Approximating Barrier Resilience and Related Notions for Disk Sensors in a Two-Dimensional Plane

by

David Yu Cheng Chan

B.Sc, The University of British Columbia, 2010

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)

October 2012

Abstract

Let $\mathtt{A}$ be an arrangement of n sensors constituting a barrier, the resilience of $\mathtt{A}$ with respect to two regions S and T, denoted $\rho(\mathtt{A}, S, T)$, is defined as the number of sensors in $\mathtt{A}$ that must be removed in order for there to be a path from S to T that is not detected by any sensor.

Previous work by Bereg and Kirkpatrick [2] proved that a 3-approximation of the resilience could be guaranteed by a polynomial time algorithm when sensors are unit disks in the plane. This was tightened to a $\frac{5}{3}$-approximation when S and T are well-separated. In this thesis, we define and analyze a Multi-Path Algorithm (MPA) which improves on these bounds to get approximation ratios of 2 and 1.5 respectively.

Additionally, we define a new notion of barrier strength called D_c-Resilience, which is the resilience of the barrier if regions covered by more than c distinct sensors in the original arrangement are treated as inaccessible, in the sense that no part of any path is permitted to intersect with such a region. We consider arrangements of disk sensors with radii in range $[1-\xi, 1]$ for any $0 \leq \xi < 1$ independent of n. We prove that for any such arrangement and any constant c, the MPA guarantees a constant approximation of the D_c-resilience in time polynomial in n for any constant c. Furthermore, we show that this tightens to a 2-approximation under certain conditions which are usually fulfilled when disk sensors are close to equal size.

Table of Contents

List of Tables

List of Figures

Acknowledgments

We thank David Kirkpatrick for supervising this research thesis.

In addition, we thank Will Evans for his feedback on the thesis and for being in the supervisory committee.

We also thank the University of British Columbia for allowing the use of their facilities for this research.

Finally, we thank family and friends for their support.

Chapter 1

Introduction

The use of Wireless Sensor Networks has been increasing in recent years, and there have even been large scale projects using over a thousand sensors such as described in the work by Arora et al. [1]. In such applications of Wireless Sensor Networks, one of the fundamental issues is to determine the coverage provided by a particular network [15].

While many definitions of coverage exist, two important classes of coverage are Area and Barrier Coverage problems. In both, the input is an arrangement of sensors $\mathtt{A}$, where each sensor is associated with some point representing its position and some area representing the region which is covered by the sensor. Area coverage problems then ask questions related to how well-covered some region R is by $\mathtt{A}$, while barrier coverage problems ask questions related to how well-separated some region S is from some other region T by $\mathtt{A}$.

For area coverage problems, sensors in $\mathtt{A}$ are assumed to be associated with possibly overlapping regions, and the strength of a coverage system is usually measured by either the fraction of R that is covered by $\mathtt{A}$, or by the minimum number of times any point in R is covered in $\mathtt{A}$. The latter is referred to as k-coverage, where is $\mathtt{A}$ said to guarantee k-coverage of R if every point in R is covered by at least k distinct sensors in $\mathtt{A}$ [8].

By these definitions of strength, determining the exact strength of an arrangement $\mathtt{A}$ tends to be straightforward in most practical applications, since $\mathtt{A}$ usually divides R into at most some small polynomial number of subregions, each of which can then be efficiently checked to determine how well-covered they are. Instead, most research reflects the fact that area coverage tends to be much more expensive than barrier coverage and one would ideally want to minimize costs. Thus, most research on area coverage problems ask not how strong an arrangement `A` is, but ask either how to optimize the strength of the arrangement subject to some constraint, or ask how to maximize some function subject to the constraint that the strength of `A` is kept above some minimum level.

For instance, many area coverage problems try to minimize the number of sensors used in `A`, and ask questions like how to cycle the sensors in `A` on and off such that k-coverage is still guaranteed at all times [8, 19, 25]. This is because in many applications, allowing the sensors in `A` to sleep for long periods of time conserves power and extends the lifetime of the sensor network [8, 19, 25].

Many problems also ask how to create `A` entirely to minimize the number of sensors subject to some constraints. For instance, Wang et al. [23] investigate the sensor placement problem, where they have to determine the minimum number of sensors needed in order to have a 1-coverage of R, subject to the constraints that there may be obstacles that block the sensors, sensors have some limited range, and sensors must be connected to one another.

Nevertheless, for Area Coverage, the question of how strong a certain fixed arrangement of sensors is usually has well-known efficient solutions. However, the same does not hold for Barrier Coverage, as even when restricted to the two-dimensional setting there exists many versions of the problem with no known polynomial-time method for determining, or even guaranteeing a constant approximation of the strength of a barrier.

For barrier coverage problems, the strength of an arrangement `A` is usually much more complicated to determine due to the fact that the number of topologically or combinatorially distinct paths through any region divided into distinct subregions is exponential in terms of the number of subregions in the worst case. This means that if an algorithm is not allowed to have a time complexity that increases exponentially with the number of sensors, it cannot check for every path how well-covered it is in a manner similar to how it can check every subregion in the Area Coverage problems. Furthermore, as opposed to the two main area coverage strength measurements, for Barrier Coverage there have been several different ways to measure the strength of a barrier defined, each with their own advantages and disadvantages.

For example, one definition of barrier strength builds on the notion of the *breach* of a path, defined as the minimum Euclidean distance between any point on the path and the point representing the position of any sensor in $\mathtt{A}$. Clearly the strength of a barrier can thus be defined as the breach of the path between S and T that has the maximum breach, otherwise known as the *Maximal Breach Path*. This definition is very advantageous for applications where sensors do not have a clear maximum distance defined but nevertheless are able to detect intruders at close range significantly better than intruders at a far distance. For instance, security cameras when pointed horizontally usually don't have a maximum distance defined, but it obviously much easier to spot intruders nearby than intruders far off in the distance. Thus rather than determine how many cameras may see the intruders, its more useful to determine how close the intruder gets to any camera. As such, much research has been done to construct efficient algorithms for deriving Maximal Breach Paths [3, 4, 14, 15, 17], or even working on variants on it such as the Average Breach Paths defined by Duttagupta et al. [5].

