Resilience of Wireless Sensor Networks

by

Kuan-Chieh Robert Tseng

B.Sc., University of British Columbia, 2009

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES
(Computer Science)

The University Of British Columbia
(Vancouver)

April 2011

Abstract

The coverage of a wireless sensor network is a measure of the quality of service. One type of coverage is k-*barrier coverage*. Given a starting region S and an ending region T, we say that a sensor network has k-barrier coverage with respect to S and T if any $S - T$ path in the surveillance domain must intersect the coverage regions of at least k sensors.

In this thesis, we focus on determining the *resilience* of a wireless sensor network. The resilience is defined to be the minimum number of sensors that need to be removed in order to ensure the existence of a $S - T$ path that does not cross any sensor coverage region. A sensor network with resilience k constitutes a k-barrier coverage.

We demonstrate that determining resilience of a wireless sensor network with 2D surveillance domain is NP-hard for the case when the sensor coverage regions are unit line segments. Furthermore, it is possible to extend the reduction to show that the problem remains NP-hard for other types of sensor coverage regions. In general, if the shape of the coverage region is non-symmetric, then determining resilience is NP-hard.

We also investigate the problem of determining resilience of a wireless sensor network with 3D surveillance domain. In this case, we show that if the coverage regions of the sensors are unit spheres, then the problem is NP-hard.

Table of Contents

List of Figures

List of Algorithms

Acknowledgments

I like to express my deepest gratitude to my supervisor, David Kirkpatrick, for guiding and encouraging me through the entire degree. He not only introduced me to the problem, but provided crucial insights and ideas. This thesis would not have been completed without his constant support and guidance.

I would also like to thank Will Evans for agreeing to read my thesis and providing many helpful suggestions and improvements.

Finally, I like to thank my family and friends for their support and care. Without their encouragements, I could not have completed the thesis.

Chapter 1

Introduction

1.1 Background

A *wireless sensor networks* consists of autonomous *sensors* spatially distributed over a *surveillance domain*. The sensors are responsible for monitoring a particular aspect of the surveillance domain (eg. security cameras monitoring for intrusion). A fundamental problem in wireless sensor network is known as the *coverage problem*. The problem focuses on measuring how well does a sensor network monitor its domain. This can be seen as measuring the quality of service of the sensor network. An overview of the research work that have been done in this area can be found in the survey paper of Cardei and Wu [2] as well as the Ph.D. thesis of Kumar [6].

In general, coverage problems can be expressed geometrically. We represent each sensor by the shape of its *coverage region*, which specifies the set of points that are *covered* by the sensor. For brevity, we use the coverage region shape as the name of the sensor type (eg. disk sensors to denote sensors with a disk shape coverage region).

Depending on the motivation, we can investigate different types of coverage. Cardei and Wu [2] classified the coverage into three categories: *area coverage*, *point coverage*, and *barrier coverage*. In area coverage, the goal is to make sure that every point in the domain is covered by at least one sensor. In point coverage, the goal is to ensure that some specified points of interest are covered by at least one sensor. Lastly, in barrier coverage, the goal is to make sure that all paths joining a start region S to a target region T must intersect at least one sensor.

Coverage problems have many potential applications. For example, the notion of barrier coverage can be applied to border control [7]. Point coverage problems also have militaristic applications, where monitoring "high priority" enemy targets is useful. In [3], coverage concepts are applied to many-robots system with the goal of using unmanned robots to complete missions such as mine sweeping and sentry duty. Another application is the *ocean coverage problem* [5], where the authors analyzed the sensor coverage for detecting ocean phytoplankton by merging data from orbit satellites. Other applications are described in [6], including intrusion detection and habitat monitoring.

1.2 Problem Statement

In this thesis, we will be expanding upon the notion of barrier coverage. Barrier coverage is attractive because it guarantees that there are no undetected transitions from the start region S to the end region T. Furthermore, the number of sensors required to achieve barrier coverage is usually less than that required to cover the entire surveillance domain with sensors. However, a key weakness in barrier coverage is the possibility of a single point of failure - if one sensor breaks down, the remaining sensors may not form a barrier.

One way to increase the robustness of barrier coverage is the notion of k*-barrier coverage*, introduced by Kumar et al. [7] in the context of measuring the coverage of a sensor network constrained within a belt region. A sensor network is said to provide k-barrier coverage if every path from the start region S to end region T intersects at least k distinct sensors. This notion is further generalized into the concepts of *resilience* and *thickness* by Bereg and Kirkpatrick [1]. Given a sensor network and two regions S and T in the surveillance domain, the *thickness* of the network is the minimum over paths from S to T of the number of times the path enters a sensor region. The *resilience* of the network is the minimum number of sensors whose removal permits a path from S to T that does not intersect any sensor region.

Equivalently, the resilience of the network is the minimum over paths from S to T of the number of **distinct** sensors the path intersects (we denote the path that achieves the minimum as the *minimal resilience path*). Note that using this definition, resilience and thickness are conceptually similar. The key difference between

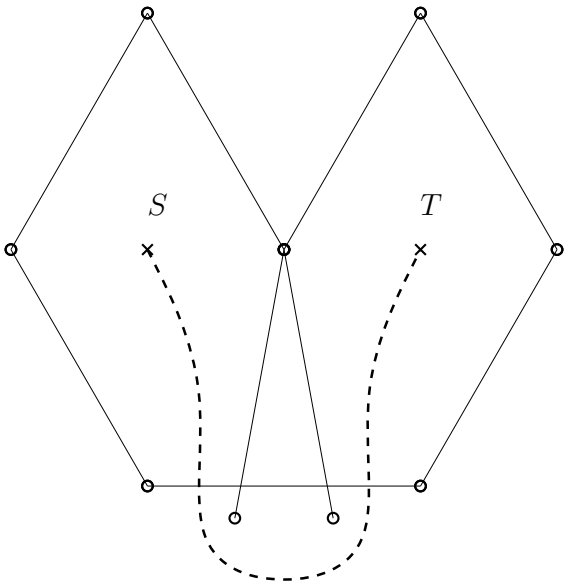


Figure 1.1: A sensor network with thickness 2 but resilience 1. The dashed line is the *minimum resilience path*.

the two is that thickness counts every visit to the same sensor while resilience only counts one. This distinction is illustrated by the sensor network depicted in Figure 1.2, where the coverage region of each sensor is a line segment.

The first question that we investigate in the thesis is: *"what is the complexity of determining the resilience of a sensor network whose surveillance domain consists of the entire 2D plane?"* We only consider the special case where both S and T are merely 2D points on the plane instead of regions. However, the distinction between using points or regions is not significant. In general, a sensor network partitions the plane into sub-regions where all points in the same sub-region are "equivalent" in terms of resilience.

A sensor network is described by a tuple $(U, \mathcal{A})$, where U is a set of sensors described by the shape of their coverage regions and $\mathcal{A}$ is an arrangement of the sensors in U on the 2D plane. More precisely, $\mathcal{A}$ is a function $\mathcal{A} : U \to \mathbb{R}^2$ mapping each sensor to a position on the 2D plane. We only consider the case where the coverage regions of every sensor in U are the same. This is known as a *homogeneous sensor network*.

As noted in [1] for the case of unit disk sensors, calculating the thickness of a sensor network $(U, \mathcal{A})$ for any two points S and T can be reduced to a shortest path problem. First, we may remove any sensors whose coverage regions include S or T.

This is because these sensors will be inevitably intersected and are not interesting. Next, we construct the dual graph of the arrangement of the sensor regions where the vertices are the faces and edges connect two vertices if the corresponding faces are adjacent. We assign each edge the weight 1 if it corresponds to a transition that enters a sensor region and weight 0 if it corresponds to a transition that exits a sensor region. The thickness is simply the shortest path from the vertex representing the face containing S to the vertex representing the face containing T. This approach may be used regardless of the shape of the coverage regions of the sensors.

Computing resilience, on the other hand, is not so easy. In particular, we will show that if the coverage region of every sensor in the network is a unit line segment, then computing the resilience is NP-complete. Furthermore, the proof may be extended to show that for any non-symmetric sensor coverage region, determining the resilience is NP-complete.

We will also consider the resilience problem for sensor networks whose domain is the entire 3D space. In this case, the arrangement function $\mathcal{A}$ is changed so that it maps every sensor to a coordinate in 3D space. We show that for a unit sphere sensor network, determining resilience is NP-complete as well.

1.3 Related Work

Kumar et al. [7] investigated the problem of determining resilience of a disk sensor network for the special case that all sensors must lie within a belt region that separates S from T. The belt region may be open, in which case the region resembles a strip in the plane, or closed, in which case the region resembles a ring. They proved that in the open belt case, computing resilience can be solved in polynomial time using maximum flow. However, the problem of determining resilience in the closed belt case remains open. The problem we investigate in this thesis can be viewed as a generalization of this problem.

The problem of approximating resilience for a sensor network in the general case was studied in [1]. Their main result is a polynomial time 2-approximation algorithm when the coverage region of each sensor is a unit disk. The authors also presented some evidence which suggests that computing the resilience may be hard.

We originally studied resilience as a more robust extension to barrier coverage.

There are also other ideas to increase the robustness. For example, Meguerdichian et al. [9] suggested measuring the coverage of a sensor network using a *maximal breach path* - a path that maximizes the distance to the closest sensor. A closely related idea is a *minimum exposure path* [10] - a path that minimize the total degree of "exposure" to sensors. Other models of studying barrier coverage introduces probabilistic assumptions [8].

1.4 Layout of Thesis

In Chapter 2 and 3, we will prove that determining resilience for a sensor network whose coverage regions are unit line segments is NP-complete. The former chapter focuses on the concept of the reduction while the latter focuses on the details of the reduction. We will also show how the proof may be extended to other non-symmetric sensor coverage region in Chapter 3. Finally, in Chapter 4, we will prove that determining resilience for a 3D sensor network with unit sphere coverage region is also NP-complete.

Chapter 2

Complexity of Computing the Resilience of Unit Line Segment Sensor Networks

The focus of this chapter is to show that determining the resilience of a unit line segment sensor network is NP-complete. We will focus on the core concepts in this chapter and leave the details of the reduction to Chapter 3.

Resilience of Unit Line Segment Sensor Network (ULS-RES)
Instance: $(U, \mathcal{A}, S, T, B)$, where U is the set of unit line segment sensors, $\mathcal{A}$ is an arrangement of U, S and T are two points in the 2D plane, and B is a positive integer.
Question: Is it possible to traverse from S to T without intersecting any sensors by removing at most B sensors from U?

The problem we will reduce from is the MAX-2-SAT problem, which has been proven to be NP-complete [4]. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of Boolean variables. A 2-CNF clause over X is of the form $c = l_1 \vee l_2$, where l_1, l_2 are literals of variables of X. A literal is either a negated variable $\overline{x_i}$ (a *negative literal*) or an unnegated variable x_i (a *positive literal*).

A *truth assignment* of X is a function $t : X \to \{\textit{TRUE}, \textit{FALSE}\}$. A positive literal $l = x_i$ is satisfied by t if $t(x_i) = \textit{TRUE}$. A negative literal $l = \overline{x_i}$ is satisfied

by t if $t(x_i) =$ *FALSE*. A 2-CNF clause is satisfied by t if at least one of the two literals of the clause is satisfied by t.

Maximum 2 Satisfiability (MAX-2-SAT)
Instance: $\{X, C, K\}$, where $X = \{x_1, x_2, \ldots, x_n\}$ is the set of Boolean variables, $C = \{c_1, c_2, \ldots, c_m\}$ is the set of clauses over X in 2-CNF form, K is a positive integer.
Question: Does there exists a truth assignment of X that simultaneously satisfies at least K clauses in C?

We will present the reduction through several layers of abstractions. The MAX-2-SAT problem is first reduced to a node colouring problem, which is subsequently reduced to an edge colouring problem. Finally, we reduce to the ULS-RES problem.

Before proceeding to the actual reduction, we make one final remark. In our discussions so far, the sensor network consists purely of sensors. Thus, there are no restrictions on the $S-T$ paths. It is natural to consider the case where in addition to a sensor network, we also have an "environment" consisting of a series of "natural barriers" where no $S - T$ path may cross. It is clear that the addition of these obstacles can dramatically change the resilience of a sensor network.

However, we note that it is possible to simulate a barrier with sensors. Suppose we are given a sensor network with N sensors and we wish to add a barrier. Let us overlay $N + 1$ sensors at the location of the barrier. Any $S - T$ path which crosses this cluster of sensors will intersect at least $N + 1$ distinct sensors, worse than if the path simply crosses every sensor in the original network. Thus, when calculating resilience, we can safely ignore any path which passes through these $N+1$ sensors. Therefore, this cluster of $N+1$ sensor can be viewed as an obstacle that limit the $S - T$ path. This technique allows us to create obstacles of shapes that can be represented as a union of sensors. Consequently, even though the ULS-RES problem only specifies a sensor network, we can also specify obstacles that no $S - T$ path can cross. We will be exploiting this idea in the future.

2.1 Reduction to Bipartite Bi-Vertex Node Colouring

We begin by reducing the MAX-2-SAT problem to a problem of node colouring a bipartite graph. The reduction demonstrates how to transform a logic problem to a graph problem. It also provides us with a way of visualizing the MAX-2-SAT problem, which we will expand upon in later sections.

2.1.1 Problem Statement

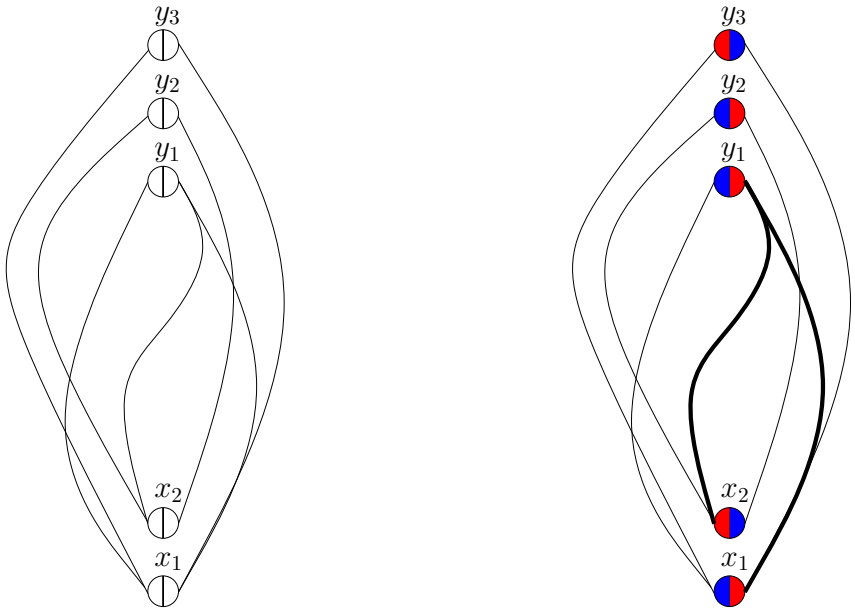
A *bi-vertex* v is a special type of node which consists of two halves: the positive half (v^+) and the negative half (v^-). A *bi-vertex graph* is a graph $G = (V, E)$ where V is the set of bi-vertices and $E \subseteq V^{\{+,-\}} \times V^{\{+,-\}}$ is the set of edges that connects two bi-vertices in V. Note that an edge not only specifies the bi-vertices it connects, but also which half of the bi-vertex it connects to. Furthermore, each edge can connect to exactly one of the two halves of a bi-vertex. It is important to note that edges connected to the same bi-vertices but different halves are distinguishable. For example, an edge that connects v_1^+ to v_2^+ is different from one that connects v_1^- to v_2^+, where v_1, v_2 are bi-vertices.

A bi-vertex graph is *bipartite* if it is possible to partition V into two sets V_1 and V_2 such that no edge connects two bi-vertices in V_1 or two bi-vertices in V_2. More formally, we describe a *bipartite bi-vertex graph* by a triplet (V_1, V_2, E), where V_1, V_2 denote sets of bi-vertices and $E \subseteq V_1^{\{+,-\}} \times V_2^{\{+,-\}}$ denotes the set of edges connecting bi-vertices in V_1 to V_2. We will insist that there are no multi-edges in our graph. The importance of this will arise in Section 3.1.5.

Given a bi-vertex graph, a *colouring scheme* colours each bi-vertex in one of the two following configurations:

1. Positive half red and negative half blue (denoted by *red-blue*)
2. Negative half red and positive half blue (denoted by *blue-red*)

An edge in the graph is *conflicting* if it connects the red half of a bi-vertex in V_1 to a red half of a bi-vertex in V_2.



(a) A bipartite bi-vertex graph

(b) A colouring scheme

Figure 2.1: A bipartite bi-vertex graph and a colouring scheme. Conflicting edges are highlighted in bold.

Bipartite Bi-vertex Node Colouring (NODE-COLOUR)
Instance: (G, L), where $G = (V_1, V_2, E)$ is a bipartite bi-vertex graph and L is a positive integer.
Question: Does there exists a colouring scheme for G where the number of conflicting edges is at most L?

An example of a bipartite bi-vertex graph and a colouring scheme is shown in Figure 2.1. In the figure, all bi-vertices are oriented such that the right side represents the positive half and the left side represents the negative half. Note that if we change the colour configuration of y_1, the resulting colouring scheme will have zero conflicting edges.

2.1.2 Reduction

Let $P = (X, C, K)$ be an instance of MAX-2-SAT. The first step in the reduction is to construct the bipartite bi-vertex graph $G = (V_1, V_2, E)$. We represent

each variable in the MAX-2-SAT instance by a bi-vertex in V_1 and each clause by a bi-vertex in V_2. We will denote bi-vertices in V_1 and V_2 by *variable bi-vertices* and *clause bi-vertices*, respectively. For each clause $c = l_1 \vee l_2$, we add two edges to E:

- Suppose l_1 represents the variable x_1 and l_2 represents the variable x_2
- The first edge will connect the variable bi-vertex x_1 to the **positive** half of the clause bi-vertex c
- The second edge will connect the variable bi-vertex x_2 to the **negative** half of the clause bi-vertex c
- To determine which half of the variable bi-vertex the edge connects to, we use the following rules:
  - If the literal is a negative literal, the edge connects to the **negative** half.
  - If the literal is a positive literal, the edge connects to the **positive** half.

Finally, to complete the reduction, we set $L = m - K$, where m is the number of clauses.

Figure 2.2 depicts the bipartite bi-vertex graph obtained from the instance of MAX-2-SAT with $X = \{x_1, x_2, x_3, x_4\}$ and $C = \{(x_1 \vee x_2), (x_3 \vee \overline{x_2}), (\overline{x_1} \vee \overline{x_4}), (x_3 \vee \overline{x_1}), (x_4 \vee \overline{x_3}), (\overline{x_4} \vee \overline{x_2})\}$. The positive and negative half of each bi-vertex is the right and left half, respectively.

Theorem 2.1. *Given a MAX-2-SAT instance $P = (X, C, K)$, the reduction to an instance of NODE-COLOUR $P' = (G, L)$ can be accomplished in $O(|X| + |C|)$ time.*

Proof. Setting the value of L takes $O(1)$ time. Constructing the bipartite bi-vertex graph G is $O(|V_1| + |V_2| + |E|)$. By construction, we know $|V_1| = |X|, |V_2| = |C|$. Lastly, observe that we add two edges per clause, so $|E| = 2|C|$. Thus, the reduction takes $O(n + m)$ time. □

2.1.3 Proof of Correctness

Let $P = (X, C, K)$ be an instance of MAX-2-SAT and $P' = (G, m - K)$ be an instance of NODE-COLOUR constructed from P as described in the previous section.

