmissions.

We develop two simple solutions to deal with different levels of attack: (1) eavesdroppers working alone, and (2) eavesdroppers working in collaboration. We compute the throughput of the two solutions, and we prove that they are optimal in the number of non-data (challenge and response) channels added.

---


2 An earlier version by Simov and Tridandapani was presented at the 1998 IEEE International Conference on Communications. [44]
4.1 Introduction

The Internet is rapidly becoming the next major world marketplace, with millions of financial transactions taking place daily. As the volume of financial data and other sensitive information being transmitted over the Internet continues to grow, the need for security is greater than ever. A network must provide adequate safeguards in order to ensure that sensitive data cannot be intercepted by parties other than the destination. Security is especially a concern in local area networks which utilize a broadcast medium, such as a WDM optical network based on a passive-star coupler.

For the most part, existing security solutions rely primarily on cryptography [38, 41]. With encryption, it is assumed that anyone could intercept the encrypted message, but decrypting the ciphertext will be impossible without the secret decryption key. A completely different paradigm for providing security involves designing the physical and media access layers of the network in such a way that untrusted users are prevented from intercepting the message in the first place. In this chapter, we will investigate such an approach to providing privacy at the media access layer. Our method relies on the fact that the physical hardware in a WDM local-area network gives us some control over the access to the channels. To the best of our knowledge, very little work has been done to incorporate privacy at the physical or media-access layer [1, 51] — our approach [41] is the first to integrate privacy and intrusion detection at the media-access layer.

In this chapter, we consider a broadcast-and-select wavelength division multiplexed (WDM) optical network and a corresponding media access protocol which allows N users to share K data channels, where \( K \leq N \). Each user has a transmitter which may either be fixed to a single channel or tunable over a range of channels, and a tunable receiver which can be tuned to any channel. We assume that the optical fiber is secure against outside intrusion. Moreover, we do not expect an attack that uses additional devices to tap into the data channels. The question is whether, under these assumptions, privacy can be guaranteed. Unfortunately, the following problem may occur. Let Alice, Bob and Eve be three of the N users in the system and suppose Alice is scheduled to send a message to Bob. At the same time Eve is idle, that is, she is not scheduled to receive any data. Then Eve may decide to tune her receiver to the channel on which Alice is sending a message to Bob. Thus Eve can perform an unauthorized read, and therefore the system does not guarantee privacy.

How can we deal with the problem described above? As already mentioned, pre-existing solutions involve cryptography [38, 4, 40]. Alice and Bob agree on a secret key in advance and then use it to encrypt and decrypt the messages before and after the transmission, respectively. However, there are a
few drawbacks to cryptographic schemes. First, there is a time overhead for encrypting and decrypting messages between Alice and Bob. This processing delay may not be tolerable for certain types of real-time traffic. Second, key management may be a non-trivial task. For example, key distribution may require either additional secure channels or long computations, depending on whether private or public key cryptography is used, respectively. In addition, keys may be forgotten, compromised, or may need to be renewed over time, which adds to the complexity of the problem. Furthermore, when providing security for multicast communications, the problem of distributing and revoking keys becomes an extremely complex issue, especially if the multicast group membership is changing dynamically. Finally, there is no easy way to detect an attack on the privacy of a message. If the key is unknowingly compromised, Alice and Bob will not be able to detect that their encoded messages are being intercepted.

In this chapter, we suggest a novel approach to privacy which does not use encryption; however, it may be combined with encryption to provide an even higher degree of privacy. In addition to the existing \(K\) data channels, we introduce a challenge channel and a response channel on which no private data is transmitted. The main idea is that every time Eve is idle (i.e., not scheduled to receive any data) she is required to read on the challenge channel, thus she is prevented from eavesdropping on a data channel. At time slot \(t\), a random bit string is sent to Eve on the challenge channel. Since Eve has only one receiver, she has to tune to the challenge channel; thus, she is not able to tune to a data channel and eavesdrop. On the next time slot, \(t+1\), Eve must send back the same random bit string that was sent to her as a challenge in time \(t\), otherwise she is in violation of the protocol.

In most existing security mechanisms, security violations are discovered after the fact, when a significant amount of damage has already been done. In the proposed scheme, violations are detected with high probability as they occur, allowing for immediate action to protect the system and to discipline the perpetrator. Once a violation is detected, a number of actions may be taken. An extreme approach is to switch off all communications in the network, thus avoiding further compromise of sensitive information. Another option is to halt only the sensitive transmissions, and to resume transmission once compliance with the protocol is detected over some arbitrary number of time slots. Off-line administrative actions against the perpetrators may be also considered.

One significant advantage of the proposed scheme is that it can provide privacy for multicast communication in an efficient manner. The trusted users in a multicast group can tune their receivers to the multicast channel, while untrusted users are simply forced to tune their receivers to a different channel. No significant modifications to the protocol are necessary.

Note that the correctness of our approach relies on the fact that the attackers have limited resources.
If, in the example above, Eve is allowed to use extra hardware, then she can add up to $K$ nontunable receivers (one for each data channel). Now she can use her original tunable receiver for the challenges that are presented to her, while at the same time she can eavesdrop on all $K$ channels using the nontunable receivers. Our protocol is not designed to counter such a committed hardware-oriented attack. We are assuming that Eve does not change the physical configuration of the system, yet she may decide to listen to someone else's conversation with the resources that she already has. A setting in which such conditions exist is, for example, a computer lab in an university. If there are surveillance cameras, this would deter users from changing the hardware configurations. However, it might still be possible for an user to modify the software in order to eavesdrop on the messages on the local network.

In this study, we will investigate a challenge-response approach to providing privacy in an optical broadcast network. Our primary measures in analyzing this approach will be the level of security provided, as well as the cost of security in terms of both the additional hardware required to implement the scheme and the degradation of network performance. The remainder of this chapter is organized as follows. In Section 4.2 we define in detail our assumptions about the channels and the underlying scheduling protocol. In Section 4.3, we show how to modify any existing scheduling protocol so that the new protocol can detect eavesdropping. We present two solutions which guarantee privacy against attacks by several attackers working individually, and several attackers working in collaboration, in Sections 4.4 and 4.5, respectively. In Section 4.6 we conclude the chapter and provide some directions for future work.

### 4.2 Assumptions about the Scheduling Scheme

We would like to make as few assumptions about the original scheduling protocol as possible. Our approach can therefore be applied to any existing protocol for multi-channel broadcast WDM networks. The protocol may employ deterministic scheduling in which every user has a predetermined channel and time slots to transmit [9, 14, 3]. The protocol will be a modified form of simple time-division multiplexing on the $K$ channels. Alternatively, the protocol may be reservation-based, such that the users "compete" for time slots on which they transmit, and the bandwidth allocated to each user over a period of time depends on the amount of information the user has to transmit [9, 8, 47, 33, 23]. Normally, protocols of the second type use at least one extra control channel on which reservations are made and resolved. Additional transmitters and receivers may be required for this control channel.

We assume that the propagation delay and the processing time are negligible and we also require that
the protocol is collision free (i.e., users know in advance that they are not chosen to transmit, thus they
do not interfere with the users who will transmit). Scheduling of transmissions is resolved early enough
so that all users have sufficient time to tune their receivers and transmitters to the correct channels.
Every user has one receiver which is tunable and one transmitter, which may or may not be tunable. In
addition, users have one or more transmitters and receivers that may be dedicated for scheduling on the
control channel. The tunability of the receiver is essential since our approach forces idle users to tune
out of the data channels. Therefore, we cannot apply our approach to a protocol in which users have
only fixed-tuned receivers. Finally, the protocol employs time slots of equal length, which correspond
to the transmission time of one packet, and transmissions on all channels are synchronized.

We denote the number of users in the system by \( N \) and the number of data channels by \( K \). The
scheduling scheme of the original protocol can be summarized by introducing two functions, \( \text{reads} \) and
\( \text{sends} \) which tell us at every time \( t \) whether a user is receiving or sending information and on which
channel.

\[
\text{reads}(i,t) = \begin{cases} 
  k & \text{if user } i \text{ is scheduled to receive on channel } k \ (1 \leq k \leq K) \text{ at time } t \\
  \text{nil} & \text{if user } i \text{ is not scheduled to receive at time } t
\end{cases}
\]

\[
\text{sends}(i,t) = \begin{cases} 
  k & \text{if user } i \text{ is scheduled to transmit on channel } k \ (1 \leq k \leq K) \text{ at time } t \\
  \text{nil} & \text{if user } i \text{ is not scheduled to transmit at time } t
\end{cases}
\]

Note that the semantic of the function \( \text{reads}(i,t) \) also captures multicasting. For example,

\[
\text{reads}(i_1,t) = \text{reads}(i_2,t) = \ldots = \text{reads}(i_j,t) \neq \text{nil}
\]

exactly when users \( i_1, \ldots, i_j \) are all scheduled to receive a data packet on the same channel at time \( t \).
For the rest of this chapter, however, we will focus only on point-to-point, i.e., unicast communications.

To further specify the operation of the protocol, we also define a function, \( \text{data}(k,t) \), which tells
us what is the data transmitted on channel \( k \) at time \( t \) (see Figure 4.1). It is useful to think about
\( \text{data}(k,t) \) as a reference to a location (channel \( k \) and time \( t \)) from which data can be read and to which
data can be written. This allows us to write \( \text{data}(k,t) \leftarrow a \) to denote that the value of \( a \) is written to
channel \( k \) at time \( t \), and \( b \leftarrow \text{data}(k,t) \) to denote that the value of \( b \) is read from channel \( k \) at time \( t \).
4.3 Secure Protocol

The main idea of the secure protocol is to exploit the tuning latency of the receivers in order to prevent eavesdropping. We want to guarantee that every idle user who is not receiving data is not tuned in to a data channel, since this would present an eavesdropping threat. We introduce a challenge channel and a response channel on which no sensitive data is transmitted. Every idle user is required to tune her receiver to the challenge channel. How can we guarantee that users comply? The execution of the protocol is supervised by a central authority, or principal. For every user, the principal sends a different challenge — a random string, over the challenge channel. In the next time slot, the principal expects every user to echo the same challenge, or a suitable function of the challenge, back on the response channel. Suppose that, at time $t + 1$, for a given user, the principal receives a reply that is different from the challenge that was sent to the user at time $t$. This is a definite indication that the protocol was violated by the user if we assume error-free transmissions. Upon discovery of a discrepancy, the principal is authorized to take appropriate actions such as disconnecting the guilty party or even shutting down the network.

On the other hand, if the principal receives the correct response, there are two possibilities. First, suppose that the user complies with the protocol: she tunes her receiver to the idle channel at time $t$, then she reads the challenge, and at time $t + 1$ she echoes it back to the principal. The tuning latency guarantees that she cannot read from a data channel at time $t$: there is not enough time for her to tune in to the challenge channel, read the challenge, then retune to a data channel and read a non-negligible portion of the data there. This is, in fact, the normal mode of operation of the secure protocol. If all users are complying with the challenge-response scheme, the responses received by the principal will not differ from the challenges that were sent.

However, there is a second possibility. Even if at time $t + 1$ the principal receives the same response
as the challenge that was sent at time $t$, the principal cannot be absolutely confident that no violation of the protocol has occurred. Consider the following scenario. Suppose a user, Eve, is idle at time $t$, yet she tunes in to a data channel. At that time she cannot receive the challenge, since it is only available on the challenge channel. However, Eve may try to guess the challenge, and at time $t + 1$ she can send back a random string, hoping that it coincides with the challenge. If Eve made a lucky guess, the principal cannot detect that Eve is in violation. However, the probability of her guessing a random string of $n$ bits, is $2^{-n}$. For example, even for a challenge as small as one byte ($n = 8$), Eve remains unnoticed with probability less than 0.4%. Thus, the probability of the principal missing a violation is negligible.

In the rest of this section we will explain why we also need to modify the data channels and how we accomplish this. Our primary motivation for introducing challenge and response channels was to prevent idle users from eavesdropping on the data channels. However, we cannot assume that a user who is not idle (one who is scheduled to receive a message) follows the rules of the protocol either. This idea is illustrated in the following attack on the original protocol. Suppose Eve is scheduled to receive a very long message from Alice. At the beginning of the message Eve realizes that she already has a copy of the message and therefore she does not have to receive the remainder. At this time Eve can roam around the rest of the channels eavesdropping, while Alice continues to transmit. No other user, not even the principal, can notice that Eve is not listening on the scheduled channel since there is no feedback expected from her. In the worst case Eve may tune in to another channel and listen to a message from Carol to Dan without their consent.

To prevent the attack described above, the principal sends challenges not only to the idle users but also to the ones that are scheduled to receive on a data channel. The latter must read the data as well as the challenge on the same channel. Also, users who are scheduled to transmit in a slot will piggyback their response with the data. More precisely, we define $D_k(t)$, the slot on channel $D_k$ ($1 \leq k \leq K$) at time $t$, to contain three fields:

$$\begin{array}{c|c|c}
D_k(d, t) & D_k(c, t) & D_k(r, t) \\
\end{array}$$

- **data, $D_k(d, t)$**

  This field corresponds to the data transmitted in the original protocol, that is,

  $$D_k(d, t) = data(k, t)$$

  At time $t$, user $i$, for whom $k = sends(i, t)$, writes the data to channel $D_k$ and user $j$, for whom $k = reads(j, t)$, receives the data from the channel.
• **challenge,** $D_k(c, t)$

The principal writes in this field a challenge (a random string). User $j$ who is supposed to tune in to channel $D_k$ at time $t$ in order to read data (that is, $k = reads(j, t)$) is also responsible for reading the challenge by the principal. This challenge is used in the next time slot, $t + 1$, to verify that user $j$ was indeed tuned to the proper channel.

• **response,** $D_k(r, t)$

In this field, user $i$ who is supposed to send data at time $t$ on channel $D_k$ (that is, $k = sends(i, t)$) writes her response to the principal, namely, the challenge that she received at time $t - 1$. The principal reads the response and compares it to the challenge to verify it.