One disadvantage of such a definition is that it doesn't take into account the number of sensors, or the distances to other sensors in the barrier. In other words, an arrangement whose maximal breach path is close to only a single sensor can be given the same strength value as an arrangement whose maximal breach path has to be close to many sensors. In light of this disadvantage, another concept called the *Minimal Exposure Path* was introduced and used as another means of measuring barrier strength. In essence, every point along the path has some exposure to the sensors in $\mathtt{A}$ which is inversely proportional to its distance from the sensors, and the Minimal Exposure Path is the path whose total exposure across all points is the smallest. Like the Maximal Breach Paths, much research has also been done to construct efficient algorithms for deriving Minimal Exposure Paths [12, 13, 16, 22]. However, unlike the Maximal Breach Path where breach has a single clearly defined definition, exposure can be defined in multiple ways and thus there actually exists multiple ways to measure the barrier strength via this method. For instance, Veltri et al. [22] notes that there are two distinct types of exposure functions: functions solely based on the distance to the closest sensor, and functions based on the distances to all sensors. Another advantage this has over the Maximal Breach Path is that sensors can now be directional. In other words, while giving sensors directions would not make much sense when computing the breach of a path, the exposure functions can easily be changed to reflect sensors which have better detection rates in some directions instead of others and recent work by Liu et al. [12] investigates exactly that.

Another possible disadvantage of Maximal Breach Paths is when sensors have some minimum range *r* whereby any entities at distances less than *r* are no easier to detect that entities at distance *r*. In such situations, if the maximal breach path has a breach value less than *r*, the barrier isn't necessarily stronger than any barrier whose maximal breach path has a breach value of *r*, so another notion of strength must be used if one wants to distinguish between these arrangements.

Instead of merely using the points representing the positions of sensors in $\mathtt{A}$, this idea of a minimum range whereby detection rates do not improve any further with decreasing distances allows one to use regions representing the detection area of sensors in $\mathtt{A}$. In other words, these regions represent the area surrounding the sensor that is within its minimum range. This then gives rise to more ways of defining the strength of $\mathtt{A}$.

For instance, Liu et al. [11] defines a barrier as providing weak coverage if all shortest paths through the barrier hit at least one sensor, and defines a barrier as providing strong coverage if all paths through the barrier hit at least one region associated with some sensor in $\mathtt{A}$. This definition is useful when sensors are very reliable or even act like walls, since a barrier providing strong coverage in this definition would truly be impassable, while a barrier that provides only weak coverage would be like a maze that crafty intruders would eventually break through. In fact, stopping crafty intruders from breaking through weak barriers was precisely why Liu et al. [11] chose this definition of strength and focused on developing algorithms for strong barrier coverage.

One advantage in common for the definitions given so far is that they analyze situations where the intruders may be extremely crafty, somehow determining the locations of all sensors and searching for optimal paths through the arrangements. However, they all also share the disadvantage of assuming that sensors are reliable, or in some sense immune to failure. As such, more ways to measure the strength of barriers arose.

Kumar et al. [9, 10] defines the strength of a barrier by the number of distinct sensors any path through the barrier must intersect. In other words, given a set of sensors, if all paths through the region are detected at least k distinct sensors, the region is k-barrier covered. In some sense, this is the k-coverage equivalent for Barrier Coverage problems. This definition addresses the disadvantage noted above, since a k-barrier covered region with larger k would require more sensors to fail in order to have an uncovered path. Kumar et al. [9, 10] proceeded to provide algorithms for determining the strength of barriers in different regions, deriving a method of determining the exact strength of barriers when sensors are disks in a two-dimensional open belt region, defined as a strip-like region that does not form a ring around S or T but has to be crossed by any path from S to T.

Bereg and Kirkpatrick [2] build on this definition by clearly defining two measurements of the barrier strength. The first notion is the thickness of the barrier, defined as the minimum number of times any path from S to T has to enter a sensor in $\mathtt{A}$. The second notion is the resilience of the barrier, defined as the minimum number of sensors that must be removed from $\mathtt{A}$ in order to reduce the thickness to 0. The key difference between the notions is that the thickness will count a sensor multiple times if it enters the sensor multiple times, whereas resilience will only count the sensor once regardless of how many visits the path makes to it. Effectively the resilience of a barrier is defined the same way as the k-barrier coverage definition by Kumar et al. [9, 10].

Bereg and Kirkpatrick [2] considered the scenario where sensors are unit disks in a two dimensional plane. They showed that the thickness of the barrier can be determined in time polynomial in n and illustrated topological properties that prove the thickness forms a 3-approximation of the resilience. Furthermore, if S and T are well-separated relative to the radii of sensors they suggested a method to guarantee a $\frac{5}{3}$-approximation of the resilience [2] in time polynomial in n. The authors also note the similarity of the problem to the NP-hard Minimum Colour Path Problem, which suggests the possibility that computing even a close approximation of the resilience may be NP-hard as well. This Minimum Colour Path Problem is described more thoroughly in the work by Yuan et al. [24], which also provides proofs of its NP-Completeness. Furthermore, work by Tseng and Kirkpatrick [20], Tseng [21] has also proven that if the regions associated with sensors are line segments, the problem of determining the resilience is NP-hard. They also extend this proof to include other types of non-symmetric sensor regions.

In this thesis, we further build on the work by Kumar et al. [9, 10] and Bereg and Kirkpatrick [2] to derive better approximation of the resilience of barriers. Noting the proofs of NP-hardness by Tseng and Kirkpatrick [20], we restrict our attention to disk-shaped sensors in a two-dimensional plane under different conditions. We focus on keeping the time complexity of algorithms in polynomial time, but do not optimize their running time beyond that. Thus some of our algorithms have completely impractical time complexities. We also consider alternative related definitions of barrier strength, argue their usefulness in various applications, and study algorithms for determining or approximating the new measurements.