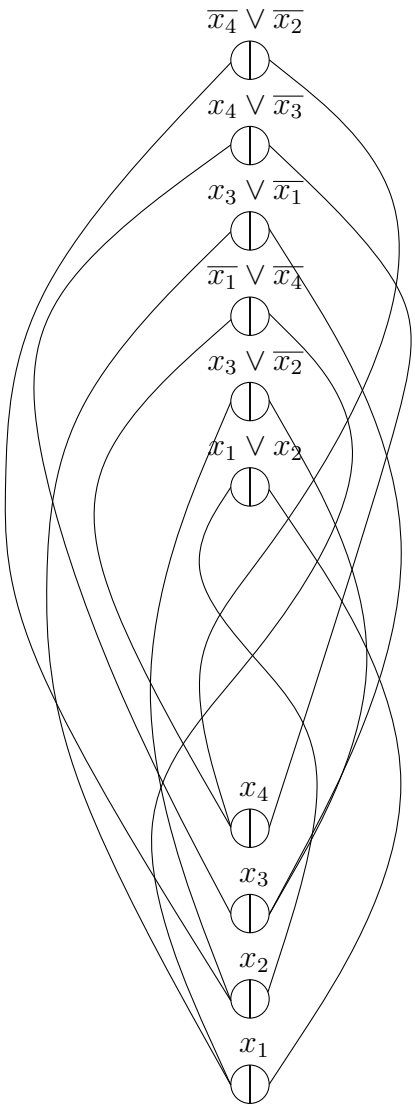


Figure 2.2: Example of reduction from MAX-2-SAT to NODE-COLOUR

Lemma 2.2. *If there exists a truth assignment of X which simultaneously satisfies at least K clauses in C, then there exists a colouring scheme for G with at most $m - K$ conflicting edges.*

Proof. Let t be the truth assignment of X which simultaneously satisfies at least K clauses in C. Consider the following colouring scheme for G: for each variable bi-vertex, we colour it **red-blue** if the corresponding variable is assigned *FALSE* and **blue-red** if the variable is assigned *TRUE*. For clause bi-vertices, we colour a clause bi-vertex **red-blue** if and only if the first literal of the clause is satisfied by t. Otherwise, we colour it **blue-red**.

We claim that the number of conflicting edges using this colouring scheme is at most $m - K$. Consider a conflicting edge and assume that it connects the variable bi-vertex representing x_i to the clause bi-vertex representing c_j. There are four cases to consider.

The first case is if the edge connected x_i^+ to c_j^+. By construction, this means that x_i appeared as an unnegated variable in the clause c_j and is the first literal. Furthermore, since x_i^+ is coloured red, it follows that $t(x_i) =$ *FALSE*. Therefore, c_j could not have been satisfied by the positive variable x_i. However, since c_j^+ is red, it must be the case that we coloured c_j **red-blue**. This can only occur if c_j is satisfied by the first literal - a contradiction.

The second case is if the edge connected x_i^+ to c_j^-. Similar to the previous case, we can argue that x_i appeared as an unnegated variable in c_j and is the second literal. Furthermore, c_j is not satisfied by the second literal. Since c_j is coloured **blue-red**, it must also be the case that c_j is not satisfied by the first literal. Thus, c_j must be an unsatisfied clause.

The third case is if the edge connected from x_i^- to c_j^+. We can argue similar to the first case that this is impossible. Finally, the last case is if the edge connected x_i^- to c_j^-. We can argue similar to the second case that this implies c_j is an unsatisfied clause.

Lastly, observe that every conflicting edge must connect to a unique clause bi-vertex. This is because exactly one edge connects to each half of a clause bi-vertex and only one half is coloured red. Thus, every conflicting edge in the colouring scheme corresponds to a unique clause that is unsatisfied by t. Since t satisfies at least K clauses, it follows that the maximum number of unsatisfied clauses is $m - K$. Consequently, the maximum number of conflicting edges of the colouring scheme is $m - K$. □

Lemma 2.3. *If there exists a colouring scheme for G with at most $m-K$ conflicting edges, then there exists a truth assignment of X that simultaneously satisfies at least K clauses in C.*

Proof. Consider the colouring scheme of G with at most $m - K$ conflicting edges. Construct the truth assignment t as follows: for each variable x_i, we assign it *TRUE* if the corresponding variable bi-vertex in G is coloured **blue-red** and assign it *FALSE* if the corresponding bi-vertex is coloured **red-blue**. We will now show that

for every clause that is unsatisfied by t, there is a corresponding conflicting edge in the colouring scheme. This completes the proof since this implies that maximum number of unsatisfied clause by t is $m - K$. Consequently, t must satisfies at least K clauses.

Suppose the clause $c = l_1 \vee l_2$ is unsatisfied. Consider the clause bi-vertex c in G. If c is coloured **red-blue**, then consider the edge which connects the variable bi-vertex representing l_1 to c (for brevity, suppose l_1 represents the variable x). First, we know that this edge is connected to the red half of c by definition. We claim that the edge also connects to the red half of the variable bi-vertex x. There are two cases to consider: $l_1 = x$ or $l_1 = \overline{x}$. For the first case, the edge must connect to x^+. Since c is unsatisfied, we know that $t(x) =$ *FALSE*. Thus, the bi-vertex x must have been coloured **red-blue**. Consequently, x^+ is coloured red and the edge does indeed connect two red halves. If $l_1 = \overline{x}$, we can make a similar argument to show that $t(x) =$ *TRUE* and the bi-vertex must have been coloured **blue-red**. Thus, the edge is conflicting in this case as well.

Similarly, if c is coloured **blue-red**, then we can show the edge that connects the variable bi-vertex representing l_2 to c is a conflicting edge. □

Theorem 2.4. *NODE-COLOUR is NP-complete*

Proof. It is easy to see that NODE-COLOUR $\in$ NP - a certificate consists of a colouring scheme. Since MAX-2-SAT is NP-hard, direct applications of Theorem 2.1 and Lemma 2.2 and 2.3 shows that NODE-COLOUR is also NP-hard. □

2.2 Reduction to Bipartite Four-Chain Edge Colouring

As mentioned before, the NODE-COLOUR problem gives us some insight on modelling the MAX-2-SAT problem pictorially. As presented, there is a vital difference between the NODE-COLOUR and the ULS-RES problem. The goal of the NODE-COLOUR problem is to determine the existence of a node colouring scheme with a constrained number of conflicting edges. However, in the context of a sensor network, the notion of a "node" does not exist. When describing a sensor network, all we have is a series of the sensors and their locations. On the other hand, edges of the graph are more related to sensors. For example, if we view the

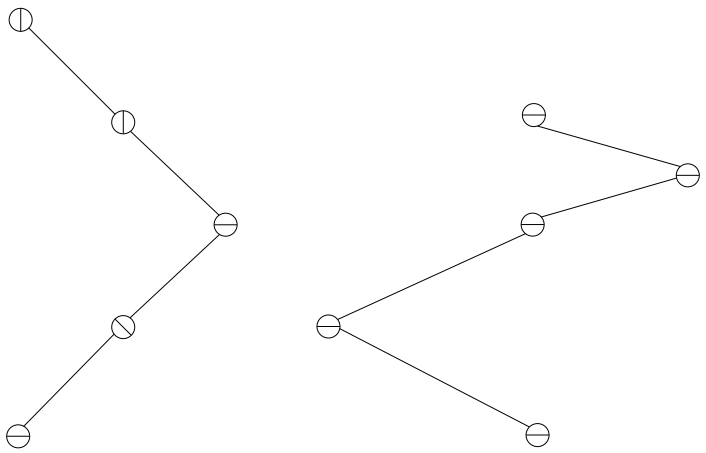


Figure 2.3: Example of a four-chain

graph presented in Figure 2.2 as an embedded graph, the edges of the graph become 2D constructs with locations, similar to sensors. In this section, we will bridge this gap by reducing the the bi-vertex colouring problem to an edge colouring problem.

2.2.1 Problem Statement

We introduce a new graph construct called a *four-chain*. A four-chain consists of five bi-vertices and four edges connecting these bi-vertices in a chain. Furthermore, each of the three middle bi-vertex has two incident edges, one to each half. Two examples are shown in Figure 2.3.

A *bipartite four-chain graph* is a graph $G = (V_1, V_2, E)$, where V_1, V_2 denotes sets of bi-vertices and $E \subseteq V_1^{\{+,-\}} \times V_2^{\{+,-\}}$ denote the set of four-chains connecting bi-vertices in V_1 to V_2. Essentially, a bipartite four-chain graph is a bipartite bi-vertex graph where every edge is replaced by a four-chain. Again, we insist that the graph has no multi-edges.

An *edge colouring scheme* of a bipartite four-chain graph assigns each edge the colour *red* or *blue*. Note that we are colouring each individual edge in the four-chain and not the chain as a whole. Thus, for a single four-chain, different edges may be coloured differently. We call an edge colouring scheme *satisfying* if for each bi-vertex in the graph (including the bi-vertices inside a four-chain), one of the following two conditions is satisfied:

- All positively incident edges are coloured red

- All negatively incident edges are coloured red

Note that a satisfying edge colouring scheme may choose to colour both positively and negatively incident edges of a bi-vertex red.

Bipartite Four-Chain Edge Colouring (EDGE-COLOUR)
Instance: (G, R), where $G = (V_1, V_2, E)$ is a bipartite four-chain graph and R is a positive integer.
Question: Does there exists a satisfying colouring scheme for G which colours at most R edges red?

For a satisfying colouring scheme, a four-chain has the special property that for any two consecutive edges, at least one of the two needs to be coloured red. A direct corollary of this is that for every four-chain, at least two edges need to be coloured red in any satisfying colouring scheme.

Finally, the choice of using a four-chain instead of using chains of other lengths is for simplicity. As we will see later, the important condition is that the chain lengths must be even. For now, using four satisfies this constraint while providing a simple framework for us to work in. On the other hand, four provides enough complexity for us to demonstrate the core idea behind the reduction that we will further exploit in later sections.

2.2.2 Reduction

Given an instance NODE-COLOUR $P = (G, L)$, where $G = (V_1, V_2, E)$, the reduction to an EDGE-COLOUR instance $P' = (G', R)$ is simple. We construct the bipartite four-chain graph $G' = (V_1', V_2', E')$ by replacing every edge in G by a four-chain. Figure 2.4 depicts this process for the instance of NODE-COLOUR shown in Figure 2.2. We have purposely drawn the bi-vertices in the bipartite chain smaller than the bi-vertices in V_1 and V_2. Bi-vertices in V_1 and V_2 are oriented such that the right and left half are the positive and negative half, respectively. Each bi-vertex in a four-chain is oriented such that the top edge is connected to the positive half and the bottom edge is connected to the negative half. Finally, we set $R = 2|E| + L$.

Theorem 2.5. *Given an instance $P = (G, L)$ of NODE-COLOUR, where $G = (V_1, V_2, E)$ is a bipartite bi-vertex graph. The reduction to $P' = (G', R)$ of EDGE-*

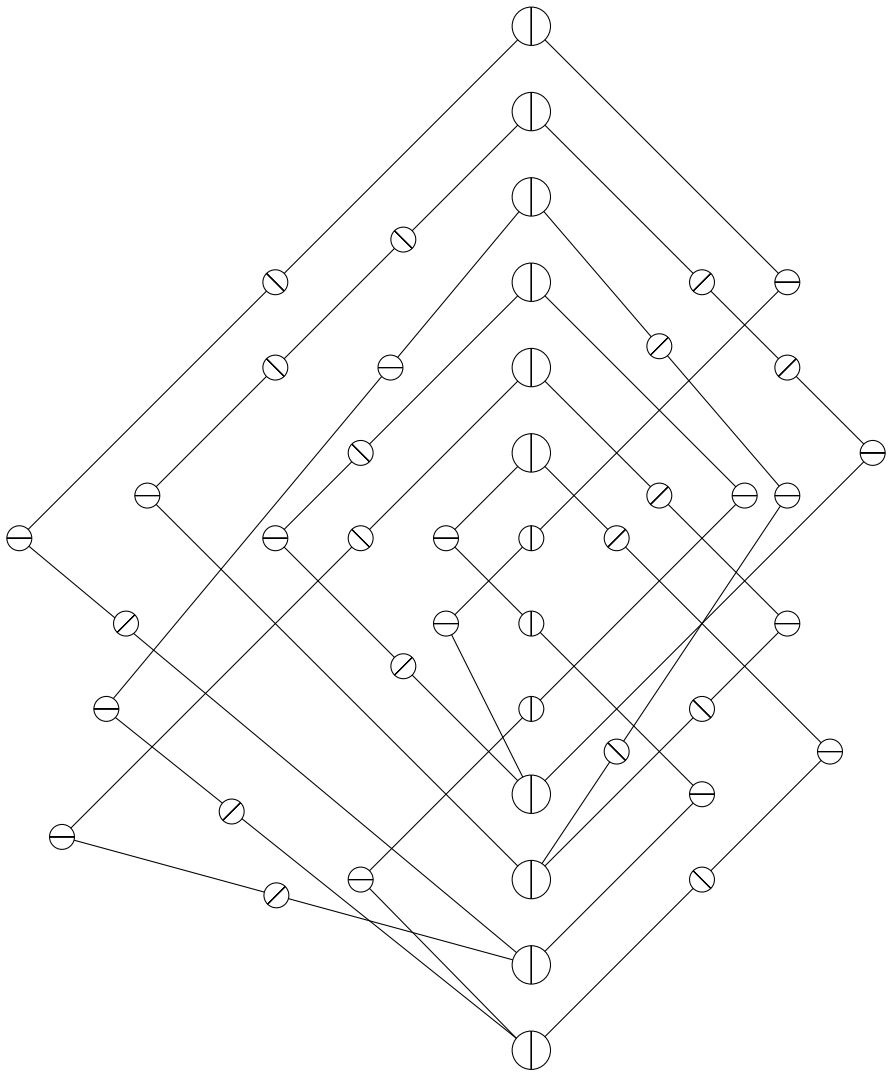


Figure 2.4: Example of reduction from NODE-COLOUR to EDGE-COLOUR

COLOUR can be realized in polynomial time.

Proof. Setting the value of R is constant time. Replacing each edge of P by a four-chain is also constant time. Since we do the replacement once per edge, the overall reduction is polynomial time. □

2.2.3 Proof of Correctness

Let $P = (G, L)$ be an instance of NODE-COLOUR, where $G = (V_1, V_2, E)$ is a bipartite bi-vertex graph. Let $P' = (G', 2|E| + L)$ be an instance of EDGE-

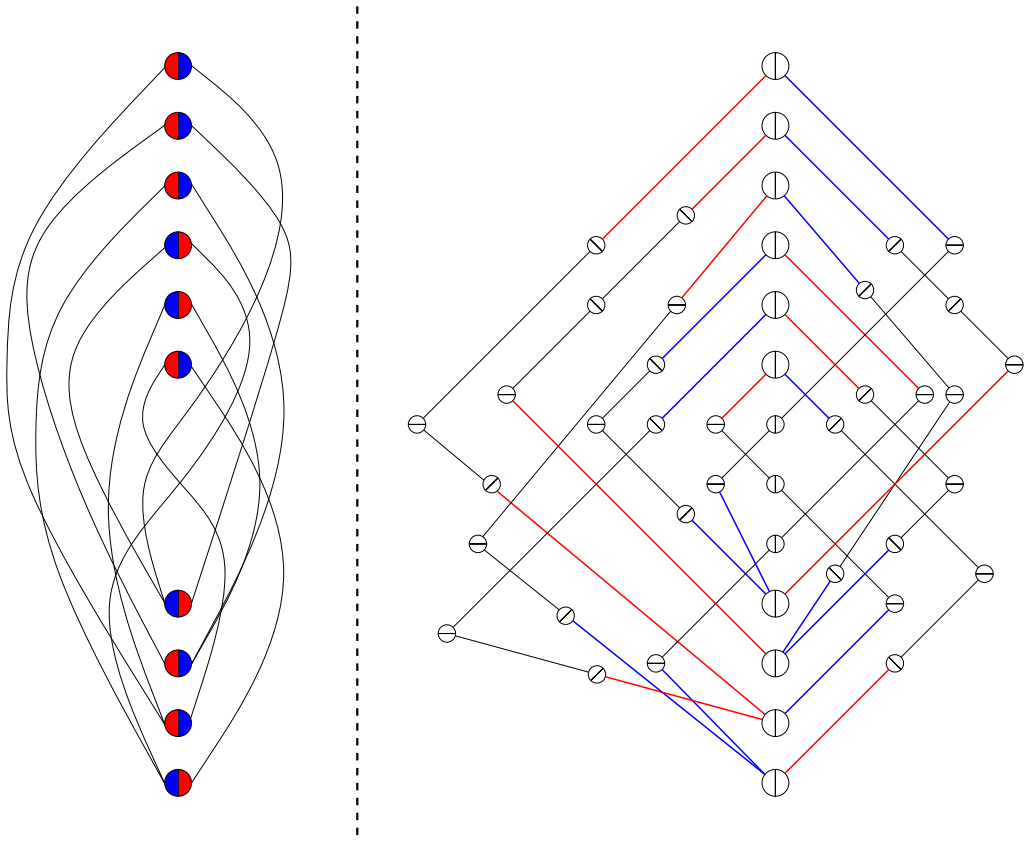


Figure 2.5: Constructing the first part of an edge colouring scheme

COLOUR constructed from P as described in the previous section.

Lemma 2.6. *If there exists a node colouring scheme for G with at most L conflicting edges, then there exists a satisfying edge colouring scheme for G' with at most $2|E| + L$ red segments.*

Proof. Consider the node colouring scheme for G with at most L conflicting edges. Construct an edge colouring scheme for G' as follows. We first iterate through all bi-vertices v in V_1'. If v is coloured **red-blue**, then we colour all positively incident edges of v **red** and all negatively incident edges **blue**. Conversely, if v is coloured **blue-red**, we colour all positively and negatively incident edges of v **blue** and **red**, respectively. Next, we repeat the same process for every bi-vertex in V_2'. An example of this process is illustrated in Figure 2.5.

By now, it must be the case that for every four-chain, the two end edges are already coloured. For each four-chain, we colour the remaining two edges according

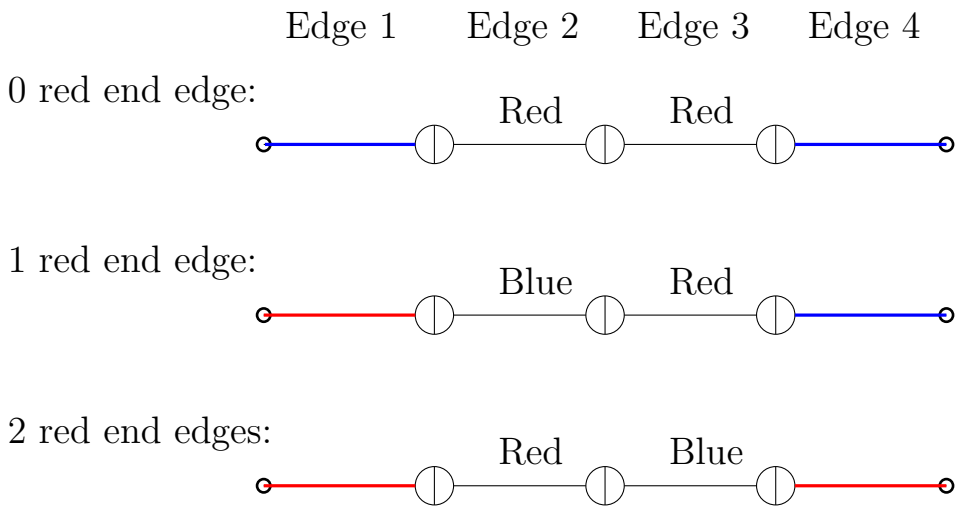


Figure 2.6: Colouring a four-chain given the colour of the end edges

to the number of end edges that are coloured red. Let us lay a four-chain horizontally and label the edges 1 to 4 from left to right. There are three cases to consider:

1. Neither end are coloured red: we colour edge 2 and 3 red.
2. One end is coloured red: without loss of generality, assume edge 1 is coloured red and edge 4 is coloured blue. We colour edge 3 red.
3. Both ends are coloured red: we colour edge 2 red.