The three parts of the slot are bit-interleaved in order to prevent a user switching back and forth between a data and response channel within the same time slot.

### 4.4 Solution against Attacks by Eavesdroppers Working Alone

In this section we provide a solution against one or more eavesdroppers who are working alone. From now on, we will refer to this solution as Solution 1. Solution 1 is secure against attacks by eavesdroppers working individually. However, in Section 4.5, we will show that Solution 1 is vulnerable to an attack by several eavesdroppers working in collaboration. We will then show an improved version which is secure against collaborating attackers as well.

#### 4.4.1 Configuration of the channels in Solution 1

We have modified the data channels as described in the previous section. Along with the $K$ data channels $D_1, D_2, \ldots, D_K$, we have added one challenge channel, $C$, and one response channel, $R$ (see Figure 4.2). The challenge channel consists of time slots of equal length. Every time slot $C(t)$ is itself divided into $N$ equal parts, with each part equal to the size of the challenge from the principal. We can think of $C(t)$ as an array indexed by the number of each user. We denote the $i$-th part of channel $C$ at time $t$ as $C(i, t)$; thus the slot $C(t)$, comprising $N$ challenges on $C$ may be written as follows:

| $C(1, t)$ | $C(2, t)$ | $\ldots$ | $C(N, t)$ |

Only the principal writes on the challenge channel. Every time slot of the challenge channel is bit interleaved.
### The channels in Solution 1 (Individual Eavesdroppers)

<table>
<thead>
<tr>
<th>Channel</th>
<th>Time $t$</th>
<th>Time $t+1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_1$</td>
<td>$D_1(d,t)$ $D_1(c,t)$ $D_1(r,t)$</td>
<td>$D_1(d,t+1)$ $D_1(c,t+1)$ $D_1(r,t+1)$</td>
</tr>
<tr>
<td>$D_2$</td>
<td>$D_2(d,t)$ $D_2(c,t)$ $D_2(r,t)$</td>
<td>$D_2(d,t+1)$ $D_2(c,t+1)$ $D_2(r,t+1)$</td>
</tr>
<tr>
<td>$D_K$</td>
<td>$D_K(d,t)$ $D_K(c,t)$ $D_K(r,t)$</td>
<td>$D_K(d,t+1)$ $D_K(c,t+1)$ $D_K(r,t+1)$</td>
</tr>
<tr>
<td>$C$</td>
<td>$C(1,t)$ $\ldots$ $C(N,t)$</td>
<td>$C(1,t+1)$ $\ldots$ $C(N,t+1)$</td>
</tr>
<tr>
<td>$R$</td>
<td>$R(1,t)$ $\ldots$ $R(N,t)$</td>
<td>$R(1,t+1)$ $\ldots$ $R(N,t+1)$</td>
</tr>
</tbody>
</table>

The response channel has an identical structure. We denote the $i$-th part of channel $R$ at time $t$ as $R(i,t)$; thus the slot $R(t)$, comprising $N$ responses on $R$ may be written as follows:

| $R(1,t)$ $R(2,t)$ $\ldots$ $R(N,t)$ |

Every user has one tunable receiver and one tunable transmitter. A slight modification of the protocol can be made to accommodate fixed-tuned transmitters. The principal must receive on and transmit to all the data channels. Also, the principal has to transmit to the challenge channel and receive on the response channels. This implies that the principal should have $K + 1$ fixed-tuned receivers and $K + 1$ fixed-tuned transmitters. These transmitters and receivers, as well as the additional challenge and response channels, are the primary fixed cost incurred by our modified approach.

#### 4.4.2 The protocol

How does the protocol work? At time $t$, every user has a specific location, including a channel and a field in the time slot, on which to transmit, and another specific location on which to receive. Users who are not scheduled to receive on any data channel must receive on the challenge channel. Users who are not scheduled to transmit on any data channel must transmit on the response channel. At time $t$, the principal sends random bits as challenges to each user on the channel on which the user’s receiver
should be tuned. The principal expects the user to respond at time \( t + 1 \) with the same sequence of random bits that the user received at time \( t \).

First, we point out that the secure protocol does not alter the scheduling of data transmissions as defined in the original protocol. That is

\[
D_k(d, t) = data(k, t)
\]

On the other hand, we define

\[
\text{challenge}(i, t) = \begin{cases} 
D_k(c, t) & \text{if } reads(i, t) = k \\
C(i, t) & \text{if } reads(i, t) = \text{nil}
\end{cases}
\]

to be the location at which user \( i \) expects a challenge at time \( t \). If the user is scheduled to receive data, the challenge is located in the corresponding field, \( D_k(c, t) \), in the scheduled data channel, \( D_k \). If the user is not scheduled to receive data, then she has to listen to the challenge in the corresponding location, \( C(i, t) \) on the challenge channel \( C \).

We also define

\[
\text{response}(i, t) = \begin{cases} 
D_k(r, t) & \text{if } sends(i, t) = k \\
R(i, t) & \text{if } sends(i, t) = \text{nil}
\end{cases}
\]

as the location in which a user is supposed to send her response. Note that, just like the challenges, the responses may be either on one of the data channels or on the response channel, depending on whether or not the user is scheduled to transmit data at that time.

The actions for user \( i \) (\( 1 \leq i \leq N \)) at time \( t \) are described in Figure 4.3. In the two conditional statements, the user commits her transmitter to a specific channel and her receiver to a specific channel, which makes switching back and forth impossible.

The actions of the principal at time \( t \) are described in Figure 4.4. The principal ensures the main property of the protocol, that is, for every user, \( i \), and for every time slot, \( t \).

\[
\text{response}(i, t + 1) = \text{challenge}(i, t)
\]

If this property is violated for some pair \((i, t)\), the principal can detect the violation by the end of time slot \( t + 1 \). At time \( t + 2 \), the principal already knows that user \( i \) is not complying with the protocol and appropriate actions can be taken.
User(i,t)
// The procedure describes the actions of user i at time t
begin
  static 0
  // Assume that 0 stores the value of the
  // challenge from time t-1
  s ← sends(i,t)
  if (s ≠ nil)
    then
      tune transmitter to channel D_s
      D_s(d,t) ← data(s,t)
      D_s(r,t) ← 0
    else
      tune transmitter to channel R
      R(i,t) ← 0
  r ← reads(i,t)
  if (r ≠ nil)
    then
      tune receiver to channel D_r
      data(r,t) ← D_r(d,t)
      0 ← D_r(c,t)
    else
      tune receiver to channel C
      0 ← C(i,t)
end

Figure 4.3 Actions of a user in Solution 1

4.4.3 Analysis of Solution 1

The protocol is successful against an attack by a single eavesdropper. At time slot t, Eve must
commit to tuning her receiver to a single channel due to our assumption about the tunability delay. If
Eve is idle (reads(Eve,t) = nil), then there are two choices for Eve. First, if Eve tunes her receiver
to channel C, then she cannot eavesdrop. Second, if Eve decides to tune to a data channel, then she
is not able to receive the challenge sent to her by the principal. Therefore, at time t + 1 she has to
guess a response, and the chance of her remaining undetected after time t + 1 is $2^{-n}$ for a challenge of
length n bits. A similar analysis applies in the case when Eve is scheduled to receive on one of the data
channels, $D_i$, and she decides to eavesdrop on a different data channel, $D_j$ ($i ≠ j$). The challenge for
Eve is sent on channel $D_i$, interleaved with the data, so if Eve tunes her receiver to channel $D_j$, then
she is not able to receive the challenge, and at time t + 1, she has to guess the response to the principal.

To compute the throughput for Solution 1, we assume a normalized capacity of 1 per channel. The
data, challenge and response channels then have a total capacity of $K + 1 + 1 = K + 2$. However, data
is transmitted only on the $K$ data channels, each having a throughput of $1 - 2/N$. (On every data
Principal(t)
//The procedure describes the actions of the principal at time t > 0.
//At time t=0, the principal sends challenges but does not check responses
begin
  0[N] // This array stores the old challenges
  for i = 1 to N do
    if response(i,t) ≠ 0[i]
      then there is a problem with node i
    for i = 1 to N do
      0[i] ← random number
      challenge(i,t) ← 0[i]
end

Figure 4.4 Actions of the principal in Solution 1 and Solution 2

channel $D_i$, the challenge and response fields, $D_i(c,t)$ and $D_i(r,t)$, are not used for data. Each field has a capacity of $1/N$. Recall from Figure 4.2 that the length of a challenge is $1/N$ of a slot.) Overall, the throughput $T_1$ is given by:

$$T_1 = \frac{K(1 - 2/N)}{2 + K} = \frac{K(N - 2)}{N(K + 2)}.$$  

Thus, in the extreme case when all $N$ users have to share a single data channel, that is, $K = 1$, the throughput is just $1/3$. On the other hand, for larger values of $K$ we get a throughput close to 1 (see Figure 4.5), which means that, in Solution 1, we have added security without noticeably depreciating the throughput performance of the underlying scheduling protocol. Note that asymptotically the throughput of Solution 1 is close to 1, which suggests that the approach is scalable in the number of users, $N$, and the number of channels, $K$.

It is interesting to note that Solution 1 is secure against a simultaneous attack by any number of non-collaborating users. However, it is vulnerable to an attack by two or more collaborating eavesdroppers. Suppose that there are several users who have decided to cooperate in order to allow one of them to eavesdrop. For simplicity, we assume that there are just two collaborators: Eve and Eric. Eve tunes in to her favorite data channel and starts to eavesdrop. Normally, she would be detected within two time slots, but now Eric covers for her. If Eric and Eve are both idle, then their challenges are received on $C$ and they have to respond on $R$. However, Eric can read both the challenge directed to him and the challenge addressed to Eve. Furthermore, Eric can respond for himself and for Eve. The principal has no means of detecting that someone else has served as a proxy for Eve, and her violation remains unnoticed. A solution to this problem will be addressed in Section 4.5.
4.4.4 Application of Solution 1 to an existing protocol

We now illustrate how Solution 1 may be applied to an existing media access control protocol. We choose the dynamic time-wavelength division multiple access (DT-WDMA) protocol described in [8].

In DT-WDMA, nodes are connected to a passive-star coupler which is an optical broadcast medium. The system requires a single control channel and $N$ data channels, where $N$ is the number of nodes. Each node is assigned its own unique channel on which to transmit data packets, but may receive data packets on any of the data channels. Thus, each node requires a fixed transmitter and a fixed receiver to access the control channel, a fixed transmitter to access a single data channel, and a tunable receiver to access all data channels. Time is divided into slots, with a slot length equal to the packet transmission time plus the receiver tuning time. The control channel is accessed using a TDM-based approach. Each slot on the control channel contains $N$ minislots, one for each of the $N$ nodes in the network. When a node $i$ has a packet to transmit, it will send a request in its assigned minislot. The request indicates the source node, the destination node, and a priority level. After a node transmits its request in a minislot in time slot $t$, it will transmit its data packet in the following time slot ($sends(i, t + 1) = i$) on its assigned channel. Since each node continuously monitors the control channel, a node will be able to determine whether it will receive a packet in the following time slot and the channel to which it should tune its receiver. If node $j$ receives a request from node $i$ at time $t$, it will set $reads(j, t + 1) = i$ and
tune its receiver to channel \( i \) in slot \( t + 1 \). If more than one node transmits a request on the control channel to the same destination node, then the destination node will receive the packet with the highest priority. If the priorities are equal, then the destination node will choose one of the transmissions based on some deterministic algorithm, such as choosing the node with the lowest index.

Modifying this protocol to accommodate Solution 1 requires one additional channel to serve as a challenge channel and one additional node to serve as the principal. The principal node requires \( N+1 \) fixed transmitters for the data channels and the challenge channel, and \( N+1 \) fixed receivers for the data channels and the control channel. Since the principal monitors the control channel, it will know on which channel each of the other nodes' receiver should be tuned in a given time slot. The principal can then transmit the challenge to the node on the appropriate channel. If a node is not scheduled to receive a packet in a given time slot, then its receiver should be tuned to the challenge channel. Each node will respond to the challenge in the following time slot on the data channel on which its transmitter is fixed. Note that, for this particular protocol, there does not exist a problem of collaborating eavesdroppers. Since each node must respond to the challenge on its own channel, and since each node only has a single fixed transmitter for the data channels, one node cannot respond to a challenge for another node.

4.5 Solution against Attack by Two or More Collaborating Eavesdroppers

In this section we describe a more elaborate version of the secure protocol called Solution 2. This protocol prevents an attack by two or more collaborating eavesdroppers as was described in Subsection 4.4.3. The main approach for improving Solution 1 is to increase the number of challenge and response channels, and to assign each user a distinct challenge-channel/response-channel pair. Since no two users share both the same challenge channel and the same response channel, one user will not be able to respond to another user's challenge while also responding to his or her own challenge. Details on Solution 2 follow.

4.5.1 Configuration of the channels in Solution 2

We assume that the number of users, \( N \), is a square of an integer, let \( M^2 = N \). This can be done without loss of generality: if \( N \) is not a square, we can pick \( M = \lceil \sqrt{N} \rceil \). The \( K \) data channels \( D_1, D_2, \ldots, D_K \) are defined exactly as in Solution 1. However, now instead of a single pair of challenge-response channels, there are \( M \) challenge and \( M \) response channels, denoted as \( C_1, C_2, \ldots, C_M \) and \( R_1, R_2, \ldots, R_M \), respectively. As shown in Fig. 4.6, every challenge channel \( C_i \) (\( 1 \leq i \leq M \)) is divided
Figure 4.6 The channels in Solution 2 (Collaborating Eavesdroppers).