Chapter 2

Problem Definition

The main problem investigated by this thesis is the Barrier Resilience Problem. The input for this problem is a tuple $(\mathtt{A}, S, T)$, where $\mathtt{A}$ is an arrangement of n sensors corresponding to disk-shaped regions on a two-dimensional plane, and S and T are two regions on the same plane. The problem is to determine the resilience, denoted $\rho(\mathtt{A}, S, T)$, which is the minimum over $S-T$ paths of the number of distinct sensors intersected, including any sensors which cover the endpoints of the path.

As noted by Bereg and Kirkpatrick [2], this problem is similar to the Minimum-Colour Path Problem. The input for this problem is a tuple $(\hat{\mathtt{A}} = (V, E), s', t')$, where:

- $\hat{\mathtt{A}}$ is an edge-coloured graph composed of vertices in V and directed edges in E, with a set of at most n distinct colours denoted by $\mathtt{S}_{\hat{\mathtt{A}}}$, such that each edge $(i, j) \in E$ is assigned a set of the colours $\mathtt{S}_{(i,j)} \subseteq \mathtt{S}_{\hat{\mathtt{A}}}$.

- s' and t' are two vertices in V.

The problem is to determine the $s' - t'$ path in $\hat{\mathtt{A}}$ which uses the fewest distinct colours, where a colour is considered used by a path if it is assigned to any edge in the path. The Minimum-Colour Path Problem has been shown to be NP-hard, and is in fact hard to approximate to within a logarithmic factor [7, 24].

There exists a variant of the Minimum-Colour Path Problem that we refer to as the Minimum-Vertex-Colour Path Problem. The input for this problem is also a tuple $(\hat{\mathtt{A}} = (V, E), s', t')$, but in this variant the edges are not assigned any colours.

Instead, each vertex $i \in V$ is assigned a set of colours $\mathtt{S}_i \subseteq \mathtt{S}_{\hat{\mathtt{A}}}$. As before, the problem is to determine the $s' - t'$ path in $\hat{\mathtt{A}}$ which uses the fewest distinct colours, where a colour is considered used by a path if it is assigned to any vertex in the path. Note that by this definition, it is trivial to show that every $s' - t'$ path must use any colours assigned to s' or t'. Thus without loss of generality we only concern ourselves with simplified problem instances where all such colours have been removed from the graph.

The Minimum-Vertex-Colour Path Problem is useful because there exists a polynomial-time approximation-preserving reduction from the Barrier Resilience Problem:

1. Assign each of the n sensors in $\mathtt{A}$ a unique colour, forming the set $\mathtt{S}_{\hat{\mathtt{A}}}$ of n distinct colours.

2. Construct a graph $\hat{\mathtt{A}} = (V, E)$ such that each face in $\mathtt{A}$ becomes a vertex in V and there exists an edge (i, j) in E if and only if the faces of i and j were adjacent in the arrangement. $\hat{\mathtt{A}}$ is referred to as the dual graph of $\mathtt{A}$, and this construction process is illustrated in Figure 2.1.

3. Assign each vertex i in V the set of colours $\mathtt{S}_i$ which correspond to the set of sensors covering i's associated face in $\mathtt{A}$.

4. For each pair of vertices s' and t' which correspond to faces intersecting S and T respectively:

    (a) Remove from $\hat{\mathtt{A}}$ all of the colours in $\mathtt{S}_{s'} \cup \mathtt{S}_{t'}$.

    (b) Solve the Minimum-Vertex-Colour Path Problem on $(\hat{\mathtt{A}}, s', t')$ to compute $W'(s', t')$, the number of distinct colours used by the Minimum-Colour $s' - t'$ path.

    (c) Readd all the colours removed.

5. Output the minimum over s' and t' of $W'(s', t') + |\mathtt{S}_{s'} \cup \mathtt{S}_{t'}|$.

The reason this is a polynomial-time reduction stems from the following observation:

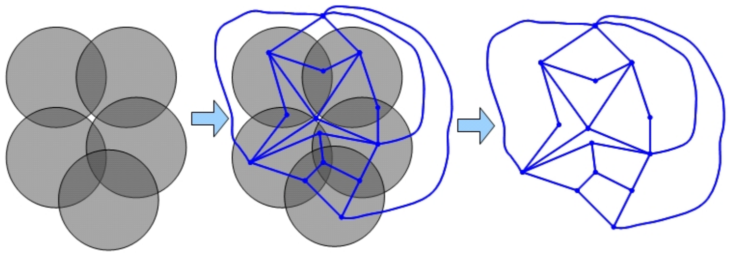

Figure 2.1: Constructing the Dual Graph of an Arrangement

Observation 2.0.1 *Both $|V|$ and $|E|$ are $O(n^2)$.*

Proof First, observe that if the boundary of any disk-shaped sensor s_i does not intersect the boundary of any other sensor, its presence in the arrangement contributes only 1 to the cardinality of V and E. This means replacing s_i with another sensor whose boundary intersects other sensor boundaries could increase $|V|$ and $|E|$ while keeping the number of sensors at n. Thus it suffices to consider only arrangements where all sensors have boundaries which intersect the boundary of at least one other sensor, since we are only aiming to show upper bounds on $|V|$ and $|E|$. Note that this also excludes the scenario where $n \leq 1$, but clearly both $|V|$ and $|E|$ are $O(n^2)$ in such a scenario.

Next, observe that for any pair of disks, the boundaries of the disks can only intersect each other at most 2 times. Effectively, since the number of sensors is n, the boundary of each disk-shaped sensor can have at most $2(n-1)$ intersection points. These intersection points divide the boundary of the sensor into at most $2(n-1)$ boundary arcs. Then consider the dual graph $\hat{A}$. During the construction process, for each pair of adjacent faces i and j, we added two directed edges (i,j) and (j,i). We note that every pair of adjacent faces must be separated by at least one boundary arc, and every boundary arc can be used in this manner for at most one pair of adjacent faces. Thus the number of pairs of adjacent faces is at most $2n(n-1)$, meaning $|E| \leq 4n(n-1)$.