These cases are illustrated by Figure 2.6.

It is easy to see that every bi-vertex inside a four-chain is satisfied by the edge colouring scheme. By construction, every bi-vertex in V_1 or V_2 is satisfied by the colouring scheme as well. Thus, this is a satisfying colouring scheme.

In the edge colouring scheme, two edges of a four-chain are coloured red in the first two cases and three edges in the last case. However, we note that for both end edges of a four-chain to be coloured red, the corresponding edge in G must be a conflicting edge. Thus, the number of four-chains that belong to the last case must be less than or equal to L. Consequently, the number of red segments is $\leq 3L + 2(|E| - L) = 2|E| + L$. □

Lemma 2.7. *If there exists a satisfying edge colouring scheme for G' with at most $2|E|+L$ red segments, then there exists a node colouring scheme for G with at most L conflicting edges.*

Proof. Consider the satisfying edge colouring scheme for G' with at most $2|E| + L$ red segments. Recall that for a satisfying colouring scheme, it must colour at least two edges of each four-chain red. Thus, the number of four-chains for which three or more edges are coloured red in this scheme must be $\leq L$.

We construct the node colouring scheme for G as follows. For each bi-vertex v in G, we look at the colour of all the positively and negatively incident edges of the corresponding bi-vertex in G'. We colour v **red-blue** if and only if all positively incident edges are coloured red. Otherwise, we colour v **blue-red**.

Now consider a conflicting edge under the constructed node colouring scheme. By construction, the corresponding four-chain in G' must have both of its end edges coloured red. Furthermore, since we have a satisfying edge colouring scheme for G', it follows that one of the middle two edges must also be coloured red in order to satisfy the middle bi-vertex of the four-chain. Referring back to Figure 2.6, this corresponds to the bottom-most case. It can be seen that at least one of edge 2 or 3 must be coloured red by any satisfying colouring scheme. Therefore, at least three edges of this four-chain are coloured red.

Thus, each conflicting edge in G corresponds to a four-chain in G' where three or more edges are coloured red. By the previous observation, this must be $\leq L$. □

Theorem 2.8. *EDGE-COLOUR is NP-complete.*

Proof. We know that NODE-COLOUR is NP-complete by Theorem 2.4. Direct application of Theorem 2.5 and Lemma 2.6 and 2.7 yields that EDGE-COLOUR is NP-hard. Finally, we note that EDGE-COLOUR is in NP - a certificate consists of an edge colouring scheme. In polynomial time, we can verify that the colouring scheme is satisfying and count the number or red edges. □

2.2.4 Relationship to Resilience of a Sensor Network

So far, we have been working with colouring problems, which seem unrelated to determining the resilience of a sensor network. Here, we provide the high level intuition on how the EDGE-COLOUR problem is related to the ULS-RES problem.

Consider the example shown in Figure 2.4. We have purposely drawn all the edges of the graph as straight line segments. Let us pretend these line segments are line segment sensors placed on the 2D plane at the exact same location (ignoring

for now the constraint that all line segments sensors need to have the same length). Bi-vertices represent 2D points where the end points of multiple sensors happen to coincide. We partition the sensors (segments) whose end points are incident to a common 2D point into positively and negatively incident sensors based on a line through the point: those segments above (or to the right of) the line are positive; those below (or to the left of) the line are negative. Consequently, these 2D points still retain the "two-halves" property: any sensor whose end points coincide to a bi-vertex connects either to the positive half or the negative half.

Let us suppose that we have placed S and T and embedded the network in an environment of obstacles such that every bi-vertex satisfies the following "magical" property:

Property 2.1. (Magical Property) Any $S - T$ path must either intersect all positively incident sensors or all negatively incident sensors of a bi-vertex.

This property is very similar to the conditions of a satisfying colouring scheme - we have simply replaced the notion of colouring an edge red by crossing the corresponding sensor with an $S - T$ path.

Using this translation, a satisfying colouring scheme with R red edges corresponds to a $S - T$ path that intersects R distinct sensors. Looking at the latter half closely, what we really have shown is that the sensor network has resilience at most R! This is the core idea that we will use to reduce EDGE-COLOUR to ULS-RES. The reduction itself will be much more complicated. In particular, there are two main issues that we need to deal with.

First, the EDGE-COLOUR problem is a graph problem while ULS-RES is a geometry problem. Thus, we need a way to impose geometry onto an instance of EDGE-COLOUR. This can be done in a similar fashion to what we have just presented. However, we need to ensure that in our reduced instance, every line segment sensors need the same length. This is clearly not the case for the current example. This issue will be addressed in Section 2.3.

Second, we need to construct the environment of obstacles in such a way that it imposes the "magical" property on every bi-vertex. Previously, we simply introduced the constraints as part of the problem definition. However, this condition is not part of the definition of the ULS-RES problem. This will be covered in depth in Section 2.4.

2.3 Reduction to ULS-RES- Part 1

In this section, we discuss how to deal with the first issue and defer the second issue to Section 2.4. We will also not worry about how to place S and T since they are intricately connected to the second issue. For now, we simply assume that we may use bi-vertices freely in the reduction.

2.3.1 Gadget - Bipartite Chain

We first introduce a gadget that we will use frequently in the reduction - a *bipartite chain*. The fundamental idea of a bipartite chain is based on the four-chain - it is a construct containing $2k + 1$ bi-vertices and $2k$ unit line segments sensors connecting the nodes in a chain fashion, for some positive integer $k > 1$. Similar to a four-chain, every bi-vertex (except the two end bi-vertices) has two incident sensors, one to each half. If we look at Figure 2.3 and pretend that the line segments are all unit line segment sensors, then the figure presents two examples of bipartite chains with $k = 2$.

Unlike the definition of four-chain, bipartite chains have a geometric limitation - two consecutive bi-vertices in the chain are connected by unit line segment sensors. Previously, we treated four-chains as a graph and so there was no geometry involved with edges in the chain. Another difference is that the number of edges/sensors in the chain is no longer limited to a fixed number. Instead, a bipartite chain can have any even number of sensors.

Note that it would not have sufficed to simply impose a geometry onto four-chains. This is because if we limited ourselves to chains of four unit line segments, the overall geometry of our reduction becomes very limited as well. As later sections will show, we need the flexibility to have any number of segments in the same chain (as long as the number is even). We emphasize the importance of having an even number of sensors, for reasons that will become apparent in Section 2.4.4.

Assuming the "magical" property of the bi-vertices, a bipartite chain has the special property that for any two consecutive sensors in the chain and any $S - T$ path, at least one of these two sensors must be intersected by the path. As a direct corollary, at least half of the sensors in every bipartite chain must be intersected by any $S - T$ path.

2.3.2 Reduction

Let $P = (G, R)$, where $G = (V_B, V_T, E)$ is a bipartite four-chain graph, be an instance of EDGE-COLOUR. We refer to bi-vertices V_B and V_T as *bottom bi-vertices* and *top bi-vertices*, respectively. We use this notation because in the reduction, bi-vertices in V_B will be located beneath bi-vertices in V_T. The general idea of the reduction is very similar to the reduction we used to reduce the NODE-COLOUR problem to EDGE-COLOUR problem - we simply replace every four-chain in E by a bipartite chain connecting the same two bi-vertices. The intricacy of the reduction lies in how we place the bi-vertices so that we can connect them using bipartite chains.

First, we introduce a coordinate system. Let us scale the length of each unit line segment sensor to $\sqrt{2}$ (we can re-scale this length back to 1 after we have finished the entire construction). Thus, a unit line segment sensor exactly spans the distance from $(0, 0)$ to $(1, 1)$. We do this since all sensors we introduce will be 45 degrees from the x-axis or the y-axis. The reason for choosing 45 degrees is to pave the way for the second part of the reduction (constructing the "environment") that will be presented in Section 2.4.

The next step is to put the bottom bi-vertices on the coordinate system. We will place these bi-vertices on the $x = 0$ line (the y-axis). The first bottom bi-vertex will be located at $(0, 0)$, the second at $(0, 4)$, and so on. In general, the j^{th} bottom bi-vertex (labelled from 1) is located at $(0, 4(j - 1))$. Thus, the bottom bi-vertices are 4 distance apart. Let us suppose P contains n bottom bi-vertices and m top bi-vertices. Then the "highest" bottom bi-vertex will have coordinate $(0, 4(n - 1))$.

Next, we also place the top bi-vertices on the $x = 0$ line. These bi-vertices will also be distance 4 apart and have the coordinates $(0, 4(n + m))$, $(0, 4(n + m) + 4)$, ..., $(0, 4(n + m) + 4(m - 1))$. Note that the y-coordinates of these bi-vertices are all divisible by 4. Finally, we orient these bi-vertices such that the positive half lies to the right of the line $x = 0$ and the negative half lies to the left.

We will now connect the bottom and the top bi-vertices using bipartite chains. For each four-chain in P, we will place a bipartite chain with the same connectivity. Suppose we are trying to connect a bipartite chain from a bottom bi-vertex at $(0, 4a)$ to a top bi-vertex at $(0, 4b)$. There are four cases to consider:

1. The chain connects from positive half to positive half. This is depicted by

Figure 2.7a. Note that the bend occurs at coordinate $(2(b-a), 2(b+a))$.

2. The chain connects from positive half to negative half. This is depicted by Figure 2.7b. Note that the two bends occur at coordinate $((b-a), (b+3a))$ and $((a-b), (3b+a))$.

3. The chain connects from negative half to negative half. This is depicted by Figure 2.7c. Note that the bend occurs at coordinate $(2(a-b), 2(b+a))$.

4. The chain connects from negative half to positive half. This is depicted by Figure 2.7d. Note that the two bends occur at coordinate $((a-b), (b+3a))$ and $((b-a), (3b+a))$.

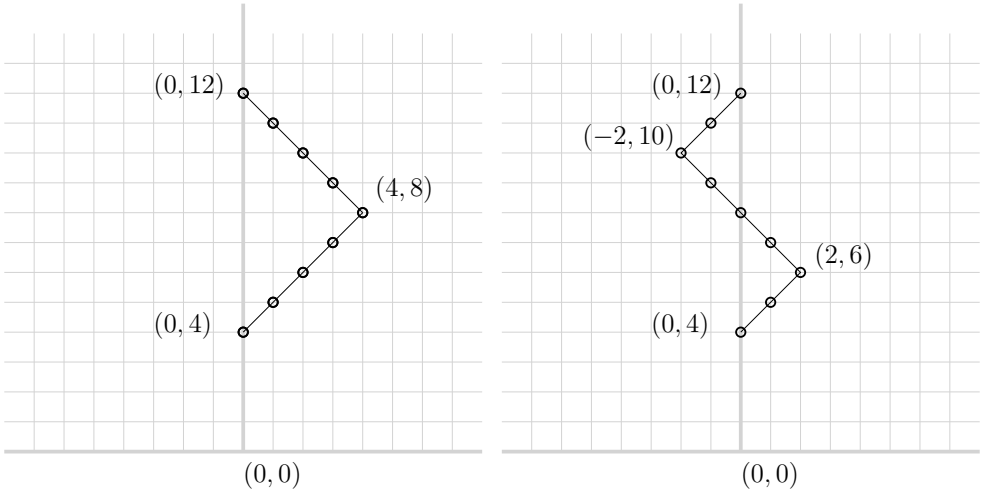
In the figure, the bi-vertices of each bipartite chain are oriented such that the bottom sensor connects to the negative half and the top sensor connects to the positive half.

It is not hard to see that every chain constructed this way must contain an even number of sensors. For every sensor we add that increases the x-coordinate of the chain by 1, there must be a corresponding sensor that decreases the x-coordinate of the chain by 1. Since the difference in x-coordinate between the variable and top bi-vertex is zero, the total length of the chain is also even.

Note that in order for the bends of the bipartite chain to occur at lattice points, we need the difference in the y-coordinate between the top bi-vertex and the bottom bi-vertex to be divisible by 4. This is why we chose to place the bi-vertices such that the y-coordinate is always divisible by 4.

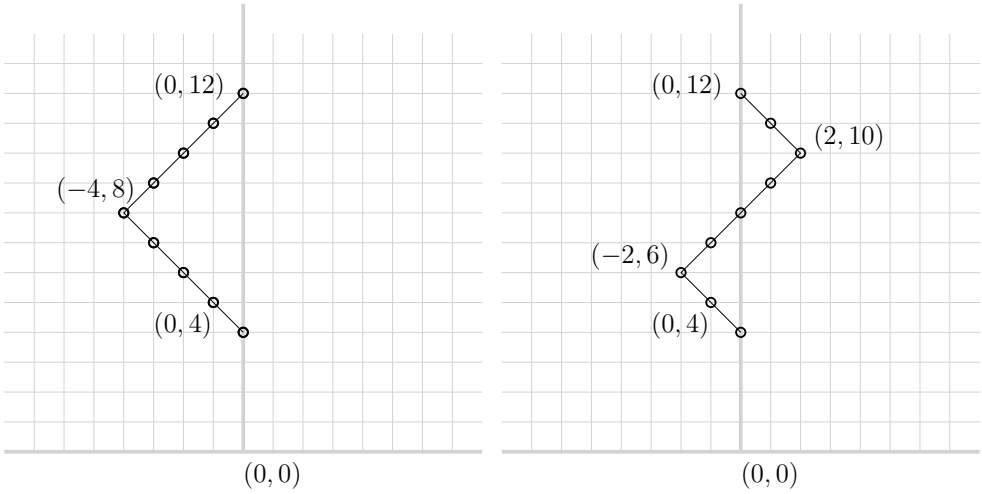
Combining everything together, Figure 2.8 demonstrates the transformation process for the EDGE-COLOUR instance shown in Figure 2.4. Note that in the figure, some of the bipartite chains overlap each other (these are chains that are drawn very close to each other); this is inevitable now since we are placing all line segments at 45 degrees. In Section 3.1.5, we will enhance the reduction we presented so that there will be no overlapping sensors. Thus, we can assume it is possible for an $S-T$ path to only cross some of the sensors but not others even if they reside at the same location in this construction of bipartite chains.

Let us assume that the total number of sensors used to construct the bipartite chains is $2l$. Note that this must be an even number since every bipartite chain consists of an even number of sensors. Finally, we set the value B for the reduced ULS-RES instance to be $l + (R - 2|E|)$.



(a) Positive half to positive half

(b) Positive half to negative half



(c) Negative half to negative half

(d) Negative half to positive half

Figure 2.7: Example of connecting bipartite chain from a bottom bi-vertex at $(4a, 0)$ to a top bi-vertex at $(4b, 0)$ with $a = 1$ and $b = 2$.

2.4 Reduction to ULS-RES- Part 2

In Section 2.3, we showed how to construct a sensor network from a EDGE-COLOUR problem. We now deal with the second issue of the reduction - how to impose the "magical" property on every bi-vertex. The key idea is to add "sensor obstacles" to the sensor network (constructed in the previous section) in such a way that every $S - T$ path is forced to obey the constraints on the bi-vertices (Property 2.1). In a sense, what we will construct in this section is the "environment" of obstacles that the sensor network resides in.

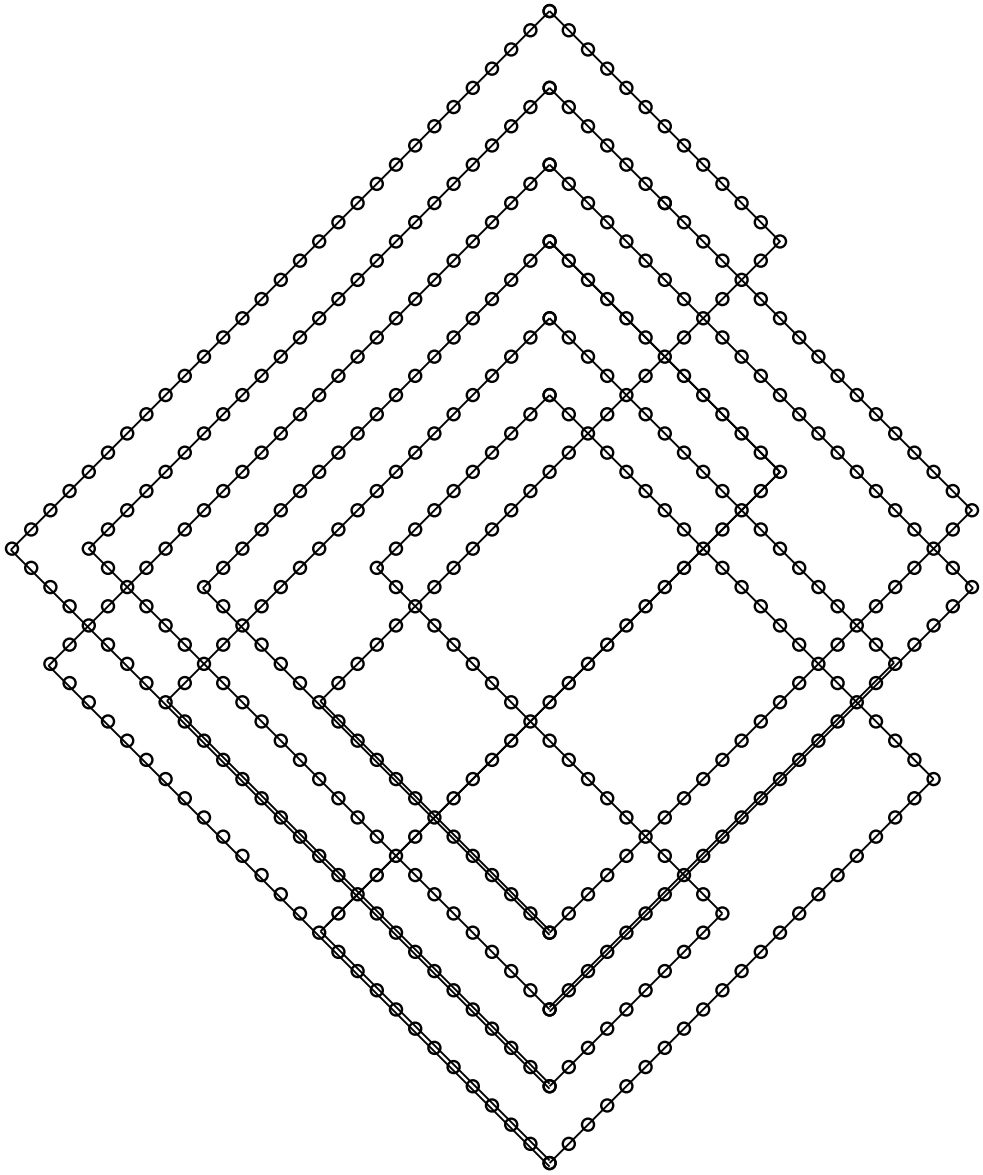


Figure 2.8: Partial reduction from EDGE-COLOUR to ULS-RES

2.4.1 Gadget - Fences

A *fence* is a collection of sensors overlaid on top of each other at the same place. The key property is that the number of sensors in a fence must be large enough so that any path which crosses the fence can not possibly be the minimal resilience path. For example, if we know the resilience of the sensor network is in the range of 10 to 20, we can construct a fence by placing 30 sensors in the same position. Any path that crosses the fence will have resilience at least 30 and cannot be the

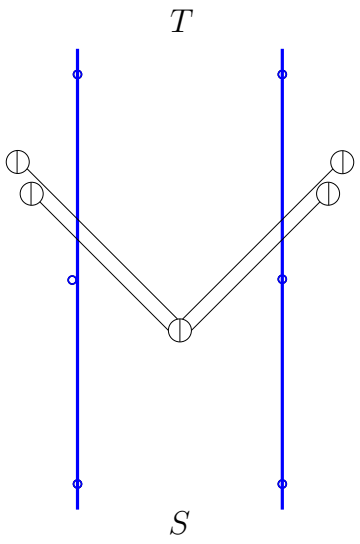


Figure 2.9: Implementing a bi-vertex, the blue bolded sensors represents fences

minimal resilience path. Thus, when searching for the minimal resilience path, we do not need to consider any paths which cross a fence. From here on, when we refer to an $S - T$ path, we will assume that the path does not intersect any fence.