<table>
<thead>
<tr>
<th>Channel</th>
<th>Time t</th>
<th>Time t + 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_1$</td>
<td>$D_1(d,t)$ $D_1(c,t)$ $D_1(r,t)$</td>
<td>$D_1(d,t+1)$ $D_1(c,t+1)$ $D_1(r,t+1)$</td>
</tr>
<tr>
<td>$D_2$</td>
<td>$D_2(d,t)$ $D_2(c,t)$ $D_2(r,t)$</td>
<td>$D_2(d,t+1)$ $D_2(c,t+1)$ $D_2(r,t+1)$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$D_K$</td>
<td>$D_K(d,t)$ $D_K(c,t)$ $D_K(r,t)$</td>
<td>$D_K(d,t+1)$ $D_K(c,t+1)$ $D_K(r,t+1)$</td>
</tr>
<tr>
<td>$C_1$</td>
<td>$C_1(1,t)$ ... $C_1(M,t)$</td>
<td>$C_1(1,t+1)$ ... $C_1(M,t+1)$</td>
</tr>
<tr>
<td>$C_2$</td>
<td>$C_2(1,t)$ ... $C_2(M,t)$</td>
<td>$C_2(1,t+1)$ ... $C_2(M,t+1)$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$C_M$</td>
<td>$C_M(1,t)$ ... $C_M(M,t)$</td>
<td>$C_M(1,t+1)$ ... $C_M(M,t+1)$</td>
</tr>
<tr>
<td>$R_1$</td>
<td>$R_1(1,t)$ ... $R_1(M,t)$</td>
<td>$R_1(1,t+1)$ ... $R_1(M,t+1)$</td>
</tr>
<tr>
<td>$R_2$</td>
<td>$R_2(1,t)$ ... $R_2(M,t)$</td>
<td>$R_2(1,t+1)$ ... $R_2(M,t+1)$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$R_M$</td>
<td>$R_M(1,t)$ ... $R_M(M,t)$</td>
<td>$R_M(1,t+1)$ ... $R_M(M,t+1)$</td>
</tr>
</tbody>
</table>

A time slot $C_i(t)$ on channel $C_i$ is itself subdivided into $M$ equal parts. Similar to Solution 1, we can think of $C_i(t)$ as an array indexed from 1 to $M$.

$$C_i(t) = \begin{bmatrix} C_i(1,t) & C_i(2,t) & \ldots & C_i(M,t) \end{bmatrix}$$

$C_i(j,t)$ is the $j$-th part of $C_i(t)$ and it represents a single challenge sent by the principal. As in Solution 1, this array representation is only conceptual, because in reality the bits of all fields of $C_i(t)$ are interleaved. The response channels are divided in an identical way, so that $R_i(t)$ ($1 \leq i \leq M$) denotes the slot of response channel $R_i$ at time $t$, consisting of $M$ parts:

$$R_i(t) = \begin{bmatrix} R_i(1,t) & R_i(2,t) & \ldots & R_i(M,t) \end{bmatrix}$$

$R_i(j,t)$ is the $j$-th part of $R_i(t)$ and represents a single response sent to the principal.

Every user has a tunable receiver and a tunable transmitter. The principal needs $K + M$ fixed-tuned
transmitters and $K + M$ fixed-tuned receivers.

### 4.5.2 The protocol

As already mentioned, the data fields are defined exactly as in Solution 1. To complete the description of Solution 2, it suffices to redefine the semantics of the functions challenge and response.

```plaintext
User(i,t)
// The procedure describes the actions of user i at time t
begin
static 0
// Assume that 0 stores the value of the
// challenge from time t-1
s ← sends(i,t)
if (s ≠ nil)
    then
        tune transmitter to channel $D_s$
        $D_s(d,t) ← data(s,t)$
        $D_s(r,t) ← 0$
    else
        tune transmitter to channel $R_{g(i)}$
        $R_{g(i)}(f(i),t) ← 0$
    end
r ← reads(i,t)
if (r ≠ nil)
    then
        tune receiver to channel $D_r$
        data(r,t) ← $D_r(d,t)$
        0 ← $D_r(c,t)$
    else
        tune receiver to channel $C_{f(i)}$
        0 ← $C_{f(i)}(g(i),t)$
end
```

Figure 4.7 Actions of an user in Solution 2

For a fixed $M$, let us arrange the users as entries in a $N \times M$ matrix, starting from the top left entry and going from left to right and from top to bottom, such that every user is in a distinct row and distinct column of the matrix. For example, if $N = 9$ and $M = 3$, the matrix looks like:

```
1 2 3  
4 5 6  
7 8 9  
```

We introduce functions, which for any user $i$, $(1 \leq i \leq N)$, help us to address the user as an entry
in the $M \times M$ matrix. For $1 \leq a \leq M$ and $1 \leq b \leq M$ define

$$h(a, b) = M(a - 1) + b \quad f(h(a, b)) = a \quad g(h(a, b)) = b$$

Given row $a$ and column $b$ in the matrix, $h(a, b)$ returns the index of the user corresponding to that entry. Also, for user $i$, $f(i)$ and $g(i)$ compute the corresponding row and column indices, respectively. In the example above, $h(3, 2) = 8$, $f(7) = 3$, and $g(6) = 3$.

We denote by $\text{challenge}(i, t)$ the location at which user $i$ expects a challenge at time $t$. If the user is scheduled to receive data, then the challenge is located in the corresponding field of the scheduled data channel. If the user is not scheduled to receive data, then the challenge is in part $g(i)$ of the appropriate challenge channel, $C_{f(i)}$.

$$\text{challenge}(i, t) = \begin{cases} D_k(c, t) & \text{if } \text{reads}(i, t) = k \\ C_{f(i)}(g(i), t) & \text{if } \text{reads}(i, t) = \text{nil} \end{cases}$$

We also define $\text{response}(i, t)$ as the location at which user $i$ is supposed to send her response. If the user is scheduled to transmit data, then the response is located in the corresponding field of the scheduled data channel. If the user is not scheduled to transmit data, then her response is in part $f(i)$ of the appropriate response channel, $R_{g(i)}$.

$$\text{response}(i, t) = \begin{cases} D_k(r, t) & \text{if } \text{sends}(i, t) = k \\ R_{g(i)}(f(i), t) & \text{if } \text{sends}(i, t) = \text{nil} \end{cases}$$

Note that, if user $i$ is neither scheduled to receive nor scheduled to transmit data, the challenge is at location $C_{f(i)}(g(i), t)$, while the response is at location $R_{g(i)}(f(i), t)$. Since for every $i$ and $j$, such that $1 \leq i < j \leq M^2$ we have $f(i) \neq f(j)$ or $g(i) \neq g(j)$, then it follows that no two users $i$ and $j$ can both share the same challenge and the same response channels: therefore, no user can cover for another.

The actions of every user $i$, where $1 \leq i \leq N$, are described in Figure 4.7. The actions of the principal are described in Figure 4.4. Note that the principals in Solution 1 and Solution 2 have different definitions of the functions $\text{challenge}$ and $\text{response}$: therefore, the two principals perform different actions even though their code looks identical.
4.5.3 Analysis of Solution 2

The key observation is that Solution 2 guarantees that no two users expect challenges and send back responses on the same pair of channels. We can show that, if Eve and Eric want to collaborate and eavesdrop, then their attempt will be detected. Without loss of generality, assume that Eve eavesdrops and Eric tries to send back a correct response on her behalf. At least one of the following is true: Eric is assigned a challenge channel different from Eve's, or Eric is assigned a response channel different from Eve's.

First, if Eric's challenge channel is different from Eve's, then he can read only one of the challenges and must guess the second challenge. With high probability his violation is detected in the next time slot. Second, if Eric's response channel is different from Eve's, then he cannot send back to the principal both his and Eve's response. Again, the violation is detected. The argument above can be generalized by induction to any number of collaborating eavesdroppers.

The throughput of Solution 2 is less than that of Solution 1. This is quite natural since we had to prevent attacks by collaborating eavesdroppers. For this solution, we had to increase the number of challenge and response channels — channels on which no data is sent. To compute the throughput, let $M = \sqrt{N}$ be the number of users that share the same challenge or response channel. If we assume a capacity of 1 for a channel, then the entire system, including all the data, challenge, and response channels, has a capacity of $K + M + M = K + 2M$. However, the $K$ data channels only have a throughput of $1 - 2/M$. Recall that on each data channel $D_i$, we have the challenge and response fields, $D_i(c, t)$ and $D_i(r, t)$, each of which has length $1/M$ of the data slot. For the throughput, $T_2$ we get:

$$T_2 = \frac{K(1 - 2/M)}{2M + K} = \frac{K(M - 2)}{M(K + 2M)}$$

which is approximately $c/(c + 2)$ for $K =c\sqrt{N}$ and $M >> 1$.

Note that for all $i$, $(1 \leq i \leq \sqrt{N})$, we can combine the challenge channel $C_i$ and the response channel $R_i$ into a single challenge-response channel $C_R_i$. With all other things being equal, this will decrease the number of additional channels in Solution 2 from $2\sqrt{N}$ to $\sqrt{N}$. The drawback is that we also decrease by half the size of the challenges, thus increasing the chance for a correct guess of a response. However, if the original challenge was $n$ bits, the size of the new one will be $n/2$ and then the probability that a guess will be unnoticed is just $2^{-n/2}$ which is still sufficiently low. For the throughput
Throughputs of Solution 2, modified Solution 2, and Solution 1

The comparison of the throughputs for the different solutions is shown in Figure 4.8. The upper surface represents the throughput of Solution 1, $T_1$. The lower surface represents the throughput of Solution 2, $T_2$. Finally, the throughput of the modified Solution 2, $T'_2$, is in the middle. Clearly, there is a trade-off between the throughput of a solution and the probability of detecting an eavesdropper.

An interesting question is whether it is possible to implement a challenge-response protocol like Solution 2, which is secure against collaborating eavesdroppers, and which uses fewer than $2\sqrt{N}$ additional channels (the number used in Solution 2). For the purpose of this discussion, we define a protocol to be secure, if the probability that eavesdropping will not be detected does not exceed $2^{-n}$, where $n$ is the size of the challenge. In Appendix B we show that at least $2\sqrt{N}$ additional channels are required and, in that sense, Solution 2 is optimal, i.e., no other protocol with the same functionality can use fewer channels. Note that the existence of the modified Solution 2 (with only $\sqrt{N}$ additional channels, does not contradict this result since each of the channels $CR_i$ in the modified Solution 2 in fact represents...
a pair of multiplexed channels — one for challenges and one for responses.

4.5.4 Application of Solution 2 to an existing protocol

Let us now apply Solution 2 to an existing media access protocol. Consider a variation of the DT-WDMA protocol in which the number of data channels, $K$, is less than the number of users, $N$. Each node is equipped with a tunable transmitter rather than a fixed transmitter to access the data channels. The format of the control channel remains the same. A request message, in addition to the source, destination, and priority information, also specifies the channel on which the transmission is to take place. If more than one node requests to transmit on the same channel in a given slot, the request with the highest priority is allowed to transmit. A deterministic algorithm can be used to break further ties.

In order to apply Solution 2 to this protocol, we require an additional $2\sqrt{N}$ channels. Half of these channels will be allocated as challenge channels, while the other half will be allocated as response channels. Each node is assigned one challenge channel and one response channel in such a way that no node shares the same challenge-channel/response-channel pair. We also need to add a principal which is equipped with $K + \sqrt{N}$ fixed transmitters for the data and challenge channels and $K + \sqrt{N} + 1$ fixed receivers for the data, response, and control channels. Since the principal only needs to listen to the control channel, it does not require an extra transmitter for the control channel. A node which is scheduled to transmit or receive a packet in a given time slot will tune its transmitter or receiver respectively to the appropriate channel. Nodes which are not scheduled to transmit must tune their transmitters to the assigned response channel, while nodes which are not scheduled to receive must tune their receivers to the assigned challenge channel. The principal will send a challenge to a node either on a data channel or on a challenge channel, depending on whether the node is scheduled to receive a packet or not. In the following time slot, the principal will expect a response on either a data channel or on a response channel, depending on whether or not the node is scheduled to transmit a data packet.

4.6 Conclusions and Future Work

In this chapter, we have developed a new method for detecting and preventing unauthorized disclosure of information at the media access level in a multi-channel network, such as a broadcast-and-select WDM optical network. Our approach is not restricted to a single protocol. Rather, it is a methodology which can be applied to any collision-free media access control protocol which uses tunable receivers.
The approach presented does not use cryptography; instead, it relies on the resource limits of potential attackers and on a challenge-response scheme to guarantee that, at any time, idle users are not eavesdropping on the data channels. If a user does not comply with the protocol, there is a very high probability that the violation will be detected, and appropriate measures can be taken. We presented a solution for attacks by individual eavesdroppers as well as a solution for eavesdroppers working in collaboration, and illustrated how these solutions can be applied to existing protocols. For the latter approach (Solution 2) it was shown that at least $2\sqrt{N}$ additional channels are required as challenge and response channels.

Since, in the proposed scheme, an attacker is able to intercept at least one slot of data before any action can be taken, a possible approach for improving the scheme is to spread the contents of a private message over a number of slots [51, 1], such that the message is only compromised if more than a certain number of slots are intercepted. Investigation of such an improvement is a topic for future research. Further work is also required to define the appropriate action to take once a violation is detected. One possible action is to halt the transmission of sensitive information until compliance with the protocol has been detected over a number of slots. Algorithms need to be developed to determine how many slots a user should wait before resuming the transmission of private data.

Another interesting question that needs to be addressed is how the principal can get truly random bits for the challenges. Indeed, truly random bits are generated very slowly, thus they are impractical in real applications. Fortunately, there are very fast generators for pseudorandom bits. Although as of today no algorithm for pseudorandom bits is proven to be secure, the ones that are used in practice have good random properties which makes our method feasible. A second issue is whether, if we have a reliable source of pseudorandom bits, it is easier to implement cryptography using a method similar to Shannon's "one-time pad" [40]. However, such an approach would require the sender and the receiver to share the same sequence of random bits, that is, create the same sequence at two different locations. On the other hand, in our method the random bits are created in a single location by the principal and they do not have to be shared.