Furthermore, $\hat{A}$ must be a planar graph by construction since the arrangement was on a two-dimensional plane, so by Euler's Formula, $|V|-|E|+|F|=2$, where $|F|$ is the number of faces in $\hat{A}$. Rearranging this and using the bound on $|E|$, we get $|V| + |F| \leq 2 + 4n(n-1)$. Then since there must be at least one face, $|F| \geq 1$, so we get $|V| \leq 1 + 4n(n-1)$. Thus both $|V|$ and $|E|$ are $O(n^2)$. ∎

Consequently, there is only a polynomial number of choices for s' and t', so there is only a polynomial number of Minimum-Vertex-Colour Path Problem instances. The correctness of this reduction follows from the observation that every $S-T$ path through $\mathtt{A}$ corresponds to a $s'-t'$ walk on $\hat{\mathtt{A}}$ which uses a colour if and only if the corresponding sensor is entered by the path, excluding the removed colours. The sensors covering the start and end of the path had their corresponding colours removed in the reduction, and must clearly be intersected by the path, so $|\mathtt{S}_{s'} \cup \mathtt{S}_{t'}|$ are added to $W'(s',t')$ to derive the number of distinct sensors intersected by the path.

Lastly, since each sensor corresponds to a unique colour, this reduction is clearly approximation-preserving. In fact, if the number of colours used in the solution presented by an approximation algorithm for the Minimum-Vertex-Colour Path Problem is at most c times the actual minimum over $s'-t'$ paths of the number of distinct colours used, the output of the reduction would be at most c times the resilience of the barrier.

Unfortunately, there is also a polynomial-time approximation-preserving reduction from the Minimum-Colour Path Problem:

1. Assign every vertex i in V no colour. In other words, $\mathtt{S}_i$ is the empty set. This satisfies the constraint that both $\mathtt{S}_{s'}$ and $\mathtt{S}_{t'}$ are empty sets.

2. For every edge (i,j) in E:

   (a) Add a new vertex k to V with $\mathtt{S}_k = \mathtt{S}_{(i,j)}$.

   (b) Add a new edge (i,k) to E.

   (c) Add a new edge (k,j) to E.

   (d) Remove the edge (i,j) from E.

3. Effectively, every edge is split into two edges, and the new vertex in between is assigned the colours of the edges. This means every path through the graph must use the same colours as before. Thus, we can simply output the solution of the Minimum-Vertex-Colour Path Problem on the resulting $(\hat{\mathtt{A}}, s', t')$.

This is a polynomial-time reduction because the resulting graph has $|V| + |E|$ vertices and $2|E|$ edges. It is correct and approximation-preserving because every path through the original graph corresponds to a path in the resulting graph which must use the same set of colours. Thus the Minimum-Vertex-Colour Path Problem is NP-hard and hard to approximate to within a logarithmic factor as well.

Nevertheless, we note that in the reduction from the Barrier Resilience Problem, because the colours are derived from disk-shaped sensors, there is additional structure in the resulting Minimum-Vertex-Colour Path Problem instances that normally would not be found in general problem instances. With that in mind, in the next chapter we present the Multi-Path Algorithms, a family of deterministic polynomial-time approximation algorithms for the Minimum-Vertex-Colour Path Problem, and the rest of this thesis focuses on elucidating what problem settings are sufficient to guarantee approximations within a constant factor.

Chapter 3

Multi-Path Algorithms

As defined in Chapter 2, the input of the Minimum-Vertex-Colour Path Problem is a tuple $(\hat{\mathtt{A}} = (V,E), s', t')$, where $\hat{\mathtt{A}} = (V,E)$ is a vertex-coloured graph, and s' and t' are vertices in V. In addition, there is a set $\mathtt{S}_{\hat{\mathtt{A}}}$ of n colours such that:

- Every vertex i in V is assigned a set of colours $\mathtt{S}_i \subseteq \mathtt{S}_{\hat{\mathtt{A}}}$.
- Both $\mathtt{S}_{s'}$ and $\mathtt{S}_{t'}$ are empty sets.

The colours used by a path is defined as the union of the colours assigned to its vertices, and $W'(i,j)$ is used to denote the minimum over $i-j$ paths of the number of distinct colours used. Note that by this definition, for any i and j in V, every colour in $\mathtt{S}_j \cup \mathtt{S}_i$ must be used by every $i-j$ path. The problem is to determine $W'(s',t')$, the minimum number of distinct colours used by any $s'-t'$ path in $\hat{\mathtt{A}}$.

The Multi-Path Algorithms are a family of deterministic polynomial-time approximation algorithms for the Minimum-Vertex-Colour Path Problem. These algorithms employ dynamic programming and use techniques from shortest paths algorithms such as the Bellman-Ford algorithm and the Floyd-Warshall algorithm. In particular, these algorithms can work for directed or undirected edges, and do not even require the graph to be connected. If no path exists between s' and t', the algorithm will report this by having output set to ∞.

3.1 Multi-Path Instances and Profiles

For ease of notation, given any vertex tuple $\tilde{\nabla}$, let $\tilde{\nabla}[x]$ denote the x-th vertex in the tuple. Before describing the Multi-Path Algorithms, we must first introduce the concept of Multi-Path instances and Multi-Path profiles. For any positive integer $X \geq 1$, an X-Path instance is an ordered sequence of X' paths for some positive integer $1 \leq X' \leq X$.

An X-Path profile is defined as a tuple $\tilde{\nabla}$ of $2X'$ vertices in V for some positive integer $1 \leq X' \leq X$. It serves as a description of the vertices used as endpoints by an X-Path instance. Any X-Path instance composed of X' paths is the realization of the X-Path profile $\tilde{\nabla}$ if for all $1 \leq x \leq X'$, the x-th path in the X-Path instance starts at $\tilde{\nabla}[2x-1]$ and ends at $\tilde{\nabla}[2x]$.