Essentially, fences are the basic building blocks for creating an environment of "sensor obstacles". By using several fences, we can create channels to steer the minimal resilience path. One sample usage is presented in Figure 2.9. The figure demonstrates how a channel may be used to implement a bi-vertex. Let us suppose that by placing other fences appropriately, any $S - T$ path must traverse the channel. It then follows that any $S - T$ path must choose to intersect either both the right sensors (positively incident sensors) or both the left sensor (negatively incident sensors). Thus, Property 2.1 is satisfied.

2.4.2 Implementing Bipartite Chain

Now that we are able to implement a bi-vertex, the next step is to implement a bipartite chain. We can view a bipartite chain simply as a series of bi-vertices strung together. Thus, to implement a bipartite chain, we first duplicate the construction depicted in Figure 2.9 as many times as there are bi-vertices in the chain. Next, we string these channels side-by-side as shown in Figure 2.10. Again, by using other fences, we may assume that any $S - T$ path must traverse every channel. Using the same argument, the magical property of every bi-vertex in the chain is satisfied.

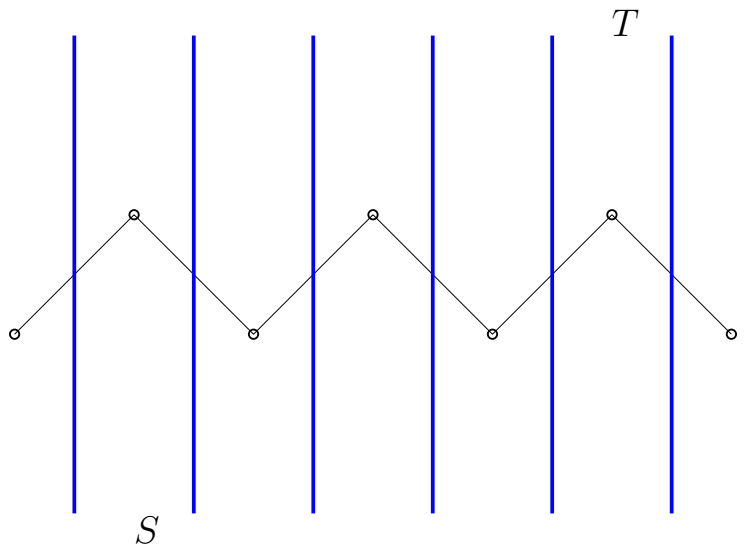


Figure 2.10: Locally implementing a bipartite chain of length 6.

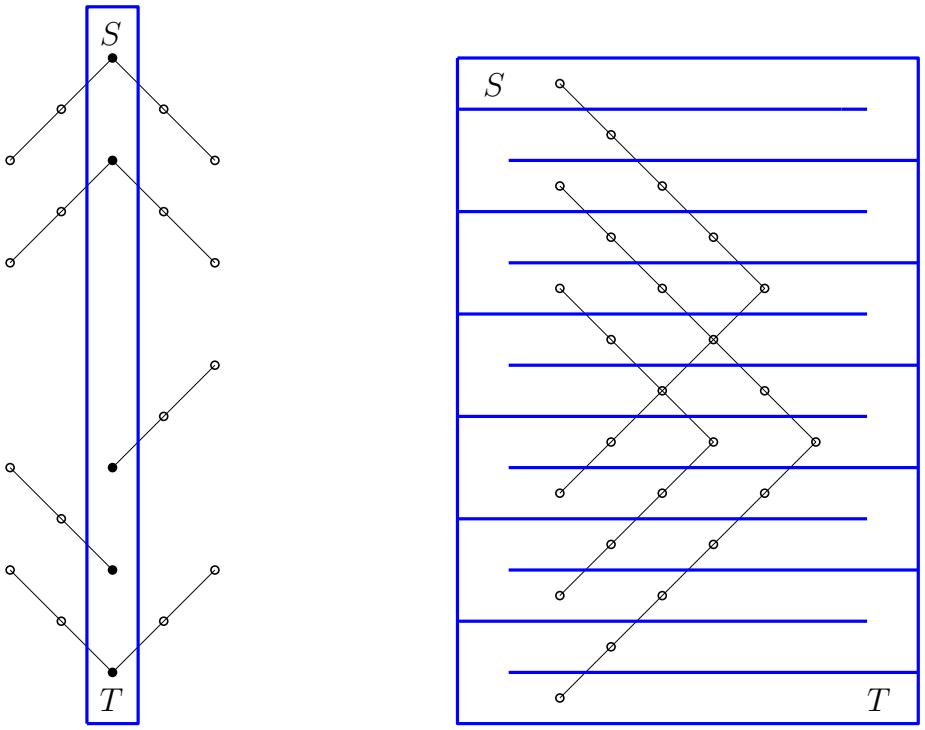
Thus, for every two consecutive sensors in a bipartite chain, at least one must be intersected by any $S - T$ path.

2.4.3 Constructing the Overall Environment

Given a sensor network constructed as per Section 2.3.2, we now show how to construct the environment of obstacles so that all of the bi-vertex constraints are obeyed. The first step is to simultaneously implement all bottom and top bi-vertices. Recall that these bi-vertices are vertically aligned on the y-axis. Thus, to implement them simultaneously, we can vertically extend the channel depicted by Figure 2.9 to create a long vertical channel. This is demonstrated in Figure 2.11a.

Next, we need to implement the individual bipartite chains that connect a bottom bi-vertex to a top bi-vertex. Note that the vertical channel already serves to implement some parts of the fence construction needed for bipartite chains of the form similar to Figure 2.7b and 2.7d. The middle bi-vertex of these chains, which lines up vertically with the bottom and top bi-vertices, is already satisfied by the vertical channel.

Thus, we can split the bipartite chains into two independent parts, one consists of parts of chains lying to the right of the vertical channel and the other of ones lying to the left. Note that it is possible for a single bipartite chain to contain both parts. We now show how to implement the chain lying to the right of the vertical



(a) Implementing all bottom and top bi-vertices (represented by filled circle) simultaneously

(b) Implementing several bipartite chains at once. Only part of each chain is shown

Figure 2.11: Constructing various parts of the environment

channel. Recall that a single bipartite chain can be implemented by the construct shown in Figure 2.10. To implement multiple chains, we need to somehow combine each individual implementation into a single global construct. It was for this reason that in Section 2.3.2, we had aligned all line segments at 45 degrees and put the entire construction on a grid. Since all bipartite chains are aligned equally, we can easily extend the construct to simultaneously satisfies multiple bipartite chains. Figure 2.11b shows how to satisfy three bipartite chains together using an underlying maze. Note that if we strip off the boundary of the maze, the resulting construct is essentially Figure 2.10 (rotated). Finally, we can implement chains lying to the left of the vertical channel similarly.

Putting everything together, what we have is a long vertical channel at the centre encompassing the bottom and top bi-vertices and the middle bi-vertex of some bipartite chains. On either side of the vertical channel, we concatenate a maze composed of horizontal rows in order to implement the bipartite chains. The final step is to add S and T and merge the constructs together so that every $S-T$ path is forced to traverse the vertical channel and both horizontal mazes. This can be achieved as

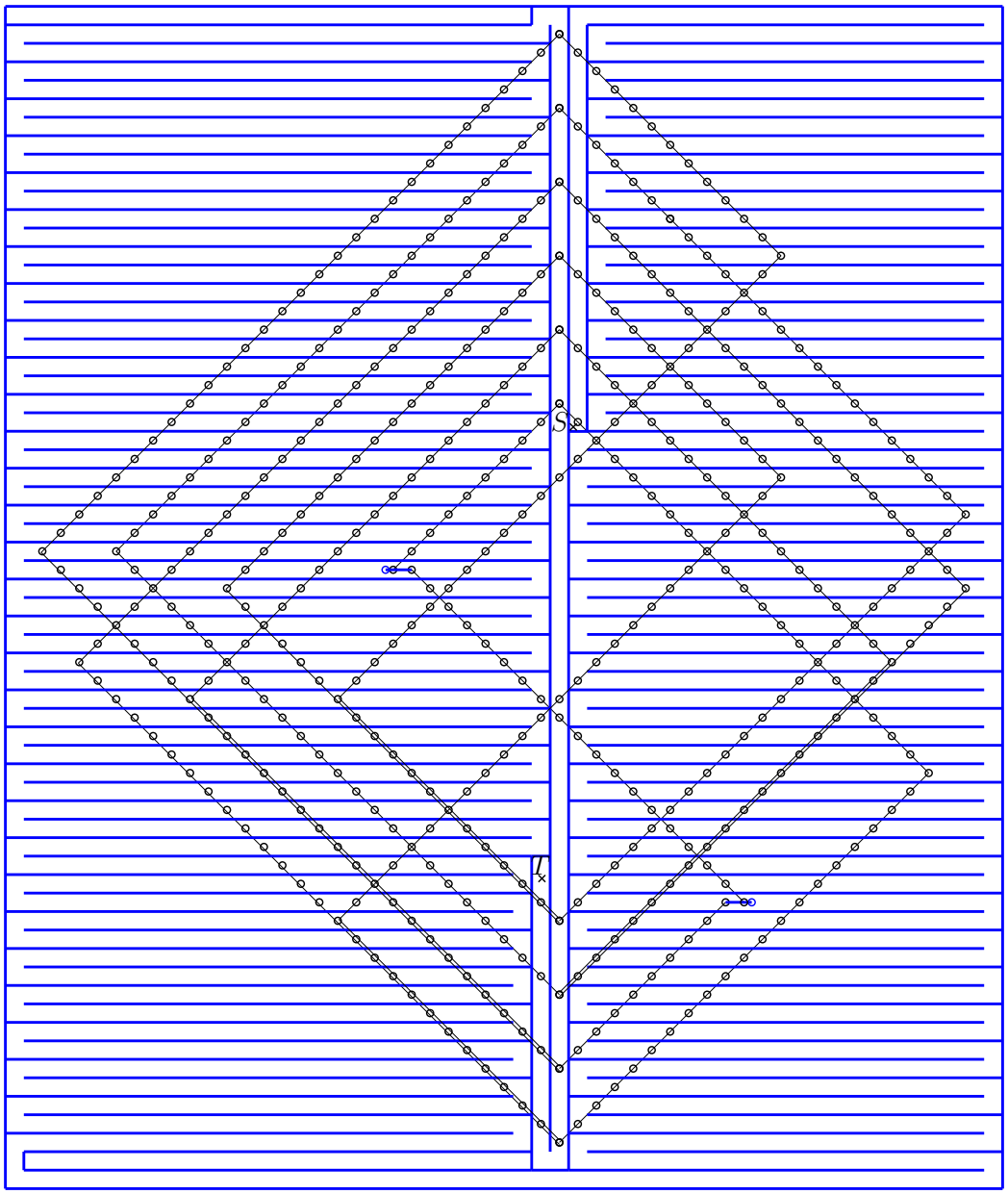


Figure 2.12: Full reduction from EDGE-COLOUR to ULS-RES

shown in Figure 2.12. For clarification, the point S is suppose to lie at the end of the right-most column (marked by an 'x'). The label is located at the central column due to space issues.

On careful inspection, it is evident that the figure is more complicated than the description provided above. For example, there are three vertical channels instead of one, some bipartite chains are shifted, the very top and bottom of the horizontal

mazes are slightly perturbed, etc. However, when combining the several individual parts together, we must take into account the interactions between these parts. These details are introduced to address the interactions and will be addressed in depth in Section 3.1. Finally, we remark that when a bipartite chain is shifted, it still retains the property that for any $S-T$ path and any pair of consecutive sensors in the chain, at least one must be intersected by the path.

2.4.4 Proof of Correctness

Let $P = (G, R)$ be an instance of EDGE-COLOUR problem, where $G = (V_B, V_T, E)$ is a bipartite four-chain graph. Let $P' = (U, \mathcal{A}, S, T, l + R - 2|E|)$ be the reduced ULS-RES instance constructed from P as described in Section 2.3.2 and 2.4.3, where $2l$ is the total number of sensors used to construct the bipartite chains in P'.

Lemma 2.9. *If there exists a satisfying edge colouring scheme for G with R red edges, then the resilience of the sensor network described in P' is at most $l + R - 2|E|$.*

Proof. Recall that if there exists a $S - T$ path in a sensor network which intersects at most $l + R - 2|E|$ distinct sensors, then the resilience of the network is at most $l + R - 2|E|$. Thus, it suffice to show that such a path exists for P'. Consider a $S-T$ path for P' which does not intersect any fences. Then the sensors that the path intersects are those used to construct the bipartite chains. Thus, we can treat such a path as a series of decisions where for each bi-vertex, the path chooses whether to intersect all positively incident sensors or all negatively incident sensors (or both). Furthermore, note that these decisions are independent.

Consider the satisfying colouring scheme for G which colours R red edges. Following a similar argument presented in Lemma 2.7, we can show that $R \geq 2|E|$. Furthermore, the number of four-chains in G for which both end edges are coloured red must be $\leq R - 2|E|$.

We now construct an $S - T$ path based on the colouring scheme. First, the path considers the bottom and top bi-vertices. For each bi-vertex, the path will choose intersect all positively incident sensors if and only if for the corresponding bi-vertex in G, the edge colouring scheme coloured all positively incident edges red. Otherwise, the path will choose to intersect all negatively incident sensors

(this means that the colouring scheme must have coloured all negatively incident edges red). At this point, it must be the case that that for every bipartite chain, the $S - T$ path has already decided whether to intersect the two end sensors or not (since the end sensors of the bipartite chains are exactly the sensors incident on the top and bottom bi-vertices). Furthermore, if an end sensor of a bipartite chain is intersected, then for the corresponding four-chain in G, the end edge of the chain must have been coloured red. Thus, the number of bipartite chains where both ends are chosen to be intersected by the path is $\leq R - 2|E|$.

We note that after the end sensors of each bipartite chain have been decided, these chains are no longer correlated. Thus, we can analyze which sensors the $S-T$ path will choose to intersect for each chain independently. Consider a bipartite chain of length $2p$, laid horizontally. We label each sensors in the chain, starting with 1 on the very left and proceeding to the right until we label the rightmost sensor with $2p$. There are 3 cases to consider:

1. Neither end sensors are intersected by the path: we can intersect exactly p sensors by intersecting sensors $2, 4, \ldots, 2p-2, 2p-1$.

2. Exactly one end sensor is intersected by the path: without loss of generality, assume sensor 1 is intersected and $2p$ is not. We can intersect exactly p sensors by intersecting sensor $3, 5, \ldots 2k-1$.

3. Both end sensors are intersected by the path: we can intersect exactly $p+1$ sensors by intersecting sensors $2, 4, \ldots 2p-2$.

Thus, for each chain, we cross p sensors in the first two cases and $p+1$ sensor in the last case. Thus, by following the above scheme for every bipartite chains, the total number of sensors crossed is l plus one for every occurrence of the last case. However, we have shown above that number of occurrences in the last case is $\leq R - 2|E|$. □

Lemma 2.10. *If the resilience of the sensor network described in P' is at most $l + R - 2|E|$, then there exists a satisfying edge colouring scheme for G with R red edges.*

Proof. If the resilience of the sensor network is at most $l + R - 2|E|$, then there must exists a $S - T$ path which intersects at most $l + R - 2|E|$ distinct sensors.

By definition, this path will not cross any fences and thus will obey the constraint at every bi-vertex. Therefore, for every bipartite chain, at least half of the sensor must be intersected by the path. Following a similar argument presented in Lemma 2.7, we can show that the number of bipartite chains where both end sensors are intersected by the path is $\leq R - 2|E|$.

We now construct a satisfying colouring scheme for G. We begin by colouring the edges incident on the bi-vertices in V_B and V_T (whose corresponding bi-vertices in P' are the bottom and top bi-vertices, respectively). For each bi-vertex, we colour all positively incident edges red if and only if the path intersected all positively incident sensors for the corresponding bi-vertex in P'. Otherwise, we colour all negatively incident edges red (this means that that path must have intersected all negatively incident sensors). At this point, the colouring scheme has coloured the two end edges of each four-chain. Furthermore, the number of four-chain with both end edges coloured red must correspond to a bipartite chain for which the path decided to intersect both end sensors. The number of such four-chains is $\leq R - 2|E|$.

Let us finish the colouring for each four-chain according to the scheme provided in Lemma 2.6. Overall, for each four-chain, we colour two red edges if at least one of the end edges is not coloured red. Otherwise, we colour three red edges. Thus, the total number of red edges is $2|E|$ plus one edge per four-chains with both end edges coloured red. Applying the previous observation yields that the total number of red edges is $\leq 2|E| + (R - 2|E|) = R$. □

Theorem 2.11. *ULS-RES is NP-complete.*

Proof. First, ULS-RES is in NP - a certificate consists of the sensors to remove. We can then check in polynomial time that an $S - T$ path exists without crossing any sensors. From Theorem 2.8, we know EDGE-COLOUR is NP-complete. We will later show in Theorem 3.1 that the reduction presented can be completed in polynomial time. This combined with Lemma 2.9 and 2.10 proves that ULS-RES is NP-hard. □

Chapter 3

Implementation Details and Extension

In the previous sections, we split the reduction into two parts: converting the EDGE-COLOUR instance to use only unit line segments and constructing an underlying maze to realize all bi-vertices. As shown in Figure 2.12, the complete reduction was not a simple combination of these individual parts. This is because we have to take into account interactions between bipartite chains and the underlying maze. Accommodating these interactions results in modifications to the maze or the bipartite chains. Some of these changes are shown in Figure 2.12, such as the addition of two "half-vertical-channels" and shifting of a bipartite chain. There are also other modifications which were not shown so that the figure does not become too cluttered. The focus of the first part of this chapter is to discuss these modifications and the reasoning behind them. Finally, we will show that the overall construction can be realized in polynomial time.

The second half of the chapter shows how the entire reduction from EDGE-COLOUR to ULS-RES may be modified for sensor networks with different types of coverage regions. In general, if the shape of the coverage region is not symmetric, the reduction can be adapted.

Notation-wise, let $P = (G, R)$ be the instance of EDGE-COLOUR where $G = (V_B, V_T, E)$ is a bipartite four-chain graph. Let P' be the reduced ULS-RES instance. We will still refer to the bi-vertices in P' that corresponds to V_B as *bottom bi-vertices* and the bi-vertices that corresponds to V_T as *top bi-vertices*.