In Solution 2, we assumed that every idle user receives challenges and sends responses on a distinct pair of channels. It would be interesting to explore whether randomization of the challenge and response channels can provide privacy against collaborators while using fewer than $2\sqrt{N}$ additional channels, where $N$ is the total number of users.

Although we consider our scheme and encryption to be complementary techniques for providing privacy, it is useful to compare the two approaches. Cryptography can provide a high level of privacy by
making it difficult for an attacker to decrypt an encrypted message. However, conventional cryptography offers no satisfactory method to detect possible attacks on the privacy of a transmission and no means to determine that a message has been compromised. Also, as we have mentioned, cryptography requires additional computation time for encryption and decryption, which may not be tolerable for real-time applications, and key management may become cumbersome. On the other hand, our scheme provides a mechanism for detecting and identifying possible eavesdroppers, and the scheme can provide information as to whether or not data has been transmitted privately. The drawback is that the scheme may allow a limited amount of private data to reach the attacker before preventive measures can take place. The proposed scheme also requires an additional trusted server to act as a principal as well as additional bandwidth for the challenge channels and the response channels. We see that both the proposed scheme and encryption have their own strengths and weaknesses. By combining these two schemes, not only can we achieve a high degree of privacy, but we can also identify and take actions against potential attackers.

A further direction of research may be the development of a hybrid protocol, which can switch dynamically between two modes — one without protection of privacy, and one in which privacy is guaranteed. A good solution would be able to use the maximum bandwidth in the unprotected mode, while upon request and suitable payment, it will have security in the private mode.
5 APPROXIMATE BOUNDS AND EXPRESSIONS FOR THE LINK UTILIZATION OF SHORTEST-PATH MULTICAST NETWORK TRAFFIC

A paper published in Performance Evaluation
Borislav H. Simov, Srini B. Tridandapani and Michael S. Borella

Abstract

Multicast traffic refers to information with one source node, but many destination nodes. Rather than setting up individual connections between the source node and each destination node, or broadcasting the information to the entire system, multicasting efficiently exploits network capacity by allowing the source node to transmit a small number of copies of the information to mutually-exclusive groups of destination nodes. Multicasting is an important topic in the fields of networking (teleconferencing, video-conferencing, local-area network interconnection) and computer architecture (cache coherency, multiprocessor message passing). In this chapter, we derive expressions for the minimum cost (in terms of link utilization) of shortest-path multicast traffic in arbitrary tree networks. Our results provide a theoretical best-case scenario for link utilization of multicast distribution in tree topologies overlaid onto arbitrary graphs. In real networks such as the Internet MBONE, multicast distribution paths are often tree-like, but contain some cycles for purposes of fault tolerance. we find that for the shuffle-net and hypercube regular topologies, our expression provides a good prediction of the cost (in terms of link utilization) of multicast communication.

5.1 Introduction

Multicast traffic refers to information with one source node, but many destination nodes. Rather than setting up individual connections between the source node and each destination node, or broadcasting the information to the entire system, multicasting efficiently exploits network bandwidth by allowing the source node to transmit a small number of copies of the information to mutually-exclusive groups of destination nodes. Multicasting is an important issue in both the networking and computer architecture fields. Current and future networks will require support for the selective broadcasting of bandwidth-intensive information, such as digital audio, still pictures, and full-motion video. Multiprocessor architectures require efficient multicasting algorithms for message passing and cache coherency. Regardless of context, it is essential to deliver multicast traffic in a timely fashion, preferably with the smallest delay possible.

We provide a theoretical lower bound on the number of links required to distribute a multicast transmission to random destinations in a tree topology. In order to support multicast communication in arbitrary networks that were designed for unicasting only, a multicast tree is overlaid onto the network. In real networks such as the Internet MBONE, multicast distribution paths are often tree-like, but contain some cycles for purposes of fault tolerance.

In the Internet, multicasting is implemented through the MBONE [6], a virtual topology constructed on top of point-to-point links. The MBONE is used to distribute pre-recorded or live audio and video content in a bandwidth-friendly manner. Connectivity is assigned in an ad hoc manner – sites are connected to topologically-close upstream neighbors – resulting in an essentially random graph for which each node may have between one and five or so edges. The majority of MBONE routers have three or four edges. The overall MBONE graph is rather tree-like, with a small number of redundant paths between any source and destination. Routing in the MBONE is dynamic and changes based on the underlying IP routing decisions being made on the network layer.

Regular topologies such as the shufflenet [19] and hypercube [39], have been studied as interconnection schemes for local-area networks and multiprocessors. The multicasting protocols proposed for these networks ([5] and [24], respectively) utilize shortest-path routing. Due to the regularity of the networks, routing can usually be achieved in a distributed manner and requires a very small amount of processing per hop. In particular, the shufflenet and related graphs [46] [22] are very attractive and popular as virtual topologies for multi-channel optical LANs [34].

The approximate lower bound derived in this chapter is a theoretical result which can be directly
applied to any regular topology. In particular, we derive expressions for the smallest possible number of links traversed by multicast information in a $d$-regular graph (such as the shufflenet or hypercube) using shortest-path routing. This result allows a protocol designer or network engineer to evaluate the link utilization of their protocols against an idealized case. By setting $d$ to the maximum number of links per router, this result can be used as a loose lower bound for required link capacity of multicasts in the MBONE or any arbitrary tree-like topology where there is a single path between any source-destination pair. Our approach is based on generalizing the Moore bound, which is a well-established lower bound on the diameter of any $d$-regular graph.

This lower bound is a new and fundamental theoretical result, in that it provides mathematical confirmation of our intuition regarding the efficiency of multicast communication over pure unicast and broadcast mechanisms, and serves as a point of reference for the efficiency of multicast protocols in trees and regular graphs.

This chapter is organized as follows. Section 5.2 discusses Moore graphs, derives the Moore bound for directed and undirected graphs and serves as the point of departure for our approximate lower bound. Section 5.3 derives both recursive and iterative expressions for the expected cost of minimum-delay multicasting in a complete, arbitrary-degree tree with arbitrary multicast size. $m$. Section 5.4 derives recursive and iterative expressions for computing a lower bound on the expected cost of minimum-delay multicasting in an arbitrarily-sized network with arbitrary-degree. Section 5.5 provides numerical results for graphs with varied degree and multicast set size.

### 5.2 Moore Graphs

This section serves as a mini-tutorial on the issues of Moore graphs, which are the theoretical basis of our approximate lower bound. We develop the relationship between Moore graphs and the Moore bound, and present the Moore bound equations for both directed and undirected graphs. We then introduce a recursive derivation of the Moore bound, which we later generalize for multicast traffic.

A $(d, K)$ graph consists of $N$ nodes, each with degree $d$, and has a diameter of $K$. Thus, it is a $d$-regular graph with maximum hop distance of $K$. A $(d, K)$ graph with the maximal number of nodes, $N^*$, is a Moore graph. A $(d, 1)$ Moore graph is a clique (complete graph) with $N^* = d + 1$. A $(2, K)$ Moore graph is a cycle of $2K + 1$ nodes. For $d > 2$, each node can be thought of as the root of a tree. From the root, $d$ new nodes can be reached in one hop, $d(d - 1)$ in the second hop, up to $d(d - 1)^{K-1}$ in the $K$th (final) hop. The relationship between $N^*$, $d$, and $K$ is given by
\[ N^* = \begin{cases} 
2K + 1 & \text{if } d = 2 \\
\frac{d(d-1)^{K-2}}{d-2} & \text{if } d > 2 
\end{cases} \quad (5.1) \]

Intuitively, a Moore graph is an optimal connectivity pattern for \( N^* \) nodes of degree \( d \). Unfortunately, Moore graphs do not exist for many combinations of \( d \) and \( K \). In particular, it has been proven that \((d, K)\) Moore graphs do not exist for \( d, K \geq 3 \) [11] and for \( K = 2 \) except when \( d = 3, 7 \) or 57 [13]. An example of a \((3, 2)\) Moore graph (otherwise known as the Petersen graph [2]) is shown in Figure 5.1.

![Figure 5.1 The Petersen graph, a \((3, 2)\) Moore graph.](image)

An important problem, due to its applications in communication networks, is to maximize the number of nodes of a \((d, K)\) graph. Some approaches for constructing \((d, K)\) graphs with large \( N \) are summarized in [32]. The dual of this problem is to, given \( N \) and \( d \), find a topology that minimizes the diameter \( K \).

The lower bound on expected number of hops, \( h \), between any two nodes for any \( N \)-node, \( d \)-regular graph is known as the Moore bound, and is derived (for \( d > 2 \)) as follows. On every hop, all nodes reached are reached for the first time. Thus, on the first hop, \( d \) nodes are reached. While on hop \( k \), \( d(d - 1)^k \) nodes are reached (for \( k < K \)). On the \( K \)th hop, \( N - \frac{d(d-1)^{K-1} - 2}{d-2} \) nodes are reached. The expression for total mean hop distance is

\[
E[h] = \left( \frac{1}{N-1} \right) \left( d \left[ 1 + \sum_{k=2}^{K-1} k(d-1)^{k-1} \right] + K \left[ N - \frac{d(d-1)^{K-1} - 2}{d-2} \right] \right) = \frac{d \left[ (d-1)^{K-1}[(d-2)(K-1)-d+1]+1 \right] + (d-2)[KN(d-2)-2]}{(N-1)(d-2)^2} \quad (5.2)
\]
In many networks, however, the links between nodes are unidirectional (directed), and it is useful to know the Moore bound in these situations. For directed $d$-regular graphs, each node has $d$ incoming links and $d$ outgoing links. The mean hop distance for these graphs (with $d > 2$), presented in [19], is

$$E[h] = \frac{d - d^{K+1} + KN(d - 1)^2 + K(d - 1)}{(N - 1)(d - 1)^2}$$  \hspace{1cm} (5.3)

In this chapter, we consider directed, $d$-regular graphs with a total of $N_K$ nodes. To enable our lower bound to be topology-independent, we consider each graph to contain a subgraph which is a complete, $d$-ary tree rooted at the source. The $K + 1$ levels of this tree are numbered $0, 1, \ldots, K$ starting from the leaf level, and the number of nodes in a subtree rooted at level $k$ is denoted $N_k = \frac{d^{k+1} - 1}{d - 1} = dN_{k-1} + 1$ ($1 \leq k \leq K$). Using this notation, we can derive an elegant, recursive expression that is equivalent to the Moore bound. We are interested in $h_{1,K}$, the expected number of links used to transmit information from the source node to the destination node, in terms of $h_{1,k}$ ($2 \leq k < K$), the number of links required to transmit information from an intermediate node at level $k$ to the destination node. The destination node may either be (1) the node at the $k$th level with probability $\frac{1}{N_k}$, or (2) below the $k$th level with probability $(1 - \frac{1}{N_k})$. For the former, the cost is 0; for the latter, the cost is $(1 + h_{1,k-1})$, since one hop is required to reach one of the nodes in the $(k - 1)$th level and $h_{1,k-1}$ hops are required by the recursive definition, to reach the final destination from the $(k - 1)$th level node. Therefore, the recursive version of the Moore bound can be written as

$$h_{1,k} = \begin{cases} 
1 + h_{1,k-1} & \text{if } k = K \\
(1 - \frac{1}{N_k})(1 + h_{1,k-1}) & \text{if } 1 \leq k < K \\
0 & \text{if } k = 0
\end{cases}$$  \hspace{1cm} (5.4)

for $2 \leq k < K$, with the base case of $h_{1,0} = 0$, and the boundary condition $h_{1,K} = 1 + h_{1,K-1}$. Note that $d$ does not appear in Equation (5.4), yet it is implicit due to the relationship between $N$, $d$, and $K$. This recursive notation will allow us to generalize the Moore bound for multicast traffic.

### 5.3 Generalizing the Moore Bound: Complete Tree Case

In this section, we present two approaches for computing the average number of required links (expected cost or link utilization) for multicasting a message to $m$ destinations that are randomly
chosen among the non-root nodes of a complete d-ary tree of height K. The first approach is recursive.

At every level of the tree we represent the expected cost of transmission to the destinations below that level as the sum of the expected costs of transmission to all subtrees, and we compute those costs recursively. This method is more intuitive and simple to derive. However, in practice, recursion leads to slower implementations. Therefore, we also derive a second approach, which is iterative and exhibits a faster running time. It exploits the symmetries at each level of the tree so that we compute the overall cost of multicasting as a sum of the costs along the different levels of the tree. We will also show that this latter method is more efficient in terms of running time and storage space.

5.3.1 Recursive approach

We examine the case where the tree is d-ary and multicast size 1 \( \leq m < N_K \). We will show that the average cost \( h_{m,K} \) of sending a message to \( m \) destinations from the root of a d-ary complete tree is given by the following recursive relation

\[
h_{m,k} = \begin{cases} 
\frac{d}{(N_K-1)} \sum_{i=1}^{m} (1 + h_{i,k-1}) \binom{N_K-1}{m_i} \binom{N_K - N_K - 1}{m - i} & \text{for } k = K \\
\frac{d}{m} \sum_{i=1}^{m} (1 + h_{i,k-1}) \binom{N_K-1}{i} \binom{N_K - N_i - 1}{m - i} & \text{for } 1 \leq k < K \\
0 & \text{otherwise}
\end{cases}
\]  

(5.5)

Note that our goal is to find the value \( h_{m,K} \).