Note that by these definitions, for any positive integer $X^* \geq X$, an X-Path instance is also an X^*-Path instance, and an X-Path profile is also an X^*-Path profile. For convenience, we will also describe an X-Path instance as a $\tilde{\nabla}$ instance if it is a realization of the X-Path profile $\tilde{\nabla}$. Also note that each X-Path profile may be realized by several different X-Path instances, but each X-Path instance can only be described by one X-Path profile.

Next, observe that a pair of directed paths from i to j and from j to k can be concatenated at j into an $i-k$ path, in the sense that an entity starting at i can follow the first path to j and then follow the second path to k. However, a pair of directed paths from i to j and from k to j cannot be concatenated into an $i-k$ path, since the direction of the second path is reversed. In other words, the directions and shared vertices of the paths affects whether they can be combined into a single path.

A similar situation arises when we want to combine a pair of X-Path instances: the ordering of the paths in each X-Path instance as well as their directions determine what combinations are possible, if any. As before, each pair of paths can only be concatenated at endpoints sharing a vertex, and the start of one path must be the end of the other. However, there are also additional restrictions imposed by the ordering of each X-Path instance. For example, consider any combination where for both X-Path instances, the first path is concatenated to the last path of the other. Such a combination would violate an ordering, since the X-Path instance that results from the combination would inevitably place a path that was last in the ordering of one X-Path instance before the path that was first in the ordering for that same X-Path instances. Thus such combinations are not allowed.

Let $\text{len}(\hat{P})$ denote the number of paths in $\hat{P}$. A pair of X-Path instances $\hat{P}_1$ and $\hat{P}_2$ can be combined to form an X-Path instance $\hat{P}_3$ if and only if there exists an ordered sequence of paths composed of the paths of $\hat{P}_1$ and $\hat{P}_2$, and this ordered sequence is such that:

- For any $1 < x \leq \text{len}(\hat{P}_1)$, the x-th path of $\hat{P}_1$ is after the $(x-1)$-th path of $\hat{P}_1$ in this ordered sequence.
- Similarly, for any $1 < x \leq \text{len}(\hat{P}_2)$, the x-th path of $\hat{P}_2$ is after the $(x-1)$-th path of $\hat{P}_2$ in this ordered sequence.
- There exists a sequence of path combination operations on this ordered sequence such that:
  - Each operation combines two consecutive paths in the sequence, reducing the number of paths in the sequence by one. Note that this path combination must respect the directions and shared vertices of the paths as described above for single paths.
  - The result of this sequence of path combination operations is an ordered sequence of paths corresponding to $\hat{P}_3$, in the sense that for all $1 \leq x \leq \text{len}(\hat{P}_3)$, the x-th path in this sequence is the x-th path in $\hat{P}_3$.

Note that this means whether two X-Path instances can be combined depends only on the endpoints, the ordering, and the directions of the paths. These details are all captured by the X-Path profiles of these X-Path instances. As a result, whether two X-Path instances can be combined can be determined by only examining their X-Path profiles, and any two X-Path instances which are realizations of the same X-Path profile can be combined with other X-Path instances in the same manner. With this in mind, we define the predicate $\oplus(\tilde{\mathbb{V}}_1, \tilde{\mathbb{V}}_2, \tilde{\mathbb{V}}_3)$ as true if and only if any $\tilde{\mathbb{V}}_1$ instance and any $\tilde{\mathbb{V}}_2$ instance can be combined into a $\tilde{\mathbb{V}}_3$ instance.

3.2 Exposed and Hidden Colours

As previously defined, the colours used by any path is the union of the colours used by its edges. The problem is to find $W'(s',t')$, which was defined as the minimum over all $s'-t'$ paths of the number of distinct colours used. As a straightforward extension, the colours used by any X-Path instance is defined as the union of the colours used by its paths, and we will use $W'(\tilde{\nabla})$ to denote the minimum over all $\tilde{\nabla}$ instances of the number of distinct colours used. Thus the problem is to find $W'(\tilde{\nabla}_{\text{output}})$, where $\tilde{\nabla}_{\text{output}}$ is a 1-Path profile with $\tilde{\nabla}_{\text{output}}[1] = s'$ and $\tilde{\nabla}_{\text{output}}[2] = t'$.

Let $\tilde{\nabla}$ be an X-Path profile, let $\hat{P}$ be a $\tilde{\nabla}$ instance. Then consider the colours used by $\hat{P}$. We can divide these colours into two types: the *exposed colours* are colours which appear on an endpoint of at least one of the paths in $\hat{P}$, while the *hidden colours* are the other colours used by $\hat{P}$. The reason for this distinction is the fact that exposed colours are trivial to determine: simply check the colours assigned to each of the path endpoints in $\hat{P}$. Furthermore, because these path endpoints are described by the X-Path profile $\tilde{\nabla}$, the exposed colours for each $\tilde{\nabla}$ instance are the same.

On the other hand, hidden colours are harder to determine, as every vertex in each of the paths of $\hat{P}$ must be considered. In some sense, the hardness of approximating the minimum over $\tilde{\nabla}$ instances of the number of distinct colours used stems from approximating the number of distinct hidden colours, rather than the number of distinct exposed colours.

This motivates a new measure $W^*(\tilde{\nabla})$, which denotes the minimum over $\tilde{\nabla}$ instances of the number of hidden colours. By this definition, it is trivial to show that $W^*(\tilde{\nabla}) \leq W'(\tilde{\nabla})$ for all $\tilde{\nabla}$, since a hidden colour is by definition a used colour. Furthermore, because $\mathtt{S}_{s'}$ and $\mathtt{S}_{t'}$ are empty sets, $W^*(\tilde{\nabla}_{\text{output}}) = W'(\tilde{\nabla}_{\text{output}})$.