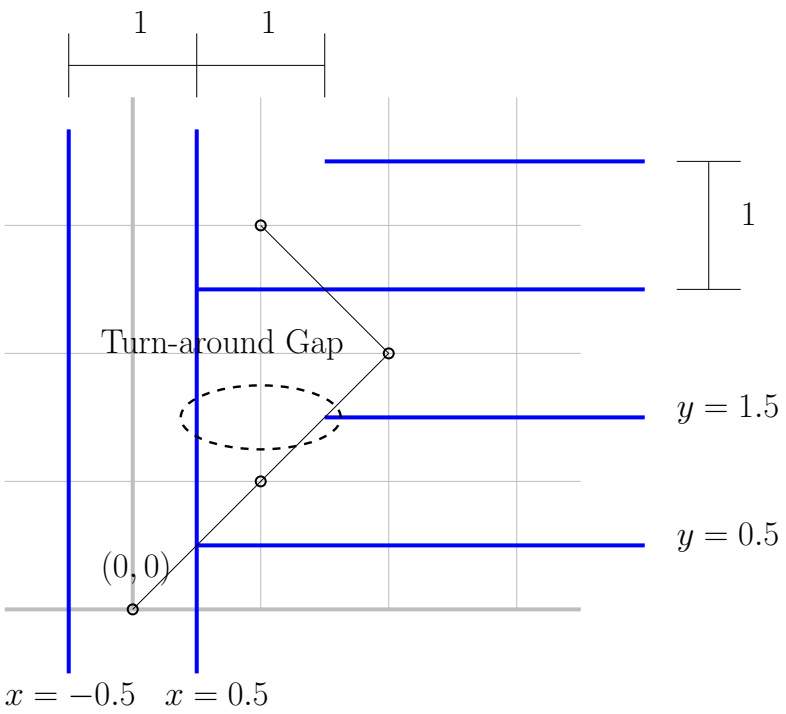


Figure 3.1: Basic specifications of the maze

3.1 Details of Implementation

3.1.1 Preliminary

Recall that in the reduction, we had placed the bottom bi-vertices and the chains on a grid. We scaled the length of each unit line segment sensor to $\sqrt{2}$. Thus, the distance from $(0, 0)$ to $(1, 1)$ is exactly spanned by one unit line segments. We will be using the same coordinate space in this section. This allows us to provide some preliminary specifications of the maze, which are summarized in Figure 3.1.

The channels are chosen to have width 1 and placed so that for each bi-vertex in a bipartite chain, it will lie exactly in the middle of the channel. Also the vertical channel encompassing the bottom and top bi-vertices spans the x-coordinate interval $[-0.5, 0.5]$. In fact, every vertical fence of the maze has *half-integer* x-coordinate and every horizontal fence of the maze has *half-integer* y-coordinate, where a *half-integer* is defined to be a number of the form $n + \frac{1}{2}$ for $n \in \mathbb{Z}$. Finally, the choice of 1 for the turn-around gap was to ensure uniformity; the gap can have any non-zero width.

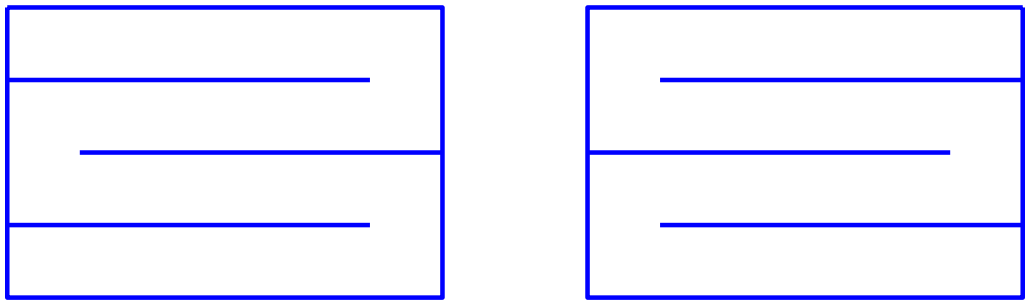


Figure 3.2: Two configurations of a horizontal maze

We note that by re-scaling the length of a unit line segment to span the distance from $(0, 0)$ to $(2, 2)$, we place fences of the maze on integer coordinates. However, the original scaling is more convenient when we discuss the parity of the difference in x and y coordinates in later sections.

3.1.2 Details of the Horizontal Mazes

For any horizontal maze, there are two possible configurations: one which has the topmost turn-around gap located to the left and one located to the right. These two configurations are mirror images and shown in Figure 3.2. For a single horizontal maze, the choice of the configuration is not important due to the inherent symmetry. Here, we make the choice that for the horizontal maze to the right of the central vertical channel, we will choose the configuration as depicted in Figure 2.12. This configuration has the property that at any intersection point between the right horizontal maze and the central vertical channel (not including the two half-vertical-channels), the difference in the x-y coordinates is even. This property will become important in later sections.

The more intricate part is the relative configuration of the two horizontal mazes in the reduction - should they be mirror images or translation? Here, we will choose the left horizontal maze to be a translation of the right horizontal maze (as depicted in Figure 2.12). This has the property that for any intersection point between the left horizontal maze and the central vertical channel, the difference in the x-y coordinate is odd. Again, this has important implications in later sections.

Finally, we make two remarks on the overall structure of the horizontal mazes. First, we added an additional horizontal channel at the very bottom so that it is possible for the $S - T$ path to traverse from the right horizontal maze to the left

horizontal maze. Secondly, in the left horizontal maze, there are two additional closed off gardens located at the very top and very bottom which are not present in the right horizontal maze. These were added to ensure that the $S - T$ path would appropriately traverse both horizontal mazes and return to the central vertical channels.

3.1.3 Bounding the Maze

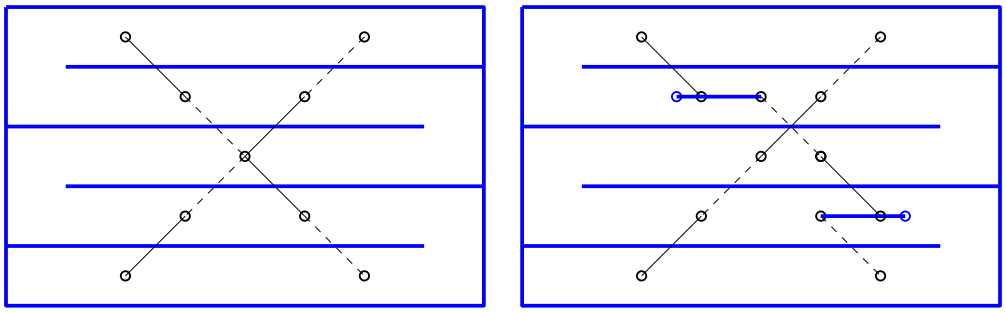
Given the specifications of the channels, we can now discuss how big the overall maze is. Let us suppose the smallest bounding box encompassing all bipartite chains, bottom bi-vertices, and top bi-vertices is defined by the rectangle with lower left coordinate (x_1, y_1) and upper right coordinate (x_2, y_2). By construction, all four variables must be integers. Clearly, this bounding box needs to be strictly contained inside the maze.

Let us suppose the maze is defined by a rectangle with lower left coordinate (x'_1, y'_1) and upper right coordinate (x'_2, y'_2). The strict containment constraint implies that $x'_1 < x_1$, $y'_1 < y1$, $x'_2 > x_2$, and $y'_2 > y_2$. On top of this, we add an additional constraint that y'_1 and y'_2 must be half-integers. This is because the uppermost and lowermost bounding wall also serves as part of the horizontal maze. As noted before, these must be half-integers in order to ensure every bi-vertex is exactly in the middle of a channel. Again, the half-integers may be avoided if we re-scale the length of a unit line segment.

The reduction is free to choose the value x'_1, y'_1, x'_2, y'_2 as long as the above conditions are met. For concreteness, we will choose $x'_1 = x_1 - 1, x'_2 = x_2 + 1$ to reduce the overhead of the number of fences used to construct the reduction. For the y-coordinates, we will follow the example presented in Figure 2.12. Since we need two additional horizontal channels on the bottom and one additional horizontal channel on top, we choose $y'_1 = y_1 - 2.5, y'_2 = y_2 + 1.5$. We make note that by our construction, y_1 will always be 0, so y'_1 will always be -2.5.

3.1.4 Cross-over Gadget

In Figure 2.12, there are several instances of different bipartite chains intersecting each other. Recall that the proof of the reduction required the independence of bipartite chains. However, as drawn in Figure 2.12, the bipartite chains will inter-



(a) (b)

Figure 3.3: Dashed line segments are sensors intersected by the $S - T$ path

fere with each other. In order to traverse the maze, the choice at one chain will affect the other. For example, Figure 3.3a illustrates an invalid configuration of sensors to intersect - the middle row is still blocked.

Referring to Figure 3.3a, suppose a path is traversing the middle row from right to left. Then the path cannot choose to intersect the second sensor (from top) of the top-left-to-bottom-right chain without intersecting the second sensor (from top) of the top-right-to-bottom-left chain. Thus, it is impossible to let the path traverse the maze while maintaining the independence of the chains.

To fix this, we replace each intersection with the cross-over gadget depicted Figure 3.3b. The idea is to use a fence to offset one of the chains so that the intersection between the two chains does not occur at a bi-vertex. Instead, we let the chains intersect at the middle of two sensors. An $S - T$ path may now intersect any sensor of one chain without being constrained to intersect a sensor from the other chain. Note that even though part of the bipartite chain is physically disconnected, the added fences ensure that among any pair of sensors, at least one must be crossed by any $S - T$ path.

3.1.5 Overlapping Sensors

In Figure 2.12, some sensors in the bipartite chains overlap each other. This is problematic since overlapping sensors are correlated - a path must either intersect all sensors or none of them. This breaks the assumption that bipartite chains are independent of each other (except for the end sensors). We now show how to modify the reduction to make sure no bipartite chains overlap.

There are two potential scenarios where overlapping sensors can occur. The

first scenarios occurs at the bottom bi-vertices. When multiple bipartite chains are incident on the same half of a bottom bi-vertex, the chains will overlap at the end connected to the bottom bi-vertex. For the same reason, overlapping can occur at the top bi-vertices as well. The second scenario of overlapping sensors occurs between two bipartite chains that cross the central vertical channel (Figure 2.7b and 2.7d). Recall that if a bipartite chain crosses the central vertical channel, the y-coordinate of the chain at the y-axis is equal to the average y-coordinate of the bottom and top bi-vertex that the chain connects. If there are two such chains with the same average y-coordinate, then these chains will overlap in the middle.

To fix the first case, we need to restrict at most one chain to be incident on each half of the bottom bi-vertex. One way to achieve this is by duplicating the bi-vertices. Let m be the number of top bi-vertices (labelled from 1 to m). We will duplicate each bottom bi-vertex $2m$ times (labelled from 1 to $2m$). We then distribute the bipartite chains incident on the original bottom bi-vertex as follows. If the chain connects to the positive half of the j^{th} top bi-vertex, it will be connected to the $(2j-1)^{th}$ copy of the duplicated bottom bi-vertex. If the chain connects to the negative half, the chain will connect to the $(2j)^{th}$ copy of the duplicated bi-vertex.

Since the duplicated bottom bi-vertices represent the same bottom bi-vertex, we need to make sure that for any $S-T$ path, either all positively incident sensors or all negatively incident sensors are intersected. In other words, we want to avoid the possibility that an $S-T$ path decides not to intersect the positively incident sensors on some duplicated bi-vertices and not to intersect the negatively incident sensors on other duplicated bi-vertices. Symbolically, this means that for the original bottom bi-vertex, neither all positively incident sensors nor all negatively incident sensors are intersected. This violates the constraints on the bi-vertex and is undesirable. Thus, we add an additional vertical fence through the duplicated bottom bi-vertices to prevent this from happening. This solution is illustrated in Figure 3.4. It should be noted that some of the duplicated bi-vertices have no incident chain. This occurs whenever there is a top bi-vertex that is not connected to the original bottom bi-vertex.

To handle overlap at the top bi-vertices, we can employ the same technique. Suppose there are n bottom bi-vertices (labelled from 1 to n). We then duplicate each top bi-vertex $2n$ times and distribute the bipartite chains similarly. By duplicating both top and bottom bi-vertices, no two chains will be incident on the same

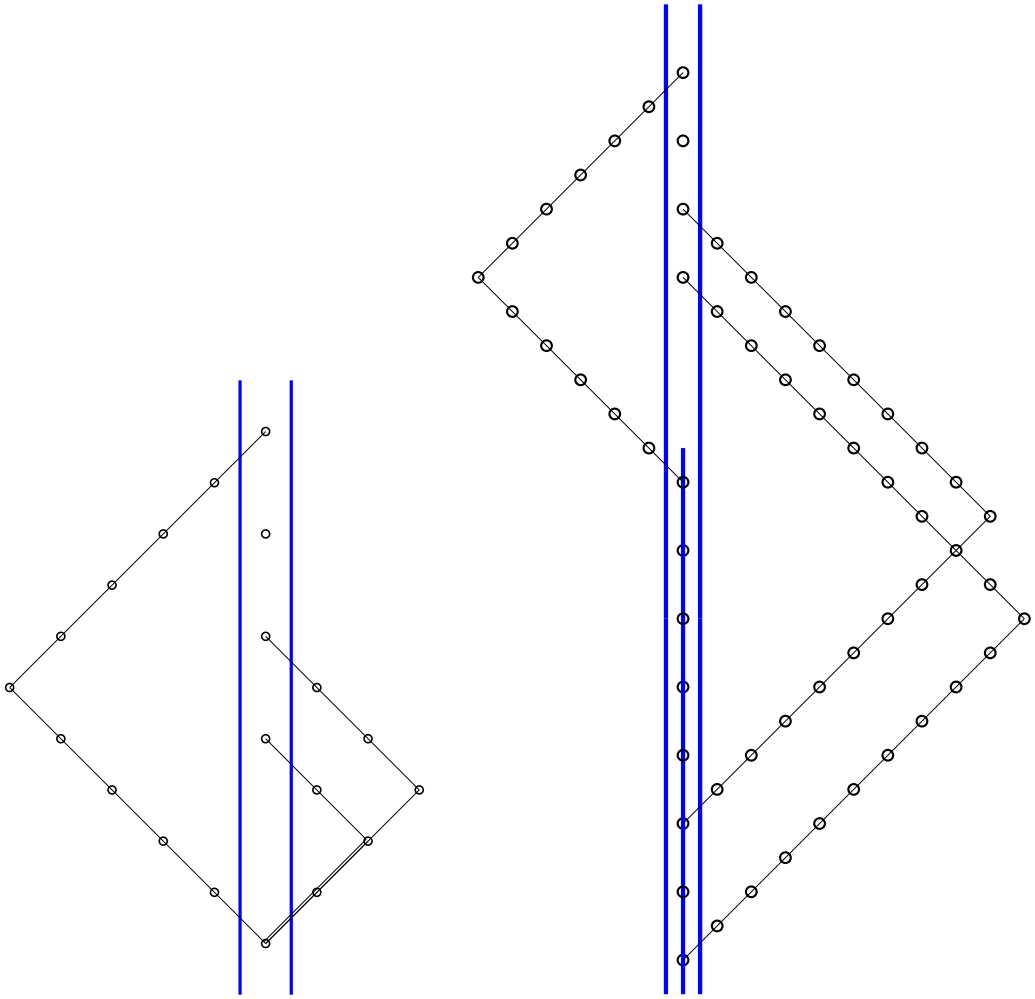


Figure 3.4: Duplicating bottom bi-vertex to avoid overlap

half of the same bi-vertex (unless multiple chains connect the same two bi-vertices, however, we assume that the graph contains no multi-edges).

Formally, we will put the j^{th} copy of the k^{th} bottom bi-vertex at coordinate $(0, 8m(k-1)+4(j-1))$, where $1 \leq j \leq 2m, 1 \leq k \leq n$. The j^{th} copy of the k^{th} top bi-vertex will be placed at coordinate $(0, 16mn + 8n(k-1) + 4(j-1)$, where $1 \leq j \leq 2n, 1 \leq k \leq m$. We have placed the top and bottom bi-vertices 4 distance apart like before. Note that there is a gap between the "highest" bottom bi-vertex (located at $(0, 8mn-4)$) and the "lowest" top bi-vertex (located at $(0, 16mn)$) - this is important for fixing the second scenario of overlapping sensors.

To fix the second scenario of overlapping, we need to rework bipartite chains which cross the central vertical channel. Instead of having the chain cross the cen-

tral vertical channel at the average y-coordinate of the bottom and top bi-vertex that are connected, we assign a different y-coordinate for each possible bottom-top bi-vertex pair. The key idea is that if each chain crosses the central vertical channel at a unique y-coordinate, then the chains will not overlap.

Note that by changing the y-coordinate of where the chain crosses the central vertical channel, the two bend points of the chain will be modified as well. The y-coordinate of the lower (respectively, upper) bend point will be the average of the y-coordinate of the bottom (respectively, top) bi-vertex and the y-coordinate where the chain crosses the central vertical channel.

For a chain that connects the j_b^{th} copy of the k_b^{th} bottom bi-vertex to the j_t^{th} copy of the k_t^{th} top bi-vertex, the bipartite chain will cross the central vertical channel at coordinate $(0, 8mn+8m(k_b-1)+4(j_b-1))$. It may seem odd that the crossing point is not affected by j_t or k_t. However, by construction, j_t and k_t can be determined from k_b and j_b - j_t is either $2k_b-1$ or $2k_b$ (depending on whether the chain connects to the positive half or the negative half of the bottom bi-vertex) and k_t is $\lceil j_b/2 \rceil$. Since $1 \leq j_b \leq 2m$ and $1 \leq k_b \leq n$, it can be verified that every chain will cross the y-axis at a unique coordinate. Furthermore, these designated coordinate range from $(0, 8mn)$ to $(0, 16mn - 4)$, which falls nicely inside the gap between the highest bottom bi-vertex and the lowest top bi-vertex.

As an example, consider a chain that originally connects the positive half of the second bottom bi-vertex to the negative half of the third top bi-vertex. The chain will connect sixth copy of the second bottom bi-vertex (located at $(0, 8m + 20)$) to the third copy of the third top bi-vertex (located at $(0, 16mn+16n+8)$). The chain's shape will still look like Figure 2.7b, except the coordinate at the central vertical channel will be at $(0, 8mn + 8m + 20)$ with the two bends occurring at coordinate $(4mn, 4mn + 8m + 20)$ and $(-4mn + 4m - 8n + 6, 12mn + 4m + 8n + 14)$.