For the case where 1 \( \leq k < K \), we know that by definition, \( h_{m,k} \) is the expected cost of sending a message to \( m \) destinations where all nodes of the tree are potential destinations, including the root of the subtree. We divide the tree into the \( d \) subtrees each of which is of height \( k - 1 \). Let \( C_s \) be the cost of multicasting to some number of destinations in subtree \( s \). Since these subtrees are disjoint, the total cost \( h_{m,k} \) is the sum of the expected costs of each of the \( d \) subtrees. Since all of the subtrees are identical, they will have the same expected cost. Therefore, by recursion

\[
h_{m,k} = d \cdot E[C_s] = d \sum_{i=1}^{m} E[C_s | i \text{ destinations in } s] \cdot \Pr\{i \text{ destinations in } s\}
\]
\[
\Pr\{i \text{ destinations in } s\} = d \sum_{i=1}^{m} (1 + h_{i,k-1}) \cdot \Pr\{i \text{ destinations in } s\} \quad (5.6)
\]

In order to calculate the probability that \(i\) destinations are in some subtree \(s\), note that the total number of ways we can choose \(m\) destinations in a complete tree of height \(k\) is \(\binom{N_k}{m}\). Out of these there are \(\binom{N_k-1}{i}\) ways to choose \(i\) destinations in subtree \(s\) and \(\binom{N_k-N_{k-1}}{m-i}\) ways to choose \(m-i\) destinations outside of subtree \(s\). Thus

\[
h_{m,k} = d \sum_{i=1}^{m} (1 + h_{i,k-1}) \frac{\binom{N_k-1}{i} \binom{N_k-N_{k-1}}{m-i}}{\binom{N_k}{m}}
\]

\[
= \frac{d}{\binom{N_k}{m}} \sum_{i=1}^{m} (1 + h_{i,k-1}) \binom{N_k-1}{i} \binom{N_k-N_{k-1}}{m-i} \quad (5.7)
\]

The case for \(k = K\) follows directly, with the modification that the possible destinations in the tree are \(N_k - 1\) (the root cannot be a destination). Therefore for \(k = K\),

\[
\Pr\{i \text{ destinations in } s\} = \frac{\binom{N_k-1}{i} \binom{N_k-N_{k-1}-1}{m-i}}{\binom{N_k-1}{m}} \quad (5.8)
\]

Efficient computation of \(h_{m,K}\) can be facilitated by dynamic programming. In the table below, we show how to build the array in memory for all \(h_{i,k}\) where \(i\) is the number of destinations \((0 \leq i \leq m)\) and \(k\) is the level of the subtree that we are considering \((0 \leq k \leq K)\)

<table>
<thead>
<tr>
<th>(h_{i,k})</th>
<th>(k = 0)</th>
<th>(k = 1)</th>
<th>(k = K-1)</th>
<th>(k = K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i = 1)</td>
<td>0</td>
<td>(h_{1,1})</td>
<td>(h_{1,K-1})</td>
<td>(h_{1,K})</td>
</tr>
<tr>
<td>(i = 2)</td>
<td>0</td>
<td>(h_{2,1})</td>
<td>(h_{2,K-1})</td>
<td>(h_{2,K})</td>
</tr>
<tr>
<td>(\vdots)</td>
<td>(\vdots)</td>
<td>(\vdots)</td>
<td>(\vdots)</td>
<td>(\vdots)</td>
</tr>
<tr>
<td>(i = m-1)</td>
<td>0</td>
<td>(h_{m-1,1})</td>
<td>(h_{m-1,K-1})</td>
<td>(h_{m-1,K})</td>
</tr>
<tr>
<td>(i = m)</td>
<td>0</td>
<td>(h_{m,1})</td>
<td>(h_{m,K-1})</td>
<td>(h_{m,K})</td>
</tr>
</tbody>
</table>

The goal is to compute the value in the lowest right corner of the table. To do this we compute the elements of the table column by column starting from the leftmost column. We know that the leftmost column \((k = 0)\) contains only 0, since this is the cost of sending messages in a tree that contains no nodes other than the root one. To compute the next \(K-1\) columns, we use the second line of Equation
(5.5), so we compute $h_{i,k}$ using the values $h_{1,k-1}, h_{2,k-1}, \ldots, h_{i-1,k-1}$, which are in the column to the left and are already computed. Finally, to compute the values in the rightmost column ($k = K$) we use the first line of Equation (5.5). (In fact, we do not need to compute column $K$ completely. We can just compute its lowest element.) At any point in this process, we only need to store one column, so the space required by this approach is $O(m)$. In addition, the computation will run in time $O(Km^3)$. (If the binomial coefficients are precomputed, then the space complexity is $O(Km)$ and the time complexity is simply $O(Km^2)$.)

### 5.3.2 Iterative approach

In this subsection we develop an alternate expression for the expected cost of transmitting a multicast to $1 \leq m < N_K$ destinations in a $d$-ary complete tree. In particular, we show that

$$h_{m,K} = N_K - 1 - \frac{1}{\binom{N_K - 1}{m}} \sum_{i=0}^{K-1} d^{K-i} \binom{N_K - 1 - N_i}{m}$$

Let $C_{m,i}^e$ represent the cost of transmitting on some edge $e$ on level $i$ given that there are $m$ destinations total. By symmetry, the cost associated with transmitting on each edge on a level is the same. Naturally, $C_{m,i}^e = 0$ if edge $e$ is not used in the multicast transmission. and $C_{m,i}^e = 1$ if edge $e$ is used. Note that

$$E[C_{m,i}^e] = \Pr\{e \text{ is used}\} = 1 - \Pr\{e \text{ is not used}\}$$

Since we exclude the root as a possible destination, there are $\binom{N_K - 1}{m}$ ways to choose $m$ destinations anywhere in the tree. Also, there are $\binom{N_K - 1 - N_i}{m} \binom{N_i}{0} = \binom{N_K - 1 - N_i}{m}$ ways to choose the $m$ destinations so that all of them are outside the subtree below edge $e$ at the $i$th level (the number of nodes below the tagged edge is $N_i$, so the number of nodes not under the edge is $N_K - 1 - N_i$, and we are choosing any $m$ of the latter). Therefore

$$E[C_{m,i}^e] = 1 - \frac{\binom{N_K - 1 - N_i}{m}}{\binom{N_K - 1}{m}}$$

(5.9)
At each level \( i \) there are \( d^{K-i} \) edges, each with expected cost \( E[C_{m,i}] \). So our total expected cost is

\[
h_{m,K} = \sum_{i=0}^{K-1} d^{K-i} E[C_{m,i}]
\]
\[
= \sum_{i=0}^{K-1} d^{K-i} \left( 1 - \frac{N_{K-1} - N_i}{N_{m-1}} \right)
\]
\[
= \sum_{i=0}^{K-1} d^{K-i} - \sum_{i=0}^{K-1} d^{K-i} \frac{N_{K-1} - N_i}{N_{m-1}}
\]

Since \( \sum_{i=0}^{K-1} d^{K-i} = \sum_{i=1}^{K} d^i = \sum_{i=0}^{K} d^i - d^0 = N_K - 1 \), we get

\[
h_{m,K} = N_K - 1 - \frac{1}{N_{m-1}} \sum_{i=0}^{K-1} d^{K-i} \left( \frac{N_{K-1} - N_i}{N_{m}} \right)
\]

The running time of this iterative approach is \( O(Km) \) and the storage requirement is \( O(1) \).

### 5.4 Generalizing the Moore Bound: Incomplete Tree Case

In the previous section, we derived a formula for the expected cost, \( h_{m,K} \), of multicasting to \( m \) destinations in a complete \( d \)-ary tree of height \( K \). In this section, we will generalize our approach to allow us to obtain the average cost in trees which are not complete (some leaves are removed from the lowest level of a complete \( d \)-ary tree).

We define a left tree to be a tree where for every pair of siblings, the sibling to the left has at least as many children as the sibling to the right. We also define a compact tree to be a tree which has the minimal height of all trees of the same maximum degree and same number of nodes. Finally, we define a heap as a left compact tree in which all the leaves at the lowest level are “packed to the left.” Figure 5.2 shows two binary trees, the tree in Part (a) is a binary heap, the one in Part (b) is a binary left compact tree.

In this chapter, we are interested in lower bounds on the cost of shortest-path multicasting. In Appendix C we prove that for all trees with the same maximum degree and same number of nodes, the heap exhibits the minimal expected multicasting cost. Thus, we will derive a formula for the expected multicasting cost, \( h_{m,K}(N) \), for a heap, where \( m \) is the number of destinations, \( K \) is the height of the heap, \( N \) is the total number of nodes in the heap and \( d \) is the maximum degree in the heap. This is
5.4.1 Notation

The height, $K$, of a heap with $N$ nodes and degree $d$ is $K = \lfloor \log_d (N(d-1)+1) \rfloor - 1$, and the number of leaves, $X$, at the lowest level ($0 < X \leq d^K$) can be expressed as $X = N - N_{K-1}$. A heap has three kinds of subtrees:

- **Left subtrees:** These are complete subtrees with $N_{K-1}$ nodes. Their root nodes are left children of the root node of the heap. There are $L$ ($0 \leq L \leq d$) left subtrees in a heap, where

  $$L = \lfloor \frac{X}{d^{K-1}} \rfloor$$

- **Right subtrees:** These are complete subtrees with $N_{K-2}$ nodes. Their root nodes are right children of the root node of the heap. There are $R$ ($0 \leq R < d$) right subtrees in a heap, where

  $$R = d - \lfloor \frac{X}{d^{K-1}} \rfloor$$

- **Central subtree:** The central subtree is the only incomplete subtree in a heap. This subtree is between the left complete subtrees and right complete subtrees of the root. There are $C = d - L - R$
central subtrees, which will be either 0 or 1. The central subtree is also a heap and the number of nodes $N^{(c)}$ in it is specified by

$$N^{(c)} = \frac{N}{\text{all nodes}} - \frac{L \cdot N_{K-1}}{\text{nodes in left trees}} - \frac{R \cdot N_{K-2}}{\text{nodes right trees}} - \frac{1}{\text{root}}$$

Note that $N^{(c)} = 0 \Leftrightarrow C = 0$.

Figure 5.3 illustrates the definitions of left, central and right subtrees of the heap. For purposes of illustration, let us consider several examples of heaps of height $K = 2$ and $d = 4$. For the heap with $N = 11$ nodes, the number of leaves at the lowest level is $X = 6$. Also, there is $L = 1$ left subtree of the root with $N_1 = 5$ nodes. There are $R = 2$ right subtrees of the root with $N_0 = 1$ node. There is $C = 1$ central subtree which has $N^{(c)} = 3$ nodes. For the heap with $N = 13$ nodes, $X = 8$, $L = 2$, $R = 2$, $C = 0$, and $N^{(c)} = 0$. For the heap with $N = 17$ nodes, $X = 12$, $L = 3$, $R = 1$, $C = 0$, and $N^{(c)} = 0$. For the heap with $N = 8$ nodes, $X = 3$, $L = 0$, $R = 3$, $C = 1$, and $N^{(c)} = 4$.

### 5.4.2 Recursive approach

As we did in the previous section for complete trees, first we present a recursive formula for the expected cost of multicasting. Following this, we will show an iterative version of the formula. The expected cost $h_{m,K}(N)$ of sending a message to $m$ destinations from the root of a heap of $N$ nodes with maximum degree $d$ and height $K$ can be obtained from the formula
for $1 \leq k \leq K$ and $1 \leq m < N + 1 - \delta(k)$. The base case of the recursion requires that $h_{m,k}(N) = 0$, for all other values of $m$ and $k$. We define $\delta(k)$ to be an indicator function such that

$$
\delta(k) = \begin{cases} 
0 & \text{if } k < K \\
1 & \text{if } k = K 
\end{cases}
$$

The intuition behind this expression is very similar to that of the complete tree (Equation 5.5). However, there is a slight distinction. In the case of the complete tree all the subtrees of the root have exactly $N_{k-1}$ nodes. In an incomplete tree this is not the case. We have a combination of subtrees of $N_{k-1}$ nodes, subtrees of $N_{k-2}$ nodes, and possibly one incomplete subtree. We represent $h_{m,k}(N)$ as the sum of the expected contributions to the overall cost by each of the three separate types of subtrees. By symmetry, each of the subtrees of the same type (e.g. the subtrees with $N_{k-1}$ nodes) will have the same expected cost. Therefore, the contribution to the overall cost by each type of subtree is equal to the number of subtrees of that type times the expected cost of one subtree of that type.

Consider the case in which $k < K$ and $1 \leq m \leq N_k$. The expected cost of a subtree of $N_{k-1}$ nodes is given by the general form

$$
\sum_{i=1}^{m} (1 + h_{i,k-1}) \frac{\binom{N_{k-1}}{i} \binom{N_k - N_{k-1}}{m-i}}{\binom{N_k}{m}}
$$

which has been used in various ways throughout this chapter. So the expected cost of the left subtrees with $N_{k-1}$ nodes is

$$
L \sum_{i=1}^{m} (1 + h_{i,k-1}) \frac{\binom{N_{k-1}}{i} \binom{N_k - N_{k-1}}{m-i}}{\binom{N_k}{m}}
$$
Analogously, we find the expected cost of the right subtrees with \( N_{k-2} \) nodes to be

\[
R \sum_{i=1}^{m} (1 + h_{i,k-2}) \frac{\binom{N_{k-2}}{i} \binom{N_{k}-N_{k-2}}{m-i}}{\binom{N_k}{m}}
\]  

(5.14)

Finally, the expected cost of the incomplete subtree is

\[
C \sum_{i=1}^{m} (1 + h_{i,k-1}(N^{(e)})) \frac{\binom{N^{(e)}}{i} \binom{N_{k}-N^{(e)}}{m-i}}{\binom{N_k}{m}}
\]  

(5.15)

The sum of the expressions in Equations 5.13, 5.14 and 5.15 represents the total cost \( h_{m,k}(N) \), which proves 5.10 for \( k < K \). The case of \( k = K \) follows closely, except for a different probability of having \( i \) destinations in each of the subtrees. Just as in Equation 5.5, we only have to substitute \( N_k - 1 \) for \( N_k \) in the formula for the probability since when \( k = K \), there are only \( N_k - 1 \) possible destinations. This is accomplished with the indicator function, \( \delta(k) \).

The space and time complexities are the same as in the complete tree case: \( O(m) \) and \( O(km^3) \), respectively. This is because the computation for each type of subtree is similar to the computation performed on the complete subtree.