3.3 The X-Path Algorithm

In this section, for each positive integer $X > 0$ we define the X-Path Algorithm. The X-Path Algorithm is similar to an all-pairs shortest path algorithm: rather than focusing on solving only $W'(\tilde{\nabla}_{\text{output}})$, it approximates $W^*(\tilde{\nabla})$ for all X-Path profiles. It behaves very similar to the Floyd-Warshall algorithm, in that it starts by considering trivial X-Path instances corresponding to single vertices or edges in the graph as realizations for some X-Path profiles, and then recursively combines the realizations of pairs of X-Path profiles to form realizations for more X-Path profiles.

The X-Path algorithm operates in a sequence of phases, where in each phase $y \geq 0$, it creates an array d_y with a unique entry $d_y[\tilde{\mathtt{V}}]$ for every X-Path profile $\tilde{\mathtt{V}}$. In the course of the algorithm, each array entry $d_y[\tilde{\mathtt{V}}]$ will be assigned either ∞ or a non-negative integer.

Let $\mathrm{len}(\tilde{\mathtt{V}})$ denote the length of $\tilde{\mathtt{V}}$. Let $\tilde{\mathtt{V}}_0$ be an X-Path profile where for all $1 \leq x \leq \frac{\mathrm{len}(\tilde{\mathtt{V}}')}{2}$, either $\tilde{\mathtt{V}}'[2x-1] = \tilde{\mathtt{V}}'[2x]$, or $(\tilde{\mathtt{V}}'[2x-1], \tilde{\mathtt{V}}'[2x])$ is an edge in E. Given these constraints, there clearly exists a realization $\hat{P}_0$ of $\tilde{\mathtt{V}}'$ whose every path is composed of either a single edge or a single vertex. In particular, $\hat{P}_0$ would have no hidden colours, because all the vertices in $\hat{P}_0$ would be path endpoints. Thus $W^*(\tilde{\mathtt{V}}')$ must be 0. In phase 0, array entries associated with such X-Path profiles are assigned a value of 0, while all other array entries are assigned ∞.

For subsequent phases, a special function referred to as the $\mathtt{M}$-function is used to assign values to the array. In order to describe it, we shall first define a few functions. Let $\mathtt{E}$ be a function that takes as input a colour $\mathtt{c}$ and an X-Path profile $\tilde{\mathtt{V}}$. $\mathtt{E}$ outputs 1 if $\mathtt{c}$ is an exposed colour for every realization of $\tilde{\mathtt{V}}$, and outputs 0 otherwise. More formally:

$$\mathtt{E}(\mathtt{c}, \tilde{\mathtt{V}}) = \begin{cases} 1 & \text{if } \exists_{1 \leq x \leq \mathrm{len}(\tilde{\mathtt{V}})} \mathtt{c} \in \mathtt{S}_{\tilde{\mathtt{V}}[x]} \\ 0 & \text{otherwise} \end{cases}$$

Next, let $\mathtt{H}$ be a function that takes as input a colour $\mathtt{c}$ and three X-Path profiles $\tilde{\mathtt{V}}_1$, $\tilde{\mathtt{V}}_2$, and $\tilde{\mathtt{V}}_3$ where $\oplus(\tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2, \tilde{\mathtt{V}}_3)$. $\mathtt{H}$ outputs 1 if $\mathtt{c}$ is an exposed colour $\tilde{\mathtt{V}}_1$ or $\tilde{\mathtt{V}}_2$ but not $\tilde{\mathtt{V}}_3$, and outputs 0 otherwise. In other words, by combining a $\tilde{\mathtt{V}}_1$ instance and a $\tilde{\mathtt{V}}_2$ instance into a $\tilde{\mathtt{V}}_3$ instance, the colour $\mathtt{c}$ goes from being an exposed colour to being a hidden colour for the $\tilde{\mathtt{V}}_3$ instance. More formally:

$$\mathtt{H}(\mathtt{c}, \tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2, \tilde{\mathtt{V}}_3) = \begin{cases} 1 & \text{if } (\mathtt{E}(\mathtt{c}, \tilde{\mathtt{V}}_1) = 1 \vee \mathtt{E}(\mathtt{c}, \tilde{\mathtt{V}}_2) = 1) \wedge \mathtt{E}(\mathtt{c}, \tilde{\mathtt{V}}_3) = 0 \\ 0 & \text{otherwise} \end{cases}$$

Observation 3.3.1 *Given a $\tilde{V}_1$ instance $\hat{P}_1$ with at most d_1 hidden colours, and a*

$\tilde{V}_2$ instance $\hat{P}_2$ with at most d_2 hidden colours, if $\hat{P}_1$ and $\hat{P}_2$ are combined into a $\tilde{V}_3$ instance $\hat{P}_3$, $\hat{P}_3$ has at most $d_1 + d_2 + \sum_{c \in S_{\hat{A}}}(\mathtt{H}(\mathtt{c}, \tilde{V}_1, \tilde{V}_2, \tilde{V}_3))$ hidden colours.

Proof Since $\hat{P}_3$ is the combination of $\hat{P}_1$ and $\hat{P}_2$, the colours used by $\hat{P}_3$ must be the union of the colours used by $\hat{P}_1$ and $\hat{P}_2$. As defined above, a Multi-Path instance uses all of its hidden colours and all of its exposed colours. Thus, $\hat{P}_3$ can only use a colour $\mathtt{c}$ if it is one of the hidden colours of $\hat{P}_1$ or $\hat{P}_2$, or $\mathtt{E}(\mathtt{c}, \tilde{\mathtt{V}}_1) = 1 \vee \mathtt{E}(\mathtt{c}, \tilde{\mathtt{V}}_2) = 1$. Then as defined above, $\hat{P}_3$ must have at most $d_1 + d_2 + \sum_{\mathtt{c} \in \mathtt{S}_{\hat{\mathtt{A}}}}(\mathtt{H}(\mathtt{c}, \tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2, \tilde{\mathtt{V}}_3))$ hidden colours. ▮