3.1.6 Crossing Between Vertical and Horizontal Channels

When a bipartite chain crosses over from a horizontal maze to a central vertical channel (or vice versa), attention must be paid to the underlying maze. There are two types of crossings that can occur, which we denote by type 1 and type 2. These are depicted by Figure 3.5. Type 1 crossing occurs when the crossing point of the bipartite chain coincides with the intersection point between the horizontal maze

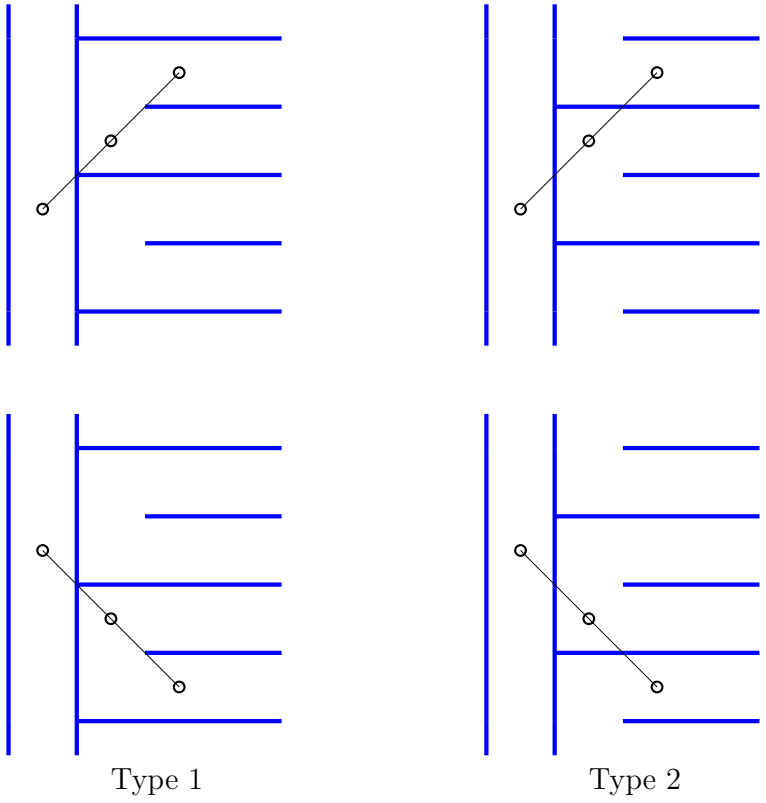


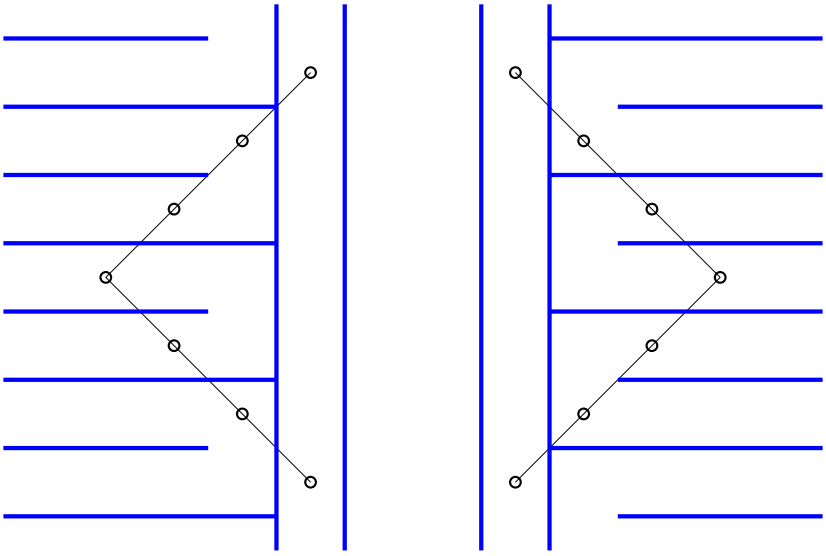
Figure 3.5: Two types of crossing from vertical channel to horizontal channel

and the vertical channel. Type 2 occurs when the crossing point is located halfway between the intersection points.

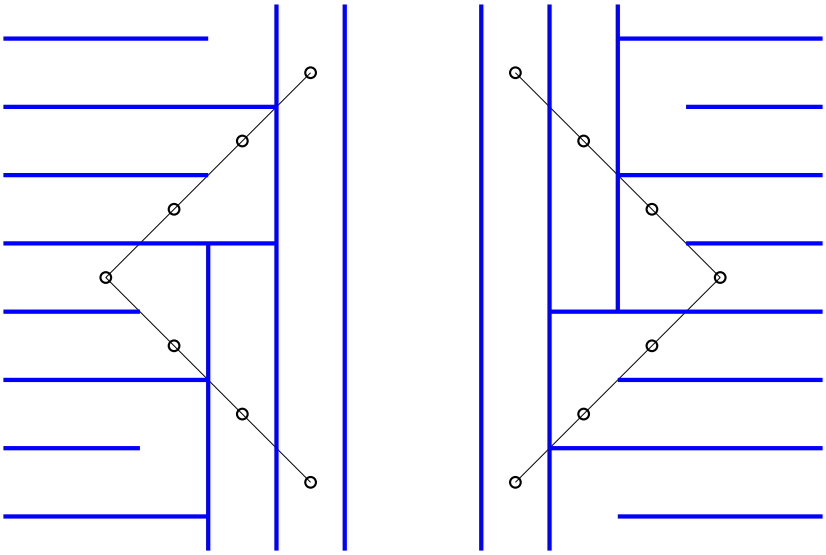
It is not hard to see that type 2 crossings are undesirable since it is possible for a $S - T$ path to intersect neither of the sensors connected to the bi-vertex. Thus, we must ensure that in the reduction, all crossings are type 1 crossings. This is the reason for introducing two additional vertical channels and shifting bipartite chains. We will explore this in more detail in later sections.

3.1.7 Number of Vertical Channels

Note that in Figure 2.12, there were three vertical channels instead of one. The two half-vertical-channel are added so that whenever a bipartite chain crosses over from the vertical channel to the horizontal maze or vice versa, we would have a type 1 crossing. First, let us demonstrate why a single vertical channel is not sufficient. Consider a simple bipartite chain that connects the bottom bi-vertex to the top bi-vertex without crossing over the central vertical channel. Regardless of whether the



(a) Why one central vertical channel is insufficient



(b) Additional vertical channels are used to ensure type 1 crossing

Figure 3.6: Example illustrating the needs of multiple vertical channels

chain is to the left or to the right of the central vertical channel, we will always have one type 2 crossing (see Figure 3.6a).

We can explain this phenomenon using parity. Let us consider the coordinate of the crossing points of the bipartite chain and take the difference between the x-y coordinate. For the crossing point near the bottom bi-vertex, the difference is even since the bi-vertex itself has even difference and the slope of the chain is 1. For the crossing point near the top bi-vertex, the difference in x-y coordinate is

odd because the top bi-vertex has even difference and the chain has slope -1. Now recall in Section 3.1.2, the x-y coordinate difference of the intersection point of the horizontal maze and the vertical channel are either all even (for the right side) or all odd (for the left side). It then follows that exactly one crossing cannot lie on the intersection point of the horizontal maze and the central vertical channel. Consequently, this must be a type 2 crossing.

To fix this, we add the two half-vertical-channels appropriately. Note that the parity of the intersection point between the horizontal maze and the newly added vertical channel will always be different from the parity of the intersection point between the horizontal maze and the central vertical channel. For the right side, we add the vertical channel at the top to ensure the correct parity. For the left side, we add the vertical channel at the bottom. This solution is illustrated in Figure 3.6b. These added half-vertical-channels only need to be long enough to ensure type 1 crossing for all bipartite chain. This means that for the middle portion of the maze, there is still only one central vertical channel as depicted in Figure 2.12.

3.1.8 Shifting of Bipartite Chain

In Figure 2.12, we shifted the bipartite chain that connected from the positive half of a bottom bi-vertex to the negative half of the top bi-vertex. The reason is that without shifting, we will incur type 2 crossings when the chain crosses the central vertical channel in the middle (Figure 3.7a). However, by shifting the chain before it crosses the central vertical channel, the chain will make type 1 crossings instead since shifting changes the parity of the crossing point (Figure 3.7b). Finally, we need to offset the shift after the chain crosses the central vertical channel in order to ensure to avoid type 2 crossing near the top bi-vertex.

It is important to use fences when we shift the chain. This is to ensure that any $S - T$ path still must choose one of the two sensors to cross. Thus, even though the bipartite chain is physically disconnected, it still retains the same desirable property and does not break the reduction. Finally, note that we do not need to shift chains that connect from the negative half of a vertex bi-vertex to the positive half of a top bi-vertex. Due to the additions of vertical channels from the previous section, the parity matches up and the chain will make type 1 crossing without any shifts.

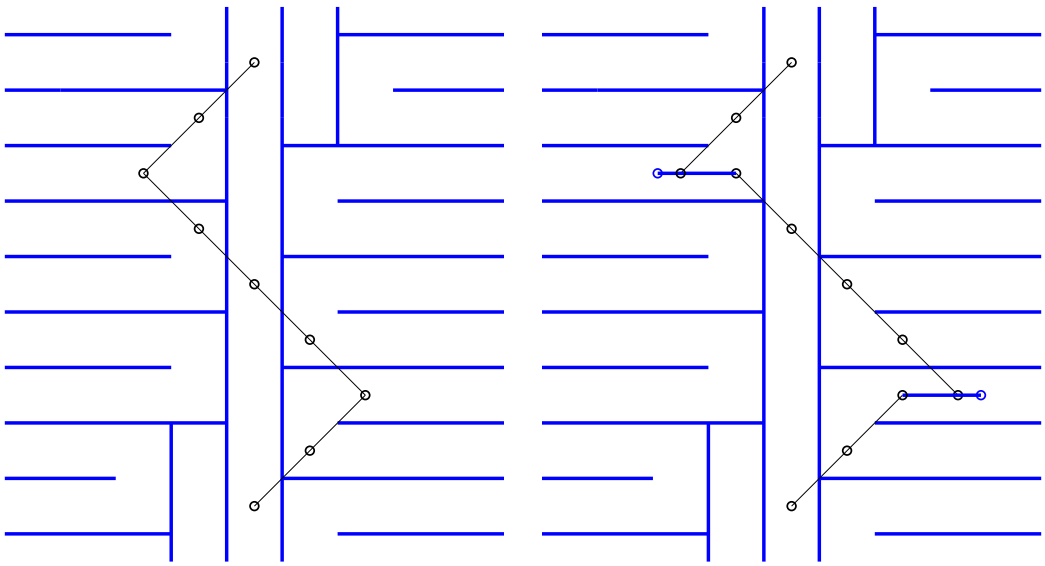


Figure 3.7: Example illustrating why we need to shift bipartite chains

3.1.9 Overall Complexity of Reduction

Theorem 3.1. *Let $P = (G, R)$ be an instance of EDGE-COLOUR, where $G = (V_B, V_T, E)$ is a bipartite four-chain graph. The reduction to an instance of ULS-RES P' is constructible in polynomial time.*

Proof. It suffices to show that the total number of sensors used is polynomial. For brevity, let $|V_B| = n$, $|V_T| = m$, and $|E| = k$. Based on the construction, the highest y-coordinate of a top bi-vertex is located at $(0, 24mn - 4)$. Thus, we can bound the height of the maze to be $O(mn)$. Note that the width of each horizontal maze is proportional to the height, so we can bound the overall width of the entire maze to be $O(mn)$ as well.

Let us now consider the total number of sensors used in bipartite chains. For each bipartite chain, the number of sensors is proportional to the difference in y-coordinate of the top and bottom bi-vertex, which is bounded by $O(mn)$ (the height of the maze). Since there are k chains in total, there are $O(kmn)$ sensors used in building bipartite chains.

We now consider how many fences are required to build the construction. Fences are used in two ways: building the underlying maze and shifting bipartite chains. For the underlying maze, the vertical channels can be constructed with $O(mn)$

fences. For the horizontal mazes, note that the number of horizontal channels is $O(mn)$ and for each horizontal channel, we need $O(mn)$ fences. Thus, we require $O(m^2n^2)$ fences for the maze.

Shifting of the bipartite chains can occur under two circumstances: when a cross-over occurs or a bipartite chain connects the positive half of the bottom bi-vertex to the negative half of the top bi-vertex. For both cases, the number of additional fences required is $O(1)$. Since there are at most constant number of cross-overs per pair of bipartite chain, we can bound the total number of cross-overs by $O(k^2)$. The number of bipartite chain which connects positive half of the bottom bi-vertex to the negative half of the top bi-vertex is bounded by $O(k)$. Overall, the number of fences required for shifting is $O(k^2)$.

Finally, we consider how many actual sensors need to be overlaid in a fence. We need the number of sensors in a fence to be larger than the total number of sensors used to construct the bipartite chain. This can be achieved by having $O(km^2n^2)$ sensors per fence. Combining each individual parts together, the overall reduction can be achieved in $O(km^4n^4)$. □

3.2 Extension to other types of sensor

In this section, we extend the results to other types of sensors. The core idea of the reduction from EDGE-COLOUR to ULS-RES is to first construct a sensor network consisting of bipartite chains connecting appropriate bi-vertices. Then, the network is embedded within a maze of sensor obstacles in order to implement the constraint on each bi-vertex. In both of these steps, the shape of the sensor is not crucial. It is not hard to envision using sensors with other type of shapes to implement the reduction. The key parts that need to be handled are how to implement a bipartite chain and how to construct a cross-over gadget.

3.2.1 Implementing Bipartite Chains

The most important part of the reduction is implementing a bipartite chain. Figure 3.8 shows how one can approach the implementation in the case of ellipse and square sensors. Note that in both examples, consecutive sensors in the chain are not physically connected. This is because the channels of the maze are too wide.

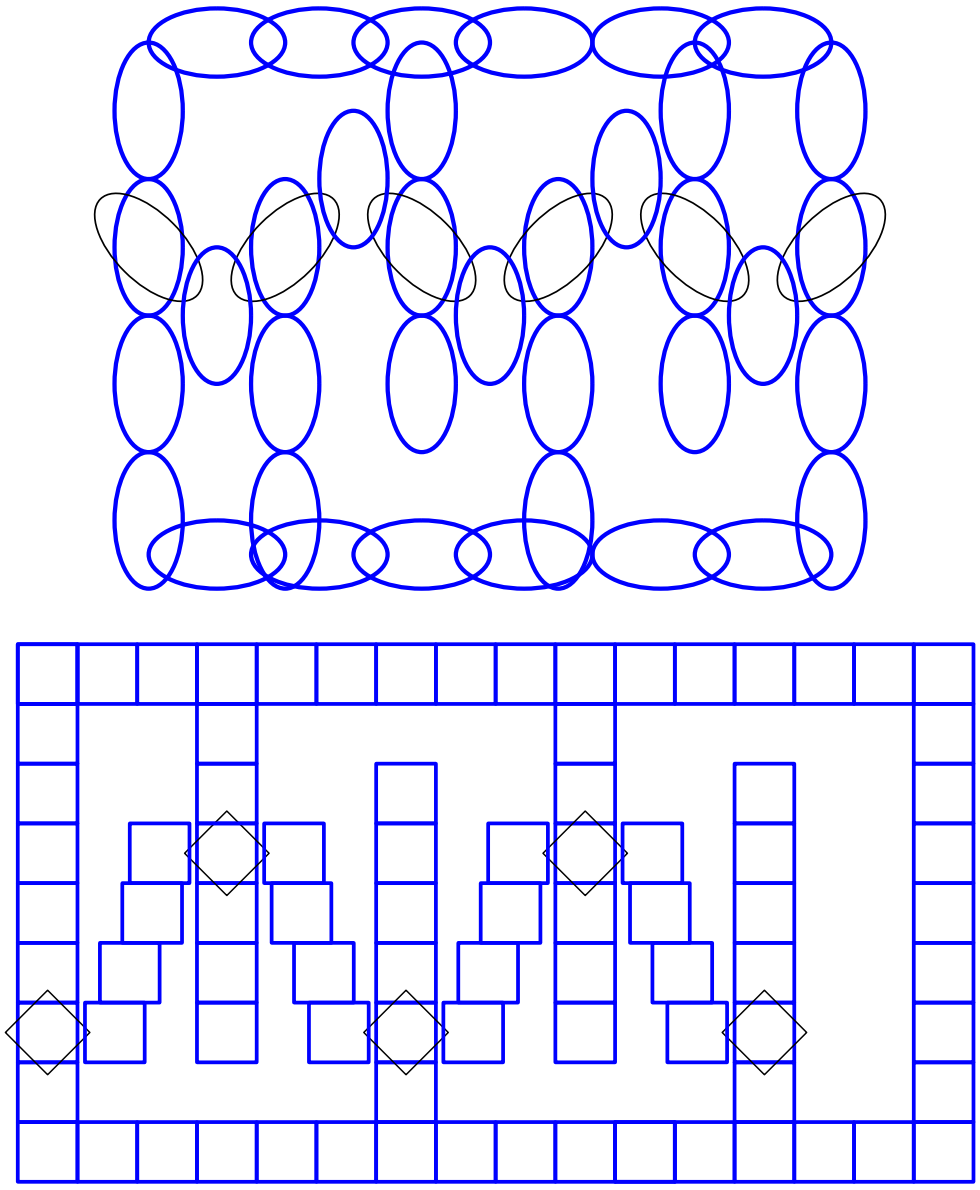


Figure 3.8: Implementing bipartite chains with other sensor shapes

Instead, we add additional fences to connect the two consecutive sensors. Since a $S - T$ path will not intersect the fence, the result is that one of the two consecutive sensors must still be intersected. With this implementation, it is also easy to shift a bipartite chain if the need arises (Section 3.1.8). To do so, we simply shift the location of the sensors and add more fences to bridge the gap.

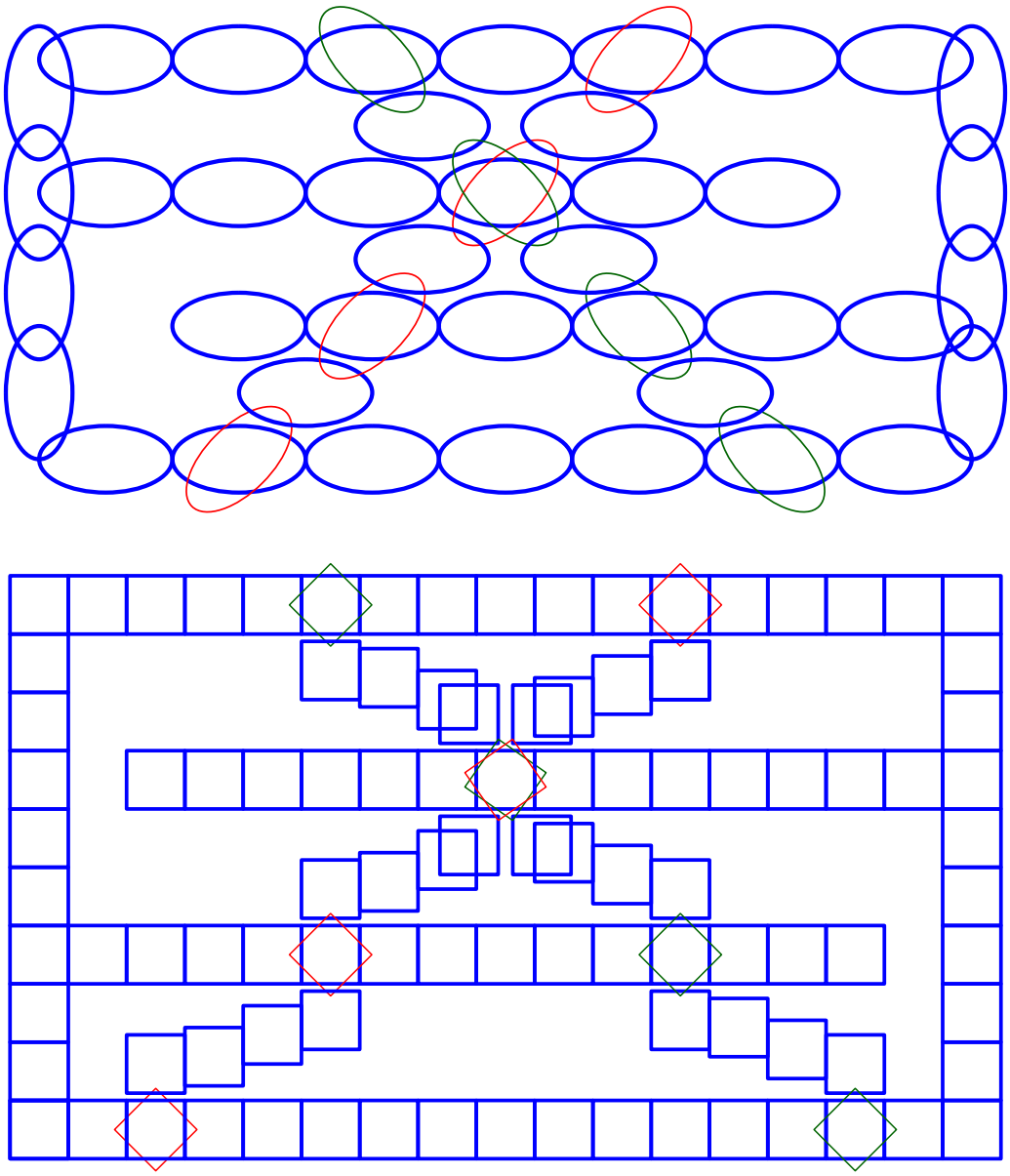


Figure 3.9: Implementing cross-overs with other sensor shapes

3.2.2 Implementing Cross-Overs

To complete the reduction, another requirement is that we can implement a cross-over gadget. The original cross-over gadget had the important property that any $S - T$ path may intersect any sensor of one chain without being constrained to intersecting a sensor of the other chain. This property can be duplicated for other shapes of sensors as well. Figure 3.9 illustrates the cross-over gadget for ellipse and square sensors.