5.4.3 Iterative approach

In this subsection we will develop an iterative approach to show that the average cost, \( h_{m,K}(N) \), of sending a message to \( m \) destinations from the root of a \( d \)-ary heap with \( N \) nodes is

\[
h_{m,K}(N) = N - 1 - \sum_{i=0}^{K-1} L_i \left( \frac{N-1-N_i}{m} \right) - \sum_{i=1}^{K-1} C_i \left( \frac{N_{k-1} + N_i + N_{k-1} R_i - 1}{m} \right) \frac{m}{\binom{N}{m} - \frac{m}{\binom{N-1}{m}}}
\]

(5.16)

where for all levels \( i, 1 \leq i \leq K - 1 \)
\( L_i = \left\lfloor \frac{N - N_{K-1}}{d^i} \right\rfloor \) is the number of complete subtrees of height \( i \) (left subtrees) and each left subtree has \( N_i \) nodes

\( R_i = \left\lfloor \frac{N_{K-1}}{d^i} \right\rfloor \) is the number of complete subtrees of height \( i - 1 \) (right subtrees) and each right subtree has \( N_{i-1} \) nodes

\( C_i = d^{K-i} - L_i - R_i \) is the number of incomplete heaps (either 0 or 1 central subtrees) at level \( i \) and there are \( N - L_i N_i - R_i N_{i-1} - N_{K-i-1} \) nodes in the incomplete heap below level \( i \)

For convenience, we assume that \( L_0 = N - N_{K-1} \). An example of the definitions of \( L_i, C_i \) and \( R_i \) is provided in Figure 5.3. For this heap, \( N_0 = 1, N_1 = 4, N_2 = 13, N_3 = 40 \). Also, \( L_0 = 12, L_1 = 4, C_1 = 0, R_1 = 5, \) and \( L_2 = 1, C_2 = 1, R_2 = 1 \).

The approach is a generalization of the one used in the case of a complete \( d \)-ary tree. Again, we exploit the fact that if at every level the subtrees below two edges are of the same size and shape, then the expected cost associated with transmission over each of the two edges will be the same. In the case of the complete tree, at the \( i \)-th level all edges have the same expected cost of transmission since all of the edges are above complete subtrees with \( N_i \) nodes. In the case of the incomplete tree, at the \( i \)-th level (we assume \( i > 0 \)) there are three types of edges (left, central and right) according to the types of subtrees under them. We know that all the left edges will have the same expected cost of transmission; similarly, all right edges will have the same expected cost. So, for example, we can compute the expected cost for one, tagged, right edge at level \( i \). By multiplying this by the number of right edges at level \( i \), we can compute the total cost, associated with all the right edges at that level. Similarly, we can compute the total cost for all left edges. Now when we add the cost for the central edge to the total cost of the right and left edges, we get the total cost of all the edges at this level. So the expected cost at \( i \)-th level is

\[
E[\text{cost at } i\text{-th level}] = L_i \cdot E[\text{cost of tagged left edge at level } i] + C_i \cdot E[\text{cost of central edge at level } i] + R_i \cdot E[\text{cost of tagged right edge at level } i]
\]
Generalizing Equation (5.9) we get

\begin{align*}
E[\text{cost of tagged left edge at level } i] &= 1 - \frac{(N_{i-1} - N_i)}{(N_{m-1})} \\
E[\text{cost of central edge at level } i] &= 1 - \frac{(N_{K-1} + N_i, L_i + N_{i-1} R_i)}{(N_{m-1})} \\
E[\text{cost of tagged right edge at level } i] &= 1 - \frac{(N_{i-1} - N_{i-1})}{(N_{m-1})}
\end{align*}

Substituting for the three costs from Equation (5.19) into Equation (5.18) yields

\begin{align*}
E[\text{cost at } i\text{-th level}] &= L_i (1 - \frac{(N_{i-1} - N_i)}{(N_{m-1})}) + C_i (1 - \frac{(N_{K-1} + N_i, L_i + N_{i-1} R_i)}{(N_{m-1})}) + R_i (1 - \frac{(N_{i-1} - N_{i-1})}{(N_{m-1})}) \\
&= (L_i + C_i + R_i) - L_i \left(\frac{(N_{i-1} - N_i)}{(N_{m-1})}\right) - C_i \left(\frac{(N_{K-1} + N_i, L_i + N_{i-1} R_i)}{(N_{m-1})}\right) \\
&= d^{K-i} - L_i \left(\frac{(N_{i-1} - N_i)}{(N_{m-1})}\right) - C_i \left(\frac{(N_{K-1} + N_i, L_i + N_{i-1} R_i)}{(N_{m-1})}\right) - R_i \left(\frac{(N_{i-1} - N_{i-1})}{(N_{m-1})}\right)
\end{align*}

The expected cost at the 0-th level is

\begin{align*}
E[\text{cost at 0-th level}] = L_0 - L_0 \left(\frac{(N_{i-1} - N_i)}{(N_{m-1})}\right)
\end{align*}

The expected cost of transmission for the tree is the sum of the costs for all levels so we add Equation (5.21) to the sum of Equation (5.20) for all levels \( i \), where \( 1 \leq i \leq K - 1 \):

\begin{align*}
h_m(K)(N) &= L_0 + \sum_{i=0}^{K-1} d^{K-i} - \sum_{i=0}^{K-1} L_i \left(\frac{(N_{i-1} - N_i)}{(N_{m-1})}\right) - \sum_{i=1}^{K-1} C_i \left(\frac{(N_{K-1} + N_i, L_i + N_{i-1} R_i)}{(N_{m-1})}\right) - \sum_{i=1}^{K-1} R_i \left(\frac{(N_{i-1} - N_{i-1})}{(N_{m-1})}\right) \\
&= L_0 + N_{K-1} - 1 - \sum_{i=0}^{K-1} L_i \left(\frac{(N_{i-1} - N_i)}{(N_{m-1})}\right) - \sum_{i=1}^{K-1} C_i \left(\frac{(N_{K-1} + N_i, L_i + N_{i-1} R_i)}{(N_{m-1})}\right) - \sum_{i=1}^{K-1} R_i \left(\frac{(N_{i-1} - N_{i-1})}{(N_{m-1})}\right)
\end{align*}

This formula is much easier to compute than the recursive one since the required space is \( O(1) \) and the time is \( O(Km) \). Note that these complexities are identical to the space and time complexities of the iterative algorithm for the complete tree case, because the same algorithm is being applied to the three types of subtrees in the incomplete tree case.
5.5 Numerical Examples

In this section, we present numerical results based on the analytical expressions derived in the previous sections. These results will be presented in two parts. First, the accuracy of the analytical expressions for our lower bound on multicasting will be evaluated by comparing numerical results obtained from them with simulation results. Following this, the analytical expressions will be used to provide general recommendations on the effectiveness of multicast routing versus multiple point-to-point and broadcast routing.

5.5.1 Effectiveness of the analytical expressions

The Internet MBONE is largely tree-like although it does have a number of redundant paths between pairs of nodes for purposes of fault tolerance. A practical application for our expressions is to determine the lower bound on the expected link utilization of multicasting in MBONE-like networks. If a very small number of redundant paths (cycles) are added to a tree network, the expected cost of multicasting in that network is guaranteed not to increase, and will probably slightly decrease due to increased connectivity. Therefore, we expect our lower bound to hold only for sparsely connected, but not richly connected graphs. Furthermore, our approach also assumes $d$-regular connectivity, which is certainly not the case for the MBONE or most real networks. However, given a real network with nodal degree distributed between 1 and $d$, if we compute our expressions using $d = d$, we will produce a lower bound (possibly a loose one) for multicasting in the network. Thus, since the MBONE is a tree-based, sparsely-connected graph, we expect that our bound will hold for the MBONE and similar virtual multicast topologies if we compute our expression using $d$ to be the greatest empirical nodal degree in the network.

In order to test the usefulness of our bound, we created one hundred 64-node random tree networks. The outdegree of each node was uniformly distributed between 1 and $d$. For example, we start with a root, which is assumed to be the source of the multicast, and assign it $\gamma$ (where $1 \leq \gamma \leq d$) children. We then assign children to the newly created nodes, and so on until 64 nodes are assigned. Then, the $m$ destinations are randomly chosen and the cost of multicasting is computed. This process is repeated for 100 destination sets in each random tree.

In Figures 5.4 and 5.5 we compare the expected multicasting cost as a function of the number of destinations in a random tree versus our lower bound expressions. For example, in Figure 5.4, the upper curve corresponds to the cost of multicasting in a random tree with $N = 64$ nodes, and $\gamma$ distributed uniformly between 1 and 2. The lower curve is obtained from our analytical expressions applied to
a heap with 64 nodes and degree $d = 2$. Figure 5.5 is a comparison between the multicast cost in a random tree with 64 nodes and $\gamma$ distributed uniformly between 1 and 4. The results show that the expressions we have derived are indeed lower bounds for multicasting in tree networks. Note that tree networks perform “single-path” routing; that is, there is a single path between any two pairs of nodes in the network.

However, the tree approach may not yield the lowest overall multicasting cost in networks more richly connected than tree networks. We would now like to compare our analytical expressions with simulations of multicasting in networks such as the shufflenet and the hypercube, each of which have multiple paths between any two nodes. In Figure 5.6, we compare the analytical multicasting cost for various $m$ in a 64-node, $d = 6$ heap (upper curve) with the simulated multicasting cost in a binary hypercube with $d = 6$ and $N = 64$ nodes (lower curve). A variation of the multicasting algorithm presented in [24] was used in the simulations. The figure shows that the analytical expression is no longer a lower bound. This is because the hypercube has multiple paths between the source and each member in the destination set. The multiple paths, as discussed above, lead to lower multicasting costs. However, the two curves follow the same trends.

In Figure 5.7 we compare the analytical expression with the average multicast cost in a 64-node
Figure 5.5  Expected multicast link utilization for various $m$ on 64-node random trees with $d \leq 4$ versus $h_{4,m}$.

$(2,4)$-Shufflenet. For the shufflenet, the recently developed multicasting algorithm presented in [5] was simulated. Again, we find that the analytical results follow the same trends as the simulation results, although the difference is a bit larger than was observed in the hypercube case. From these results, it appears that the analytical cost of multicasting in a heap is a reasonable approximation of the cost of multicasting in corresponding regular networks.

5.5.2 Recommendations on multicasting

We will now use the analytical results developed in the previous sections to study the lower bounds of broadcasting, multiple unicasts and multicast communication. Comparing the performance of these three techniques allows the system designer to decide, based on the multicast set size $(m)$, which technique is the most cost-effective. While broadcasting and multiple-unicast are usually simpler to implement algorithmically, multicasting will always perform as well or better than these rudimentary techniques.

Note that the cost of minimum-delay multiple-unicast is obtained by simply multiplying the destinations set size, $m$, with the cost of unicasting a message in the network, i.e., $h_{1,K}(N)$. The minimum cost of broadcasting a message is given by $(N - 1)$. 
Figure 5.6 Expected multicast link utilization for various \( m \) in the 64-node hypercube versus \( h_{6,m} \).

Figure 5.8 shows expected cost versus destination set size for multicast, multiple-unicast, and broadcast traffic, for a network with \( K = 5 \) and \( d = 2 \) (\( N_K = 63 \))\(^2\). Broadcasting is a trivial solution; the cost does not change with \( m \). Therefore, it is not a good technique for transmitting small multicasts. The cost of multiple unicast increases linearly with \( m \) and is therefore not a good technique when \( m \) approaches \( N_K \). As expected, multicast communication provides the low-cost solution for all values of \( m \).

We define the intersection of the broadcast and multiple-unicast curves to be the Maximum Benefit Point (MBP) of multicasting.

\[
MBP = \frac{N - 1}{h_{1,K}(N)}
\]

At the value of \( m \) closest to the MBP (which we refer to as \( m' \)), multicasting demonstrates the greatest increase in performance over that of both broadcasting and multiple-unicast. In Figure 5.9 we plot \( m' \) for various \( N \) and \( d \). We note that the increase in the MBP is more highly correlated with the degree than with the size of the network. This is more apparent in the next figure, Figure 5.10, where

\(^2\)Note that the curves shown in this figure are very similar to the figures developed in [24] where a heuristic multicasting algorithm developed for hypercube networks was examined. The differences are the following: in [24], the curve for multicast communication plotted was obtained from simulating the heuristic, and the multiple unicast curve was obtained from an approximate analysis. In contrast, the curves which we plot in this chapter are exact.
we plot \( m' \), normalized to maximum possible destination set-size, for various values of \( N \) and \( d \). The figure shows that for a fixed degree, increasing the size of the network actually results in a smaller normalized \( m' \). These results indicate that good multicasting algorithms are to be recommended for small destination set sizes for networks with small degrees.

A measure of the actual amount of bandwidth saved due to multicasting can be given by subtracting the cost of multicasting at \( m' \) from the cost of multiple-unicast or broadcast at \( m' \). We define this difference to be the Maximum Benefit (\( c \)) of multicasting. Figure 5.11 plots \( c \) versus \( d \) for varied \( N \). Note that, as the degree and the size of the network grows, the benefit of multicasting increases rapidly. In Figure 5.12 we plot the \( c \) normalized to the cost of broadcasting. The latter shows that for networks with large degrees multicasting provides relatively insignificant benefits.

5.6 Future Work and Concluding Remarks

Multicasting is an important area of research in telecommunications. In this chapter, we have provided a new and fundamental lower bound to the cost of building minimum-delay trees for multicast communication in tree-like networks. Furthermore, the expressions we have derived provide an approximation to the cost of multicasting in regular graphs. The bound is similar to the Moore bound, which
Figure 5.8  Expected cost versus destination set size, for multicast, multiple-unicast, and broadcast traffic, for a network with $K = 5$ and $d = 2$ ($N_K = 63$).

provides a lower bound on the expected cost of point-to-point communication in networks.