Finally, the $\mathtt{M}$-function of the X-Path Algorithm is described as follows:

$$\mathtt{M}(d_1, d_2, \tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2, \tilde{\mathtt{V}}_3) = \begin{cases} d_1 + d_2 + \sum_{\mathtt{c} \in \mathtt{S}_{\hat{\mathtt{A}}}}(\mathtt{H}(\mathtt{c}, \tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2, \tilde{\mathtt{V}}_3)) & \text{if } \oplus(\tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2, \tilde{\mathtt{V}}_3) \\ \infty & \text{otherwise} \end{cases}$$

In each phase $y > 0$, the $\mathtt{M}$-function is used to assign values to all array entries $d_y[\tilde{\mathtt{V}}]$ as follows:

$$d_y[\tilde{\mathtt{V}}] = \min\{d_{y-1}[\tilde{\mathtt{V}}], \min_{\oplus(\tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2, \tilde{\mathtt{V}}_3)}(\mathtt{M}(d_{y-1}[\tilde{\mathtt{V}}_1], d_{y-1}[\tilde{\mathtt{V}}_2], \tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2, \tilde{\mathtt{V}}))\}$$

The sequence of phases terminates after completing a phase $Y > 0$ where $d_Y[\tilde{\mathtt{V}}] = d_{Y-1}[\tilde{\mathtt{V}}]$ for all X-Path profiles $\tilde{\mathtt{V}}$. The algorithm then outputs $d_Y[\tilde{\mathtt{V}}_{\text{output}}]$, where as previously defined, $\tilde{\mathtt{V}}_{\text{output}}$ is a 1-Path profile with $\tilde{\mathtt{V}}_{\text{output}}[1] = s'$ and $\tilde{\mathtt{V}}_{\text{output}}[2] = t'$.

Theorem 3.3.2 *The time complexity of the X-Path Algorithm is $O(n|V|^{5X})$.*

Proof Begin by observing that the number of distinct X-Path profiles is $O(|V|^{2X})$. Thus in each phase $y > 0$, a naive implementation of X-Path Algorithm would make $O(|V|^{6X})$ calls to the $\mathtt{M}$-function. Furthermore, each call of $\mathtt{M}$-function may require up to $O(n)$ time, since it has to perform a summation over the at most n colours in $\mathtt{S}_{\hat{\mathtt{A}}}$. Thus each phase $y > 0$ would take $O(n|V|^{6X})$ time.

However, observe that the $\mathtt{M}$-function only produces finite output when we have $\oplus(\tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2, \tilde{\mathtt{V}}_3)$, and this constraint can only be true if the number of distinct vertices in the vertex tuples of the input X-Path profiles is at most $3X$. Thus by only making `M`-function calls with inputs that can produce finite output, the number of `M`-function calls can be reduced to $O(|V|^{3X})$. Furthermore, observe that the summation over the colours in $\mathtt{S}_{\hat{\mathtt{A}}}$ is independent of the current phase, and can thus be precomputed in a lookup table. This precomputation would require $O(n|V|^{3X})$ time, but would effectively reduce the time complexity of each `M`-function call to $O(1)$. Thus each phase $y > 0$ would take $O(|V|^{3X})$ time.

Next, observe that since there are at most n colours, no array entry should ever be assigned a finite value greater than n. Furthermore, every value assigned must be a non-negative integer and every phase must produce an array such that all array entries have values at most equal to the corresponding entries in the previous array. In fact, every non-terminal phase must have at least one array entry which has a weight strictly less than the corresponding entry in the previous array. Since there are $O(|V|^{2X})$ array entries which may be assigned at most $O(n)$ distinct weights, this means the number of phases must be $O(n|V|^{2X})$.

Thus the total time complexity is $O(n|V|^{5X})$, including the precomputation time. Note that this is under the assumption that initialization phase 0 which reads the input graph can be completed in $O(n|V|^{5X})$ time, which we note is true when the input graph is provided in a standard representation such as an adjacency list or an adjacency matrix. ■

3.4 Valid Derivation Trees for a Multi-Path Algorithm

To make Multi-Path Algorithms easier to analyze, this section introduces the concept of valid derivation trees. Observe that as defined in the previous section, in every phase $y > 0$, every array entry $d_y[\tilde{\mathrm{V}}]$ with a finite value must have had its value derived from a pair of array entries in a previous array. Thus given any array entry with a finite value, it is straightforward to construct a derivation tree to see how its value was derived from the array entries in d_0.

Let VDT be a binary tree with levels numbered in ascending order such that the bottom level of VDT is 0. VDT is an *X-Path valid derivation tree* for some positive integer $X > 0$ if it obeys the following constraints:

- All leaf nodes of VDT are at level 0.

- Each internal node has exactly two child nodes.
- Each internal node must be at a higher level than its child nodes.
- Each node corresponds to an X-Path profile $\tilde{\mathtt{V}}_{\text{Node}}$.
- Each leaf node must have an X-Path profile $\tilde{\mathtt{V}}_{\text{Node}}$ such that in the X-Path Algorithm, the array entry $d_0[\tilde{\mathtt{V}}_{\text{Node}}] = 0$.
- For each internal node, let $\tilde{\mathtt{V}}_1$ and $\tilde{\mathtt{V}}_2$ denote the X-Path profiles of its child nodes. Then the internal node must have an X-Path profile $\tilde{\mathtt{V}}_{\text{Node}}$ such that $\oplus(\tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2, \tilde{\mathtt{V}}_{\text{Node}})$.

Note that by this definition, any subtree of an X-Path valid derivation tree must also be an X-Path valid derivation tree. Furthermore, observe that any X-Path valid derivation tree is also an $(X+i)$-Path valid derivation tree since every X-Path profile is by definition an $(X+i)$-Path profile. Then given any X-Path valid derivation tree VDT, we assign each node of VDT a weight w_{Node} in ascending order as follows:

- If the node is a leaf node, set $w_{\text{Node}} = 0$. Note that as defined above, this is equivalent to $d_0[\tilde{\mathtt{V}}_{\text{Node}}]$.
- If the node is an internal node, let $\tilde{\mathtt{V}}_{\text{Node}}$ denote the X-Path profile of the node and let $\tilde{\mathtt{V}}_1$ and $\tilde{\mathtt{V}}_2$ denote the X-Path profiles of its two child nodes. Similarly, let w_1 and w_2 denote the weights of the two child nodes. Then set $w_{\text{Node}} = \mathtt{M}(w_1, w_2, \tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2, \tilde{\mathtt{V}}_{\text{Node}})$. Note that this must be a finite value since $\oplus(\tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2, \tilde{\mathtt{V}}_{\text{Node}})$.