Note that for the square sensors, we needed to slightly rotate the square sensors at the cross-over point. The result is that one square sensor "pokes" out of the square sensor from the other chain. The poked out regions are what allow us to maintain the independence of the two chains. In general, for a non-symmetric shape, we can use the same rotation trick to construct a cross-over gadget.

3.2.3 Limitations

Let us consider the case where the shape of the sensors are disks, which are perfectly symmetrical. The above rotation trick will not work to construct a cross-over gadget. If two disk sensors are overlaid on top of each other, no matter how much we rotate the sensor, we cannot achieve the "poking out" effect that we had in the square sensors case. Without the poked out regions, intersecting a sensor in one chain automatically forces the path to intersect a sensor from the other chain. Thus, the same technique fails to produce a cross-over gadget. Even if we allow disks to have different radius, we still cannot simulate the "poking out" effect. Thus, the reduction presented here cannot be extended to the case of disk sensors.

Chapter 4

Complexity of Computing the Resilience of Sphere Sensor Networks in 3D

Previously, we have shown that determining the resilience of a unit line segment sensor network is NP-hard. Furthermore, the reduction may be extended to sensors whose coverage area is 2D shapes such as squares or ellipses. Unfortunately, the extension breaks down for unit disk sensors due to the inability to construct a cross-over gadget.

It is natural to contemplate whether we can prove the hardness if we further relax the problem by adding the third dimension. The 3D extension of disk sensors is spherical sensors. Previously, we had restricted the domain of the sensor network to the 2D Cartesian plane. Now, we relax the domain to the entire 3D space. The question we investigate is then: *"given a sensor network consisting of spherical sensors in 3D space and two points S and T, what is the resilience of the sensor network?"*

Resilience of Sphere Sensor Network (USPHERE-RES)
Instance: $(U, \mathcal{A}, S, T, B)$, where U is the set of sphere sensors, $\mathcal{A} : U \rightarrow \mathbb{R}^3$ is an arrangement of U in 3D, S and T are two points in 3D, and B is a positive integer.
Question: Is it possible to traverse from S to T without intersecting any sensors by removing at most B sensors from U?

In this chapter, we will show that USPHERE-RES is NP-complete. Conceptually, the third dimension provides an alternative mean to construct a cross-over gadget. We can weave one bipartite chain beneath the other at the cross-over point. Thus, we can adapt the previous reduction to show the hardness. However, this approach is complicated and contain many subtleties (see Appendix A).

Instead, we will provide an alternative reduction from the vertex cover problem. Given a graph $G = (V, E)$, a vertex cover of G is a set of vertices $V' \subseteq V$ such that for every edge in E, at least one endpoint of the edge is in V'.

Vertex Cover (VC)
Instance: $\{G, K\}$, where $G = (V, E)$ is a graph and K is a positive integer.
Question: Does there exist a vertex cover of G containing at most K vertices?

4.1 Reduction from Vertex Cover

We re-use the concept of *fences* - a group of overlapping sensors which the minimal resilience path will not intersect. Similar to the previous reduction, we use fences to create an environment of obstacles. These obstacles are placed such that the the minimal resilience path is forced to make a series of choices which correspond to deciding the vertices in the minimum vertex cover.

4.1.1 Gadget - Tube

A tube is the main gadget that we will use to restrict the minimal resilience path. It is the generalization of the maze channels that we used in the previous reduction. First, note that we can approximate a cylinder using spherical sensors in a similar fashion of using circles to approximate a rectangle (Figure 4.1). Thus, we can use cylinders as a basic construct in our sensor network. We will further assume that each spherical sensor used to construct the cylinder is actually a fence. Thus, any $S - T$ path which intersects a cylinder cannot be the minimal resilience path. Note that in the rightmost example, the approximated rectangle can bend. This is true for approximated cylinders as well.

A *tube* is made up of several cylinders connected to each other and forming a closed loop. In other words, if we consider each cylinder as a node and add an edge

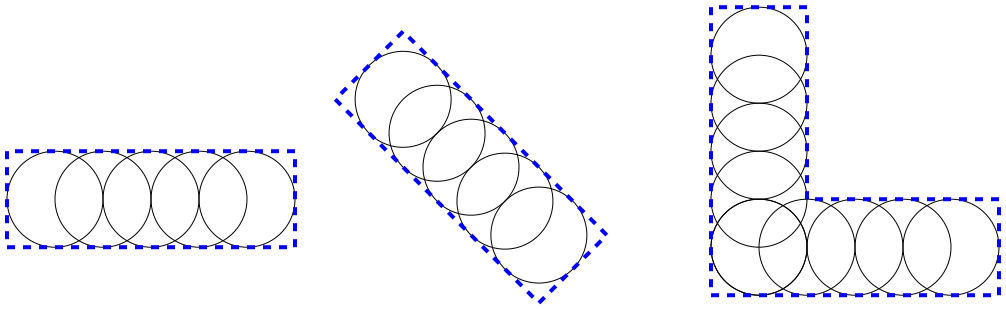


Figure 4.1: Approximating rectangles with circles

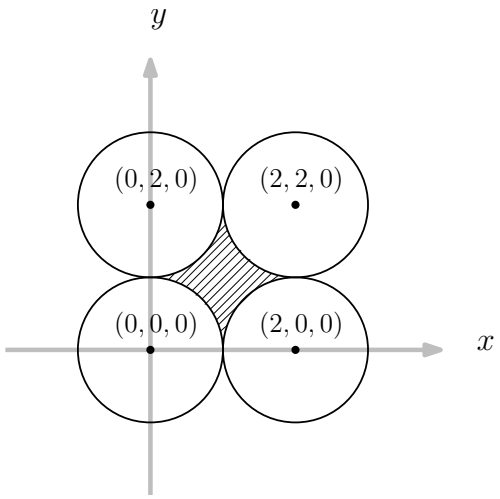


Figure 4.2: Cross section of a tube with the $z = 0$ plane

between two nodes if the corresponding cylinders intersect or touch each other, the resulting graph is a cycle. The concept is best illustrated by an example. Consider 4 vertical cylinders (ie. the cross section with the xy plane is a circle and to the xz or yz plane is a rectangle) with radius 1 and height 2. Let us place the cylinder such that the centres are at location $(0, 0, 0), (0, 2, 0), (2, 2, 0), (2, 0, 0)$. The resulting construct is a tube. The cross-section of this tube with respect to the $z = 0$ plane is shown in Figure 4.2. We can envision a tube as a garden hose where the outer rubber is made up of cylinders. In our reduction, our tube will always be closed at either end (by placing additional sensors). Thus, the tube partition the 3D space into two parts. The region that is shaded in Figure 4.2 is denoted as the *interior* of the tube.

By placing S in the interior of one end of the tube and T in the interior of the other end, we can ensure that the minimal resilience path must be constrained inside

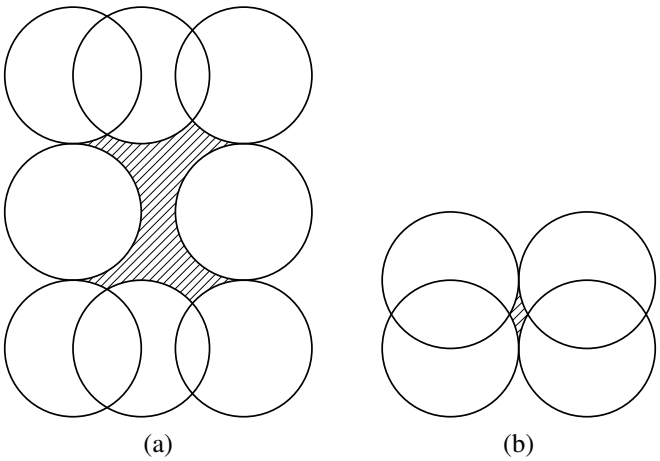


Figure 4.3: Modifying the size of the interior of a tube

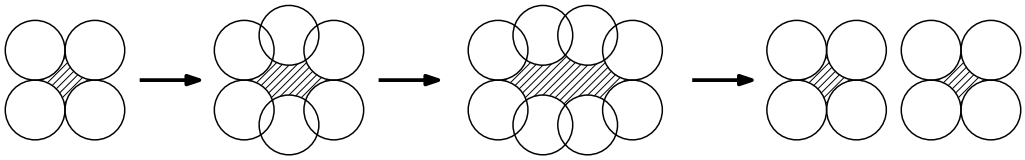


Figure 4.4: Frame by frame cross-section view of a tube splitting into two

the tube. This is similar to the maze environment from the previous reduction where the minimal resilience path is forced to traverse the maze in an orderly fashion.

Tubes are very flexible constructs. Recall that an approximated cylinder can bend and turn. Consequently, tubes may turn and bend as well. We can also control the size of the interior of a tube. Figure 4.3a and 4.3b demonstrates how to increase and decrease the interior of a tube respectively. Another useful property is that we can split a single tube into several tubes. This can be accomplished by first widening the interior of the tube, then inserting another set of cylinders down the middle. Figure 4.4 demonstrates the cross-section views of a tube splitting into two. Finally, note that a tube may use any number of cylinders. For most cases, we will only use 4, as demonstrated in Figure 4.2.

4.1.2 Reduction

The key idea of the reduction is to use a tube to simulate a non-deterministic choice. Consider the scenario depicted in Figure 4.5 (we have drawn the figure in 2D because it is easier to present). The circles represent spherical sensors and the

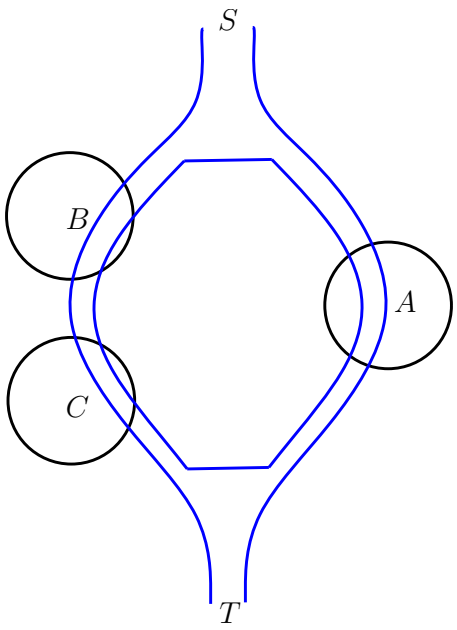


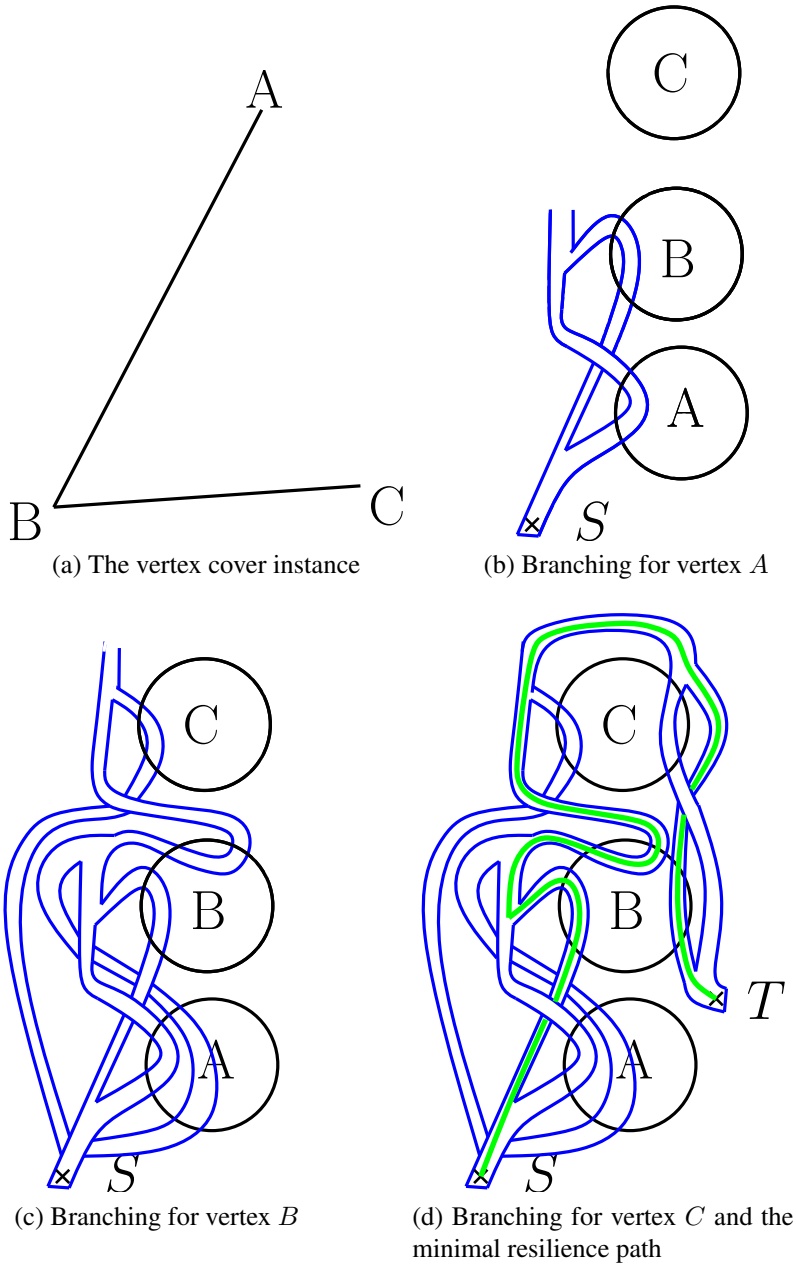
Figure 4.5: Implementing a non-deterministic choice

channels represent the interior of a tube. Any $S - T$ path that does not intersect the wall of the tube is forced to choose whether to cross sensor A or sensor B and C. Note that we may make a channel as small as we want since we can make the interior of a tunnel as small as we want.

We may chain several of the above constructs together in order to simulate a sequence of non-deterministic decisions and thus a non-deterministic algorithm. This is the core principle of the reduction - we will use the tubes to implement the non-deterministic algorithm for solving the VC problem presented in Algorithm 4.1. The algorithm hinges on the observation that for every vertex v, a vertex cover must either include v itself or all neighbours of v.

The first step to implement this algorithm is to represent each vertex in G by a sphere sensor. We denote this set of sensors by *vertex sensors*. Now we add a tube and put both S and T in the interior of the tube. For each vertex v, the tube will branch into two tubes - one will pass through the vertex sensor representing v, the other will pass through the vertex sensors representing the neighbours of v. The two tubes will then join back together afterward. Altogether, the tube will branch $|V|$ times. A simple example is illustrated in Figure 4.6.

Lemma 4.1. *Let $G = (V, E)$ be a graph and $P = (G, K)$ be an instance of VC. Let $P' = (U, \mathcal{A}, S, T, K)$ be an instance of USPHERE-RES where $U, \mathcal{A}, S, T$ are*



(a) The vertex cover instance

(b) Branching for vertex A

(c) Branching for vertex B

(d) Branching for vertex C and the minimal resilience path

Figure 4.6: Sample reduction from USPHERE-RES to VC

Algorithm 4.1 Non-deterministic algorithm to find minimum vertex cover

Given: $G = (V, E)$
Output: S - the minimum vertex cover

$S \leftarrow \emptyset$
for $v \in V$ **do**
  $S \leftarrow S \cup \{v\}$
  OR
  $S \leftarrow S \cup \{u \mid (u, v) \in E\}$
end for
return S

constructed per the description above. Then G has a vertex cover with size less than or equal to K if and only if the resilience of P' is less than or equal to K.

Proof. If the resilience of P' is less than or equal to K, then there exists a $S - T$ path p which intersects at most K distinct sensors. By definition of tubes, p cannot exit the tubes and must stay inside the interior. Thus, we can view p as a series of decisions at the branching points. At the branching point for vertex v, we say p chose the *inclusion branch* if it took the branch of tubes that passed through the vertex sensor v. On the other hand, we say p chose the *exclusion branch* if p chose the branch that intersected the vertex sensors of the neighbours of v.

Now consider the following branch of Algorithm 4.1: for each vertex v, if p chose the inclusion branch, the algorithm adds v to S. Otherwise, the algorithm adds the neighbours of v to S. It is easy to see that at the end of the for loop, each element in S corresponds exactly to the vertex sensors that p visited and S is a vertex cover of G. Thus, G has a vertex cover with size less than or equal to K.

Conversely, let V' be a vertex cover of G such that $|V'| \leq K$. First, note that there exists an execution of Algorithm 4.1 which outputs a vertex cover S such that $S = V'$: for each vertex v, the algorithm will add v to S if $v \in V'$ or add the neighbours of v if $v \notin V'$. By reversing the above mapping, we can show that there exists a $S - T$ path p which intersects exactly the vertex sensors corresponding to the vertex in S. Thus, p intersects at most K distinct sensors and P' has resilience at most K. □

Theorem 4.2. *USPHERE-RES is NP-complete.*

Proof. First, USPHERE-RES is in NP - a certificate consists of the sphere sensors

to remove. We can then check in polynomial time that there is a sensor-free path from S to T. Applying Lemma 4.1 and observing that the transformation described above can be constructed in polynomial time (proven later in Theorem 4.3) shows that USPHERE-RES is NP-hard. □

4.1.3 Implementation Details

In the previous section, we have presented the example in 2D because it is easier to visualize. However, the reduction will not work in 2D because of the cross-overs among different branches of the tube. As drawn, one branch blocks off the other branch entirely - an undesirable effect. However, we can avoid the blockage problem in 3D by having one tube pass beneath the other.

The second subtlety of the reduction is how to handle the scenario where several tubes intersect the same vertex sensor. Recall that every sphere in the sensor network is the same size. For the reduction to work, if a tube intersect a vertex sensor, a portion of the interior of the tube must lie entirely within the vertex sensor. At first glance, it seems like we will run into a "space" problem trying to fit so many tubes into the same vertex sensor. Looking back at Figure 4.6, we had drawn the tubes to be very narrow compared to vertex sensors. Imagine if the thickness of the wall of the tubes were the same width as the vertex sensors. In 2D, the construction will become impossible if too many tubes cross the same vertex sensor.