The bound presented in this chapter will be useful in (1) comparing various network topologies designed to support multicasting services, (2) providing exact lower bounds on the performance of networks whose structures make them difficult to analyze, and (3) providing mathematically intuitive insights regarding the effectiveness of multicasting over that of multiple unicasting and broadcasting.
Figure 5.9 Maximum benefit point

Figure 5.10 Normalized maximum benefit point
Figure 5.11  Maximum benefit

Figure 5.12  Normalized maximum benefit
6 CONCLUSION

In this dissertation we gave a general overview of visibility-based pursuit-evasion. We presented polynomial-time algorithms for clearing a polygonal region by a single 1-searcher and by a pair of 1-searchers (each of them equipped with a single flashlight).

While recently there have been a number of important advances in the pursuit-evasion field, there are still a number of interesting open problems. A natural extension of our results will be an algorithm for clearing a polygon with \( k \) collaborating 1-searchers, or \( \infty \)-searchers, who are not required to maintain mutual visibility at all time. Note that LaValle et al [26] showed that the problem of finding the minimal number of pursuers required to clear a given polygon is \( \text{NP-hard} \). However, in their proof they used a polygon with holes. It is not known whether a similar result applies in the case of polynomials without holes.

Other interesting directions for future research is combining our results is with the minimal sensing approach of Sachs et al [37], i.e., whether the two pursuers can find a winning strategy without prior knowledge of the shape of the polygon.

Finally, extensions of the current results to three-dimensional environments will create greater incentive for implementation of the algorithms into real-world devices.
APPENDIX A PROOF OF LEMMA 3.4

Assume that \( X_0, Y_0, X_1, Y_1 \) and \( k \) are as defined in Lemma 3.4. Also, for convenience, assume that we order the points of \( \partial P \) starting from \( x_0 \), the left boundary of the interval \( X_0 \). We breakdown the proof according to the different values of \( k \):

- \([k = 0]\). We prove the following statement.

**Assume that** \( k = 0 \). **Condition 3.1 is satisfied if and only if** \( y_0 < x_1^* \).

**Proof.**

\((\Leftrightarrow)\) Suppose that \( y_0 \geq x_1^* \). It follows that for every \((x_i, y_i) \in B(X_i, Y_i), i = 0, 1:\)

\[
y_0 \geq y_0 \geq x_1^* \geq x_1,
\]

so \( y_0 \geq x_1 \) and Condition 3.1 cannot be satisfied.

\((\Rightarrow)\) Suppose that \( y_0 < x_1^* \). Let \((x_0^*, y_0)\) and \((x_1^*, y_1^*)\) be two extreme points of \( C(X_0, Y_0) \) and \( C(X_1, Y_1) \), correspondingly. Since,

\[
x_0^* < y_0 < x_1^* < y_1^*,
\]

the tuple \( I_v = (x_0^*, y_0, x_1^*, y_1^*, 0) \) satisfies Condition 3.1. \( \square \)

Both the check \( y_0 < x_1^* \) and the construction of \( I_v \) can be done in \( O(1) \) time, regardless of the whether the boxes are independent or not.

- \([k = 1]\).

- \([X_0 \cap X_1 \neq \emptyset \) and \( Y_0 \cap Y_1 \neq \emptyset]\). We prove the following statement.

**Assume that** \( k = 1 \), \( X_0 \cap X_1 \neq \emptyset \) and \( Y_0 \cap Y_1 \neq \emptyset \). **Condition 3.1 is always satisfied.**

**Proof.** Let \( X = X_0 \cap X_1 \neq \emptyset \) and \( Y = Y_0 \cap Y_1 \neq \emptyset \). From the lemmas in Section 3.2 of [29] it follows that \( B(X, Y) \) is a VHC box. Consider an interior point \((x_0^*, y_0^*) \in C(X, Y) \subseteq \)
$C(X_0, Y_0)$. Suppose we move in $X$ from point $(x_0^*, y_0^*)$ down and to the right to point $(x_1^*, y_1^*)$. Assume that the move is sufficiently short, so that $(x_1^*, y_1^*) \in C(X, Y) \subseteq C(X_1, Y_1)$. From the direction of the move it follows that $x_0^* < x_1^* < y_0^* < y_1^*$, so Condition 3.1 is always satisfied by tuple $I_0 = (x_0^*, y_0^*, x_1^*, y_1^*, 1)$. □

Determining whether Condition 3.1 can be satisfied, or equivalently, whether sets $X$ and $Y$ are empty can be done in $O(1)$ time. In order to construct $I_0$, we compute the parameters of $B(X, Y)$ by computing a finite number of visibility polygons in $O(n)$ time. Given the parameters of $B(X, Y)$ we can construct $I_0$ in $O(1)$ time.

- $[X_0 \cap (X_1 \cup Y_1) = \emptyset, Y_0 \cap X_1 \neq \emptyset$ and $Y_0 \cap Y_1 \neq \emptyset]$. We prove the following statement.

Assume that $k = 1$, $X_0 \cap (X_1 \cup Y_1) = \emptyset$, $Y_0 \cap X_1 \neq \emptyset$ and $Y_0 \cap Y_1 \neq \emptyset$. Condition 3.1 is always satisfied.

Proof. Pick $y_1^* \in Y_0 \cap Y_1$ and construct a point $(x_1^*, y_1^*) \in C(X_1, Y_1)$. Similarly, pick $y_0^*$, where $y_0^* \in (\max(x_1^*, y_0), y_1^*)$. Construct $(x_0^*, y_0^*) \in C(X_0, Y_0)$. Since

$$x_0^* \leq x_1^* \leq x_1^* < y_0^* < y_1^*$$

the tuple $I_0 = (x_0^*, y_0^*, x_1^*, y_1^*, 1)$ satisfies Condition 3.1. □

In order to construct $(x_0^*, y_0^*)$ and $(x_1^*, y_1^*)$ we need to construct two visibility polygons. So $I_0$ can be constructed in time $O(n)$,

- $[X_0 \cap (X_1 \cup Y_1) = \emptyset$ and $Y_0 \cap Y_1 = \emptyset]$. We prove the following statement.

Assume that $k = 1$, $X_0 \cap (X_1 \cup Y_1) = \emptyset$ and $Y_0 \cap Y_1 = \emptyset$. Condition 3.1 is satisfied if and only if $x_1 \leq y_0$.

Proof.  

$(\Leftarrow)$ Suppose that $x_1 \geq y_0$. It follows that for every $(x_i, y_i) \in B(X_i, Y_i)$, $i = 0, 1$:

$$x_1 \geq x_i \geq y_0 \geq y_0$$

So $x_1 \geq y_0$ and Condition 3.1 cannot be satisfied.
Suppose that $x_1 < y_0$. Let $(x_0^*, y_0)$ and $(x_1^*, y_1^*)$ be two extreme points of $C(X_0, Y_0)$ and $C(X_1, Y_1)$, correspondingly. Since,

$$x_0^* \leq x_0 < x_1 < y_0 < y_1^*$$

the tuple $I_v = (x_0^*, y_0, x_1^*, y_1^*, 1)$ satisfies Condition 3.1.

Both the check $x_1 < y_0$ and the construction of $I_v$ can be done in $O(1)$ time, regardless of the whether the boxes are independent or not.

- $[k = 2]$

We consider two cases, depending on the mutual position of the boxes.

- $[X_0 \cap X_1 \neq \emptyset \text{ and } Y_0 \cap Y_1 \neq \emptyset]$. We prove the following statement.

Assume that $k = 2$, $X_0 \cap X_1 \neq \emptyset$ and $Y_0 \cap Y_1 \neq \emptyset$. Condition 3.1 is always satisfied.

Proof. Let $X = X_0 \cap X_1 \neq \emptyset$ and $Y = Y_0 \cap Y_1 \neq \emptyset$. From the lemmas in Section 3.2 of [29] it follows that $B(X, Y)$ is a VHC box. Consider an interior point $(x_0^*, y_0^*) \in C(X, Y) \subseteq C(X_0, Y_0)$. Suppose we move in $X$ from point $(x_0^*, y_0^*)$ down and to the left to point $(x_1^*, y_1^*)$. Assume that the move is sufficiently short, so that $(x_1^*, y_1^*) \in C(X, Y) \subseteq C(X_1, Y_1)$. From the direction of the move it follows that $x_0^* < x_1^* < y_1^* < y_0^*$, so Condition 3.1 is always satisfied by tuple $I_v = (x_0^*, y_0^*, x_1^*, y_1^*, 2)$.

Determining whether Condition 3.1 can be satisfied, or equivalently, whether sets $X$ and $Y$ are empty can be done in $O(1)$ time. In order to construct $I_v$, we compute the parameters of $B(X, Y)$ by computing a finite number of visibility polygons in $O(n)$ time. Given the parameters of $B(X, Y)$ we can construct $I_v$ in $O(1)$ time.

- $[X_0 \cap X_1 = \emptyset]$. We prove the following statement.

Assume that $k = 2$ and $X_0 \cap X_1 = \emptyset$. Condition 3.1 is satisfied if and only if $y_1 < y_0$.

Proof.

$(\Leftarrow)$ Suppose that $y_1 \geq y_0$. It follows that for every $(x_i, y_i) \in B(X_i, Y_i)$, $i = 0, 1$:

$$y_1 \geq y_i \geq y_0 \geq y_0$$
So \( y_1 \geq y_0 \) and Condition 3.1 cannot be satisfied.

(\( \Rightarrow \)) Suppose that \( y_1 < \overline{y}_0 \). Let \((x_0^*, \overline{y}_0)\) and \((x_1^*, y_1)\) be two extreme points of \( C(X_0, Y_0) \) and \( C(X_1, Y_1) \), correspondingly. Since,

\[
x_0^* \leq x_0 \leq x_1 \leq x_1^* \leq y_1 < \overline{y}_0
\]

the tuple \( I_v = (x_0^*, \overline{y}_0, x_1^*, y_1, 1) \) satisfies Condition 3.1. \( \square \)

Both the check \( y_1 < \overline{y}_0 \) and the construction of \( I_v \) can be done in \( O(1) \) time, regardless of the whether the boxes are independent or not.
APPENDIX B  LOWER BOUND ON THE NUMBER OF ADDITIONAL CHANNELS IN A CHALLENGE-RESPONSE PROTOCOL

In this section we will prove that no challenge-response protocol which is secure against collaborating eavesdroppers, uses fewer than \( 2\sqrt{N} \) additional channels, the number used in Solution 2.

Suppose that we have a challenge-response protocol \( P \) for \( N \) users, \( u_1, u_2, \ldots, u_N \). \( P \) uses \( a \) challenge channels, \( C_1, C_2, \ldots C_a \) and \( b \) response channels, \( R_1, R_2, \ldots, R_b \). \( P \) also has the property that no two users share the same pair of challenge and response channels, that is, no subset of users can successfully collaborate.

For the protocol \( P \) we define an arbitrary bipartite graph \( G \) which represents the challenge and the response channels for each user. Note that in a graph we normally associate users with vertices and channels with edges. However, in our model we represent every user as an edge, while the challenge channels and the response channels constitute the two parts of the vertex set. We define the challenge-response graph \( G \) as follows:

- **vertex set**, \( V(G) = \{C_1, C_2, \ldots, C_a\} \cup \{R_1, R_2, \ldots, R_b\} \)

  That is, there is a vertex for every challenge or response channel.

- **edge set**, \( E(G) = \{(C_i, R_j) \mid \exists k, t, t' : \text{challenge}(k, t) = C_i(.) \land \text{response}(k, t') = R_j(.)\} \)

  There is an edge between a challenge channel \( C_i \) and a response channel \( R_j \) if and only if there is a user \( u_k \) who is challenged on channel \( C_i \), that is, \( f(k) = i \) and responds on channel \( R_j \), that is, \( g(k) = j \). The graph is bipartite since no edge connects two vertices from \( \{C_1, C_2, \ldots, C_a\} \) or two vertices from \( \{R_1, R_2, \ldots, R_b\} \)

An immediate observation for the challenge-response graph \( G \) is that \( ab \geq N \). There are at most \( ab \) edges between a vertex from \( \{C_1, \ldots, C_a\} \) and a vertex from \( \{R_1, \ldots, R_b\} \). Therefore, if there are more than \( ab \) users, there is a pair of users \( u_x \) and \( u_y \) both of whom correspond to the same edge \( (C_i, R_j) \). Then \( u_x \) and \( u_y \) share the same challenge channel \( C_i \) and response channel \( R_j \) and they can collaborate.
From $ab \geq N$, using the fact that $a$, $b$ and $N$ are natural numbers, we derive

$$b \geq \left\lfloor \frac{N}{a} \right\rfloor$$

Our goal is to find a lower bound on $a + b$:

$$a + b \geq a + \left\lfloor \frac{N}{a} \right\rfloor \geq a + \frac{N}{a}$$

$$= \left( \sqrt{a} - \frac{\sqrt{N}}{\sqrt{a}} \right)^2 + 2\sqrt{N} \geq 2\sqrt{N}$$

with equalities for $\sqrt{a} = \frac{\sqrt{N}}{\sqrt{a}}$ or equivalently, for $a = b = \sqrt{N}$.

We have shown that Solution 2 is optimal in the sense that it uses a total of $2M = 2\sqrt{N}$ challenge and response channels and every other challenge-response protocol which disallows collaboration needs at least as many channels as Solution 2.
APPENDIX C OPTIMALITY OF A HEAP

C.1 Notation

Suppose that we are given a $d$-ary tree $T$ which has $K$ levels. For each level $i$, $0 \leq i \leq K - 1$, we have $d^{K-i}$ edges immediately above the nodes of that level. For $1 \leq j \leq d^{K-i}$, let $E_{i,j}^T$ denote the $j$th edge on the $i$th level, counting edges from left to right (see Figure C.1). Also, let $U_{i,j}^T$ denote the number of leaves which are at the lowest (0th and possibly 1st) level and under edge $E_{i,j}^T$.

![Diagram of a heap]

Figure C.1 Example of a heap with $N = 13$ and $d = 2$

For purposes of illustration, consider the compact tree $T$ from Figure C.1. The the values of $U_{i,j}^T$ for the tree $T$ are enumerated in the table below:

<table>
<thead>
<tr>
<th>$U_{i,j}^T$</th>
<th>$j = 1$</th>
<th>$j = 2$</th>
<th>$j = 3$</th>
<th>$j = 4$</th>
<th>$j = 5$</th>
<th>$j = 6$</th>
<th>$j = 7$</th>
<th>$j = 8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i = 0$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$i = 1$</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$i = 2$</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let us fix a compact tree $T$ of height $K$. We may think of the expected cost $C^T$ of sending a message to $m$ nodes of $T$ as the sum for all levels ($0 \leq i \leq K - 1$) of the expected costs, $C^T_i$, of each level.

$$C^T = \sum_{i=0}^{K-1} C^T_i$$  \hspace{1cm} (C.1)

For every level $i$ ($0 \leq i \leq K - 1$), and for every edge $E_{i,j}$ ($1 \leq j \leq d^{K-i}$) from that level, let $c_{i,j}$ be the expected contribution of that edge to the total cost of level $i$. Then

$$C^T_i = \sum_{j=1}^{d^{K-i}} C_{i,j}$$  \hspace{1cm} (C.2)

We will now prove that of all compact trees of a fixed degree, the heap has lowest expected cost for sending a multicast to $m$ destinations. It suffices to prove that a heap has a lower average cost compared to any other left compact tree of the same degree, due to the symmetry between left and right compact trees. Let $H$ be a heap and $A$ be an arbitrary left compact tree of the same degree. We will show that

$$C^H \leq C^A$$  \hspace{1cm} (C.3)

To prove Equation C.3, we will show that

$$C^H_i \leq C^A_i$$  \hspace{1cm} (C.4)

for $0 \leq i \leq K - 1$.

### C.2 A Heap Has Lower Cost at Any Complete Level than Any Other Compact Tree

Proving Equation C.4 can be accomplished by showing that for every level $i$ ($0 \leq i \leq K - 1$)

$$\sum_{j=1}^{d^{K-i}} C^A_{i,j} \geq \sum_{j=1}^{d^{K-i}} C^H_{i,j}$$  \hspace{1cm} (C.5)

or, equivalently

$$\sum_{j=1}^{d^{K-i}} C^A_{i,j} - \sum_{j=1}^{d^{K-i}} C^H_{i,j} \geq 0$$  \hspace{1cm} (C.6)
At an arbitrary level $i$

\[
\sum_{j=1}^{d_{K-i}} C_{i,j}^A - \sum_{j=1}^{d_{K-i}} C_{i,j}^H = \sum_{j=1}^{d_{K-i}} (1 - \Pr\{\text{no dests use } E_{i,j}^A\}) - \sum_{j=1}^{d_{K-i}} (1 - \Pr\{\text{no dests use } E_{i,j}^H\})
\]

\[
= \sum_{j=1}^{d_{K-i}} \Pr\{\text{no dests use } E_{i,j}^H\} - \sum_{j=1}^{d_{K-i}} \Pr\{\text{no dests use } E_{i,j}^A\} \tag{C.7}
\]

The probability that no destinations use edge $E_{i,j}^T$ is the probability that there are no destinations below edge $E_{i,j}^T$. The number of possible destinations below edge $E_{i,j}^T$ is equal to the sum of the number of nodes in a complete tree with $i - 1$ levels, $N_{i-1}$, and the number of nodes below edge $E_{i,j}^T$ in the 0th level, $U_{i,j}^T$. Noting that $N_i = 0$ for $i < 0$, we get

\[
\sum_{j=1}^{d_{K-i}} C_{i,j}^A - \sum_{j=1}^{d_{K-i}} C_{i,j}^H = \sum_{j=1}^{d_{K-i}} \left( \frac{N_{i-1} - N_{i-1} - U_{i,j}^H}{m} \right) - \sum_{j=1}^{d_{K-i}} \left( \frac{N_{i-1} - N_{i-1} - U_{i,j}^A}{m} \right) \tag{C.8}
\]

\[
= \frac{1}{(N-1)^m} \sum_{j=1}^{d_{K-i}} \left[ \left( \frac{N_{i-1} - U_{i,j}^H}{m} \right) - \left( \frac{N_{i-1} - U_{i,j}^A}{m} \right) \right] \tag{C.9}
\]

where $X_i = N - 1 - N_i$. So, to prove Equation C.5 it suffices to show that

\[
\sum_{j=1}^{d_{K-i}} \left[ \left( \frac{N_{i-1} - U_{i,j}^H}{m} \right) - \left( \frac{N_{i-1} - U_{i,j}^A}{m} \right) \right] \geq 0 \tag{C.10}
\]

Consider the 0th level of both the heap and the compact tree. Recall that the leaves of the heap in this level are "packed to the left." Therefore, if we compare the heap and compact tree, we will find that for any level $i$, $U_{i,j}^H \geq U_{i,j}^A$ for values of $j$ from 1 to some $J$, while $U_{i,j}^H \leq U_{i,j}^A$ for values of $j$ from $J + 1$ to $d_{K-i}$. In general, $J$ may not be unique, but for our purposes below, any valid $J$ may be used. In order to prove that the summation in Equation C.10 is non-negative, we will partition it into two summations. Based on $J$.

\[
\sum_{j=1}^{d_{K-i}} \left[ \left( \frac{N_{i-1} - U_{i,j}^H}{m} \right) - \left( \frac{N_{i-1} - U_{i,j}^A}{m} \right) \right] = J \sum_{j=1}^{J} \left[ \left( \frac{N_{i-1} - U_{i,j}^H}{m} \right) - \left( \frac{N_{i-1} - U_{i,j}^A}{m} \right) \right] \tag{C.11}
\]
Next we will prove two lemmas. Lemma C.1 will give us a lower bound on the left sum of Equation (C.11) and Lemma C.2 will give us a lower bound on the right sum in Equation (C.11). Later we use the two lemmas for a lower bound on the entire sum.

Lemma C.1.

$$\sum_{j=1}^{J} \left[ \left( \frac{X_i - U_{i,j}^H}{m} \right) - \left( \frac{X_i - U_{i,j}^A}{m} \right) \right] \geq - \left( \frac{X_i - U_{i,j}^A}{m-1} \right) \sum_{j=1}^{J} (U_{i,j}^H - U_{i,j}^A)$$  \hspace{1cm} (C.13)

Proof: For all \( j \) (\( 1 \leq j \leq J \)) we have

$$\left( \frac{X_i - U_{i,j}^H}{m} \right) - \left( \frac{X_i - U_{i,j}^A}{m} \right) = - \left( \frac{X_i - U_{i,j}^A}{m} \right) \sum_{l=U_{i,j}^A+1}^{U_{i,j}^H} \left( \frac{X_i - l}{m-1} \right)$$  \hspace{1cm} (C.14)

Exploiting the combinatorial identity $\sum_{i=0}^{n-1} \binom{n-1}{r-1} = \binom{n}{r}$, we find that

$$\left( \frac{X_i - U_{i,j}^H}{m} \right) - \left( \frac{X_i - U_{i,j}^A}{m} \right) = - \sum_{l=U_{i,j}^A+1}^{U_{i,j}^H} \left( \frac{X_i - l}{m-1} \right)$$  \hspace{1cm} (C.15)

Since $U_{i,j}^A \leq U_{i,j}^H < l$ in all terms of the summation

$$\left( \frac{X_i - U_{i,j}^H}{m} \right) - \left( \frac{X_i - U_{i,j}^A}{m} \right) \geq - \sum_{l=U_{i,j}^A+1}^{U_{i,j}^H} \left( \frac{X_i - U_{i,j}^A}{m-1} \right)$$  \hspace{1cm} (C.16)

$$\geq - \sum_{l=U_{i,j}^A+1}^{U_{i,j}^H} \left( \frac{X_i - U_{i,j}^A}{m-1} \right) = -(U_{i,j}^H - U_{i,j}^A) \left( \frac{X_i - U_{i,j}^A}{m-1} \right)$$  \hspace{1cm} (C.17)

Summing over \( j \) (\( 1 \leq j \leq J \)) completes the proof.$\square$
Lemma C.2.

\[
\sum_{j=J+1}^{d-K-1} \left[ \left( \frac{X_{i-1} - U_{i,j}^H}{m} \right) - \left( \frac{X_{i-1} - U_{i,j}^A}{m} \right) \right] \geq \left( \frac{X_{i-1} - U_{i,j}}{m-1} \right) \sum_{j=J+1}^{d-K-1} (U_{i,j}^A - U_{i,j}^H) \tag{C.20}
\]

Proof: This lemma proceeds in a similar fashion to Lemma C.1. For \( j \) \((J + 1 \leq j \leq d-K-1)\) we have

\[
\left( \frac{X_i - U_{i,j}^H}{m} \right) - \left( \frac{X_i - U_{i,j}^A}{m} \right) = \sum_{l=U_{i,j}^H + 1}^{U_{i,j}^A} \left( \frac{X_i - l}{m-1} \right) \tag{C.21}
\]

\[
\geq \sum_{l=U_{i,j}^H + 1}^{U_{i,j}^A} \left( \frac{X_i - U_{i,j}^A}{m-1} \right) \tag{C.22}
\]

\[
\geq (U_{i,j}^A - U_{i,j}^H) \left( \frac{X_i - U_{i,j}^A}{m-1} \right) \tag{C.23}
\]

Summing over \( j \) \((J + 1 \leq j \leq d-K-1)\) completes the proof \(\Box\)

We can now combine Lemmas C.1 and C.2 to find the lower bound for the sum in Equation C.10

\[
\sum_{j=1}^{d-K-1} \left[ \left( \frac{X_{i-1} - U_{i,j}^H}{m} \right) - \left( \frac{X_{i-1} - U_{i,j}^A}{m} \right) \right]
\]

\[
\geq - \left( \frac{X_{i-1} - U_{i,j}^A}{m-1} \right) \sum_{j=1}^{J} (U_{i,j}^H - U_{i,j}^A)
\]

\[
+ \left( \frac{X_{i-1} - U_{i,j}^A}{m-1} \right) \sum_{j=J+1}^{d-K-1} (U_{i,j}^A - U_{i,j}^H)
\]

\[
= \left( \frac{X_{i-1} - U_{i,j}^A}{m-1} \right) \sum_{j=1}^{J} (U_{i,j}^A - U_{i,j}^H)
\]

\[
+ \left( \frac{X_{i-1} - U_{i,j}^A}{m-1} \right) \sum_{j=J+1}^{d-K-1} (U_{i,j}^A - U_{i,j}^H)
\]

\[
= \left( \frac{X_{i-1} - U_{i,j}^A}{m-1} \right) \left( \sum_{j=1}^{J} (U_{i,j}^A - U_{i,j}^H) + \sum_{j=J+1}^{d-K-1} (U_{i,j}^A - U_{i,j}^H) \right)
\]
This proves that Equation C.10 is true, therefore, Equation C.4 holds. But since Equation C.4 implies Equation C.3, this shows that a heap has the lowest expected cost of sending a message to m destinations compared to any other compact tree.
BIBLIOGRAPHY


125


ACKNOWLEDGEMENTS

Words cannot express my eternal gratitude to my parents, Violeta and Hristo. They, together with my aunt, Rilka, and my late grandparents, made sure I was always surrounded by unconditional love, care and support through all the years, both in Bulgaria and in the USA. They have made a lot of personal sacrifices to help me become who I am right now. Any achievement that I have made is based on their persistent help and encouragement ever since my childhood.

I would like to thank my co-major professors, Dr. Giora Slutzki and Dr. Steven M. LaValle, for their academic guidance as well as personal support. Dr. Slutzki has been behind me throughout my numerous years at Iowa State University. He helped me during hard times, at moments when I was drifting without direction. He also encouraged me later when I had ideas worth pursuing. His deep knowledge of theoretical computer science has been an invaluable resource for me. Dr. LaValle has provided me with a strong boost by introducing me and motivating me to work in pursuit-evasion. He has given me the rare opportunity and resources to work in an area which is both practical and mentally challenging. He has patiently given me answers in a wide spectrum of fields, varying from practical C programming tricks all the way to deep theoretical insights in topology.

I have thoroughly enjoyed working on several papers with Dr. Srini Tridandapani. He together with Dr. Michael Borella and Dr. Jason Jue have introduced me to interesting topics in optical networks.

I would like to thank the members of my POS committee, Dr. Chaudhuri, Dr. Cruz-Neira and Dr. Fernández-Baca, as well as all the faculty at Iowa State for their advice which directly or indirectly has helped me learn and grow. In particular, Dr. Jack Lutz who was my M.S. advisor, continued teaching and supporting me long after my M.S. degree was granted.

There are many people from Bulgaria who have made my studying in the USA possible. Among them, Dr. Dimitar Shishkov, professor at Sofia University, and Mr. Ogaian Drenski, my former boss and mentor at CUV-Sofia, have helped and encouraged me while I was a student in Bulgaria, providing me with guidance and resources to pursue my interests in Computer Science.

There are quite a few people who have given me valuable feedback on my research, including
Peng Cheng, Kishore Dhara, Stjepan Rajko, Krishnan Subramaniam, Libo Yang, fellow students from Iowa State, and Jose Castejon-Amenedo, Ron Luman and Ratan Nalumasu, co-workers from Hewlett-Packard. I appreciate the support of my managers Mo Mohundro and Robert Schwehr, who made it possible for me to finish up my dissertation while working at Hewlett-Packard.

Last, but by far not least, I would like to express my boundless love and appreciation to my son, Christo, and my dear wife, PeRu. Christo’s smile has always given me an extra boost of energy to work. PeRu has been with me during the glory moments, but more importantly she has stayed beside me during the dark times when her support has been essential. She took bigger burdens on herself to allow me to concentrate on my studies. She was the one who kept encouraging me, she motivated me to believe in myself. Thank you!