However, in 3D, we have more flexibility. First, note that using the idea presented in Figure 4.3b, we may make the interior of the tube as small as possible (the two top cylinders only need to offset from the bottom two cylinder by ϵ). Furthermore, we may stack several tubes together very tightly by overlapping the bottom two cylinders of the top tube with the top two cylinders of the bottom tube. Figure 4.7 illustrates the stacking of two tubes. Assuming that the height of each tube interior is ϵ, the total height of two tube interiors is 2ϵ. Similarly, we can extend the stacking so that we can stack k tubes such that the total height of the k interiors is $k\epsilon$. Since we may make ϵ as small as we want, we can fit as many tube interiors as we need into a fixed space (in this case, a vertex sensor).

This provides the following mechanism for us to allow multiple tubes to intersect the same vertex sensor. First, we congregate all tubes that need to cross the same vertex sensor so that they are stacked as described above. These tubes then

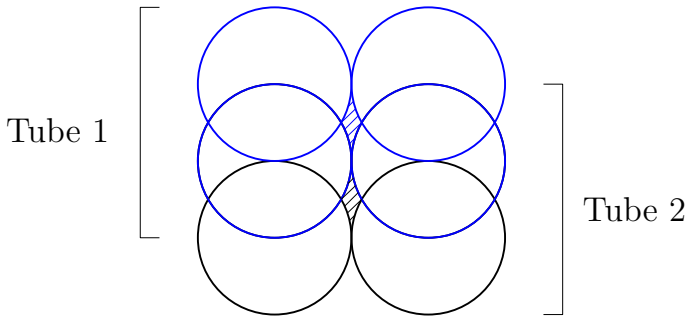


Figure 4.7: Two tubes stacking

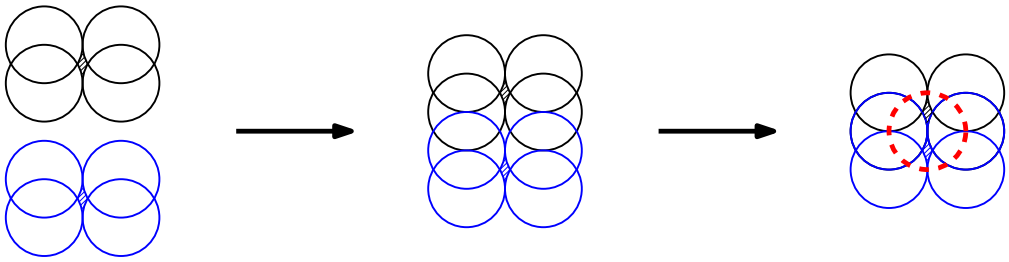


Figure 4.8: Two tubes congregating to pass through a single vertex sensor (shown by the dashed circle)

pass through the vertex sensor together. Figure 4.8 provides a cross-section view of 2 tubes stacking and passing through the same vertex sensor. Afterward, these tubes may disconnect from each other. This can be visualized by reversing the order presented in Figure 4.8. Note that we can use the above procedure because the tubes have a lot of free space to turn and bend.

Theorem 4.3. *Given an instance $P = (G, K)$ of the VC problem, where $G = (V, E)$ is a graph. The reduction to an instance P' of the USPHERE-RES problem can be done in polynomial time.*

Proof. It suffices to show that the total number of sphere sensors used is polynomial. Note that the number of vertex sensors is $|V|$. Thus, the resilience of the network is $\leq |V|$ and we can construct a fence by placing $2|V|$ sensors at the same location. Consequently, we only need to show that the number of fences used for constructing the tubes is polynomial.

Note that the number of fences used for a tube is linearly proportional to the length of the tube. Thus, we only need to show one way to construct the reduction such that the total length of the tubes is polynomial in length.

First, we place vertex sensors on the $z = 0$ plane. We label the vertex sensors from 1 to $|V|$ and place the j^{th} vertex sensor at coordinate $(j|V|, 0, 0)$. We will place S at $(0, 0, 0)$ and T at $(|V|^2 + |V|, 0, 0)$. We will construct the tubes in phases. We handle one branching per phase, resulting in $|V|$ phases in total. Each phase consists of splitting a tube into two branches, weaving each branch of the tube to pass through the appropriate vertex sensors, and finally merging the two branches. For one particular phase, the tube will be constructed such that the middle of the interior of the tubes will all have the same z-coordinate. The first phase will lie in the $z = 0$ plane, the second in the $z = |V|$ plane, the third in the $z = 2|V|$ plane, etc.

We now show how to construct a single phase. First, since the entire phase lies in the same z-plane, we will use 2D coordinates to describe the construction. Within a single phase, a tube always starts at $(0, 0)$ and end at $(|V|^2+|V|, 0)$. The tube will first proceed to the $y > 0$ region before branching. After branching, the two tubes will pass through the appropriate vertex sensors. The tube always passes through a vertex sensor in a very specific manner: to pass through the j^{th} vertex sensor (whose centre is located at $(j|V|, 0, 0)$), the tube will follow the line $x = k|V|$ from the $y > 0$ plane to the $y < 0$ plane. Finally, the two branches will merge at the $y < 0$ plane and eventually end up at $(|V|^2 + |V|, 0)$. Figure 4.9 demonstrates the construction of the tube in the first phase. In the figure, circles represent vertex sensors and curves represent tubes. One branch of the tube passes through sensor A while the other branch passes through B and C. In some cases, the two branches of a tube may intersect. In this case, we simply have one branch pass beneath the other. The phases are well separated enough in the z-direction to allow this.

There are two things missing from the construction. First, we need to add tubes to connect one phase to the next phase. For example, to connect the first phase to the second phase, we need a tube from $(|V|^2 + |V|, 0, 0)$ to $(0, 0, |V|)$. Second, as described above, only tubes in the first phase will actually pass through the vertex sensors. For the other phases, the tubes will pass directly above the vertex sensors. Recall that we need to first stack all the tubes entering the same vertex sensor. In this case, we will stack the tubes in the z direction. To do this, we make a slight modification to the construction of each phase. When a tube is about to pass through a vertex sensor, the tube will actually dive down to the appropriate z-coordinate in order to pass through the vertex sensor. After passing through the vertex sensor, the tube will rise back to the appropriate plane. This is demonstrated in Figure 4.10,

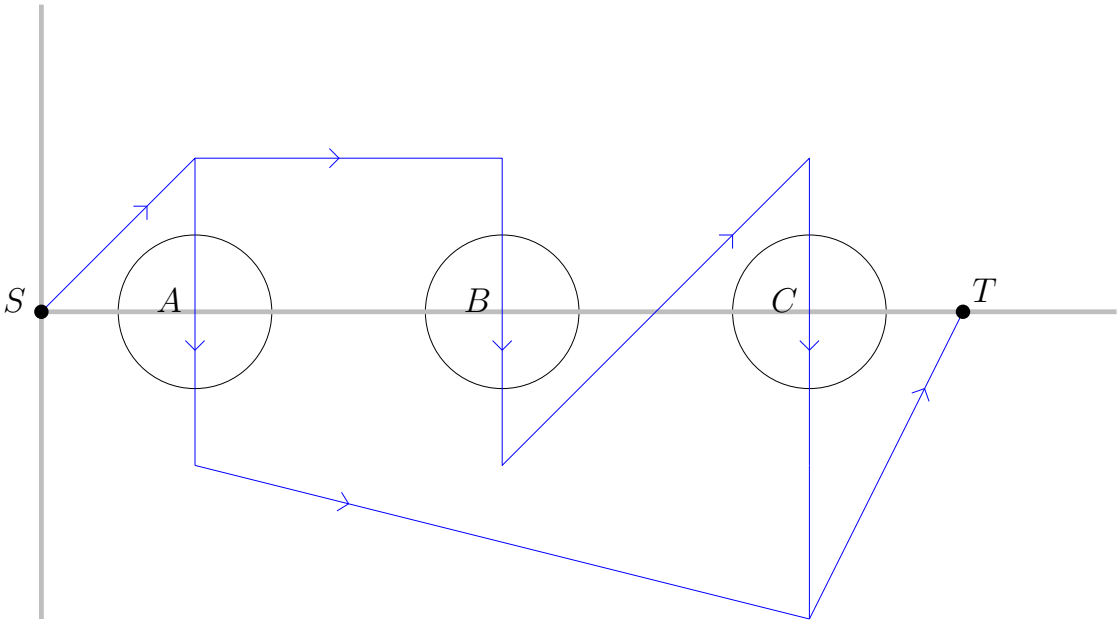


Figure 4.9: Sample construction of tubes in the first phase

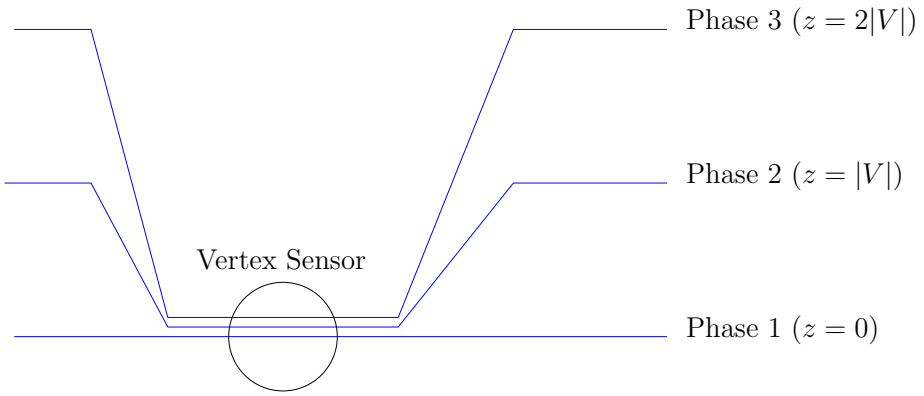


Figure 4.10: Tubes from multiple phases crossing the same vertex sensor

which provides the cross-section view with respect to the plane $x = |V|$. The circle represent the first vertex sensor and the lines represent tubes from different phases passing through the same vertex sensor.

The tube used to connect one phase to the next phase is clearly polynomial in length. Thus, it suffices to show that the total length of tube used in each phase is polynomial. Based on Figure 4.9, we can see that for each branch of the tube, the x-coordinate is strictly increasing. In the y-coordinate, the tube will change direction at most $2|V|$ time (twice for each sensor it need to pass through). Since we can bound each phase by a $O(|V|^2) \times O(|V|^2)$ box, it follows that the total length of

the tube in each phase must be polynomial. Finally, the additional length of the tubes required to achieve the diving effect in Figure 4.10 is polynomial for each dive since the maximum z-coordinate for any tube is $O(|V|^2)$. There are at most $O(|V|^2)$ dives in total, so the total length of the tube is polynomial. □

Chapter 5

Conclusion

We have shown that determining resilience of a unit line segment sensor network in the 2D plane is NP-complete by reducing from the MAX-2-SAT problem. As our construction has shown, the problem remains hard even if the line segment sensors are only oriented horizontally, vertically, and diagonally. Furthermore, it is possible to extend this proof to sensors with non-symmetric coverage regions.

We also investigated the resilience problem in the context of 3D sensor networks. Using a different technique, we proved that determining the resilience of a spherical sensor network is also NP-hard by reducing from the VC problem.

5.1 Future Direction

In our reduction of ULS-RES, we used three orientations of unit line segment sensors. One natural extension is to refine the proof to use only horizontal and vertical segments. For the cost of additional complexity, we strongly believe that it is possible to do this (by simulating diagonal sensors with only horizontal and vertical sensors).

Another extension is to consider whether computing the resilience of a disk sensor network is a hard problem. As mentioned before, our reduction breaks down for this case (even if the disks are allowed to have different radiuii). To our knowledge, the complexity of computing resilience for a disk sensor network is still an open problem.

For practical applications, we often turn to approximation algorithms for NP-

hard problems. A 2-approximation algorithm for resilience of unit disk sensor network has been demonstrated [1]. The authors used a packing argument to show that the minimal resilience path will not intersect the same sensor too many times. However, this is not true in general for other types of sensors. Thus, finding an approximation algorithm for line segment sensors and other non-symmetric shapes is another venue worth exploring.

Bibliography

[1] S. Bereg and D. Kirkpatrick. Approximating barrier resilience in wireless sensor networks. In S. Dolev, editor, *Algorithmic Aspects of Wireless Sensor Networks*, volume 5804 of *Lecture Notes in Computer Science*, pages 29–40. Springer Berlin / Heidelberg, 2009.

[2] M. Cardei and J. Wu. Coverage in wireless sensor networks. In M. Ilyas and I. Mahgoub, editors, *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, chapter 19, pages 432–446. CRC Press, 2005.

[3] D. W. Gage. Command control for many-robot systems. In *Proceedings of the 19th Annual AUVS Technical Symposium, AUVS 92*, pages 28–34, 1992.

[4] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1(3):237 – 267, 1976. ISSN 0304-3975.

[5] W. Gregg, W. Esaias, G. Feldman, R. Frouin, S. Hooker, C. McClain, and R. Woodward. Coverage opportunities for global ocean color in a multimission era. *Geoscience and Remote Sensing, IEEE Transactions on*, 36(5):1620 –1627, Sept. 1998. ISSN 0196-2892.

[6] S. Kumar. *Foundations of coverage in wireless sensor networks*. PhD thesis, Ohio State University, 2006.

[7] S. Kumar, T. H. Lai, and A. Arora. Barrier coverage with wireless sensors. In *Proceedings of the 11th annual international conference on Mobile computing and networking*, MobiCom '05, pages 284–298, New York, NY, USA, 2005. ACM. ISBN 1-59593-020-5.

[8] B. Liu, O. Dousse, J. Wang, and A. Saipulla. Strong barrier coverage of wireless sensor networks. In *Proceedings of the 9th ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '08, pages 411–420, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-073-9.

[9] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. Srivastava. Coverage problems in wireless ad-hoc sensor networks. In *INFOCOM 2001. Twentieth*

Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, volume 3, pages 1380 –1387 vol.3, 2001.

[10] S. Meguerdichian, F. Koushanfar, G. Qu, and M. Potkonjak. Exposure in wireless ad-hoc sensor networks. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, MobiCom '01, pages 139–150, New York, NY, USA, 2001. ACM. ISBN 1-58113-422-3.

Appendix A: Reduction from MAX-2-SAT to USPHERE-RES

Here, we provide a sketch of how one prove that USPHERE-RES is NP-hard by reducing from the MAX-2-SAT problem and extending the techniques presented in Chapter 2. We will also point out the difficulties of using this approach.

In the 2D case, the key idea of the hardness proof is to construct a sensor network consisting of a series of bipartite chains and to embed these chains inside a maze "environment". We will call these two parts the *chain component* and the *environment component* respectively. To extend this approach for USPHERE-RES, we first need to replace each sensor in the chain component and the environment component by spherical sensors. We can let the centre of each spherical sensor lie on the $z = 0$ plane. After combining the two components in a similar fashion as before, we end up with the entire reduction lying on the $z = 0$ plane. However, to ensure the $S - T$ path actually stays within the maze, we need to add obstacles that prevent the path from traversing in the z-direction out of the maze. To do this, we add a layer of dense spherical sensors directly above and below the maze. These can be visualized as the "ceiling" and the "floor" of the maze (see Figure A.1a, which provides an orthogonal view with respect to the $z = 0$ plane)

Finally, we need to handle the cross-overs. As mentioned previously, the third dimension allows us to maintain the independence of the two chains simply by having one pass beneath the other chain. We can have one chain remain at the $z = 0$ plane. The other chain will dive below the plane before the cross-over point and rise back to the plane afterward. Figure A.1b demonstrates the general idea (from the view orthogonal to the $z = 0$ plane). However, this adds several complications to the reduction. When the chain dives below the plane, we still need to enforce the properties of the bipartite chain. This means that the maze walls need to extend

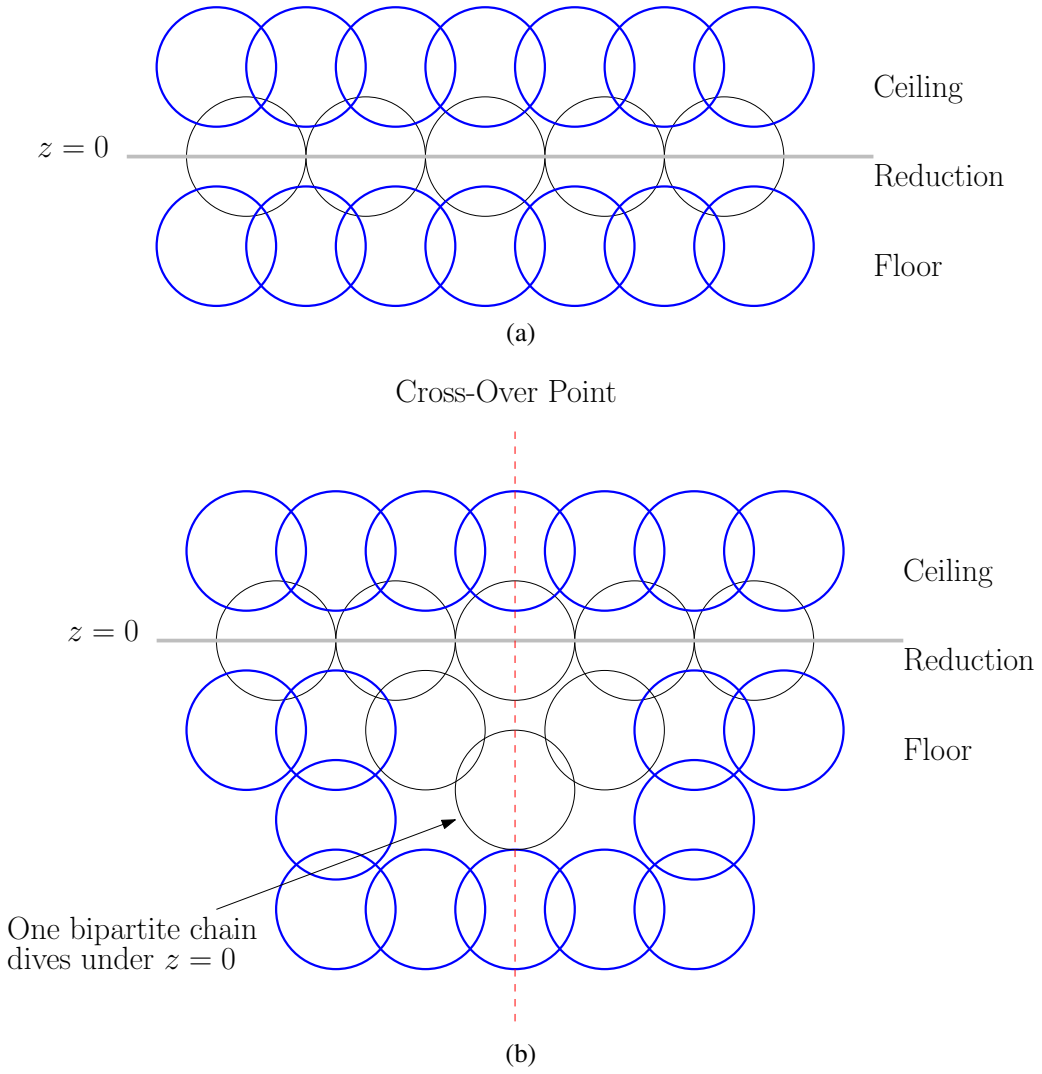


Figure A.1: Extending the previous reduction to 3D

vertically beneath the $z = 0$ plane as well. Consequently, the "floor" needs to be lowered. Thus, in the local area around the cross-over, the $S-T$ path now has much more freedom due to the increased vertical height between the floor and ceilings. Other sensors must be added to make sure that an $S - T$ path cannot simply ignore the two chains.

Overall, constructing a cross-over gadget seem conceptually simple, but the actual realization is quite difficult. There are many subtleties that must be approached with care. Furthermore, this reduction uses an unnecessarily large overhead of spherical sensors. Thus, even though it is possible to extend from previous results, we opt for a different reduction instead.