Note that all weights assigned are finite, non-negative integers. Similarly, we assign each node of VDT an X-Path instance $\hat{P}_{\text{Node}}$ which is a realization of its X-Path profile $\tilde{\mathtt{V}}_{\text{Node}}$ in ascending order as follows:

- If the node is a leaf node, $d_0[\tilde{\mathtt{V}}_{\text{Node}}] = 0$ in the X-Path Algorithm, so there must exist a realization of $\tilde{\mathtt{V}}_{\text{Node}}$ whose every path is composed of either a single edge or a single vertex. Simply set this realization as $\hat{P}_{\text{Node}}$.

- If the node is an internal node, let $\tilde{\mathtt{V}}_1$ and $\tilde{\mathtt{V}}_2$ denote the X-Path profiles of its two child nodes. Similarly, let $\hat{P}_1$ and $\hat{P}_2$ denote the weights of the two child nodes. As defined above, it must be the case that $\oplus(\tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2, \tilde{\mathtt{V}}_{\text{Node}})$, so set $\hat{P}_{\text{Node}}$ to the $\tilde{\mathtt{V}}_{\text{Node}}$ instance formed by combining $\hat{P}_1$ and $\hat{P}_2$.

We note that by the nature of the X-Path Algorithm, it in some sense searches over all X-Path valid derivation trees. Thus not only does every array entry have a corresponding X-Path valid derivation tree, the X-Path Algorithm must also perform no worse than any X-Path valid derivation tree. However, for completeness we shall provide proof of these claims:

Lemma 3.4.1 *For any given positive integer $X > 0$, every array entry $d_y[\tilde{\mathtt{V}}]$ with a finite value in the X-Path Algorithm must have at least one X-Path valid derivation tree* VDT *with a y-th level root node which has X-Path profile $\tilde{\mathtt{V}}_{\text{Node}} = \tilde{\mathtt{V}}$ and weight $w_{\text{Node}} = d_y[\tilde{\mathtt{V}}]$.*

Proof If this is false for some positive integer $X > 0$, consider the earliest phase y' where there exists a counterexample array entry $d_{y'}[\tilde{\mathtt{V}}']$ with finite value but no X-Path valid derivation tree VDT with a y'-th level root node which has X-Path profile $\tilde{\mathtt{V}}_{\text{Node}} = \tilde{\mathtt{V}}'$ and weight $w_{\text{Node}} = d_{y'}[\tilde{\mathtt{V}}']$.

First, observe that it is trivial to construct an X-Path valid derivation tree VDT$'$ with a 0-th level root node which has X-Path profile $\tilde{\mathtt{V}}_{\text{Node}} = \tilde{\mathtt{V}}'$ and weight $w_{\text{Node}} = d_0[\tilde{\mathtt{V}}']$. Thus y' cannot be 0. Then as defined in Section 3.3, observe that $d_{y'}[\tilde{\mathtt{V}}']$ must be equivalent to $\mathtt{M}(d_{y'-1}[\tilde{\mathtt{V}}_1], d_{y'-1}[\tilde{\mathtt{V}}_2], \tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2, \tilde{\mathtt{V}}')$ for some pair of X-Path profiles $\tilde{\mathtt{V}}_1$ and $\tilde{\mathtt{V}}_2$ where $\oplus(\tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2, \tilde{\mathtt{V}}')$.

Next, because of how y' was defined, there must exist an X-Path valid derivation tree VDT$_1$ with a $(y'-1)$-th level root node which has X-Path profile $\tilde{\mathtt{V}}_1$ and weight $w_1 = d_{y'-1}[\tilde{\mathtt{V}}_1]$. Similarly, there must exist an X-Path valid derivation tree VDT$_2$ with a $y'-1$-th level root node which has X-Path profile $\tilde{\mathtt{V}}_2$ and weight $w_2 = d_{y'-1}[\tilde{\mathtt{V}}_2]$. Let $\hat{P}_1$ and $\hat{P}_2$ be the $\tilde{\mathtt{V}}_1$ instance and the $\tilde{\mathtt{V}}_2$ instance associated with the root nodes of VDT$_1$ and VDT$_2$ respectively.

Construct a binary tree VDT$'$ whose root node has X-Path profile $\tilde{\mathtt{V}}_{\text{Node}} = \tilde{\mathtt{V}}'$ and weight $w_{\text{Node}} = d_{y'}[\tilde{\mathtt{V}}']$, and has the root nodes of VDT$_1$ and VDT$_2$ as its two child nodes. The root node of VDT$'$ also has X-Path instance $\hat{P}_{\text{Node}}$, which is the

$\tilde{\nabla}'$ instance formed by combining $\hat{P}_1$ and $\hat{P}_2$. Note that this must be possible since $\tilde{\nabla}_1 \oplus \tilde{\nabla}_2 = \tilde{\nabla}'$. Then observe that because the subtrees of VDT$'$ are X-Path valid derivation trees and the root node of VDT$'$ satisfies all constraints required, VDT$'$ is an X-Path valid derivation tree.

Finally, since the root node of VDT$'$ must be on level y', there clearly exists an X-Path valid derivation tree VDT with a y'-th level root node with X-Path profile $\tilde{\nabla}'$ and weight $d_{y'}[\tilde{\nabla}']$. This contradiction means there can be no phase y' where there exists such a counterexample array entry. ∎

As defined in Section 3.3, a Multi-Path Algorithm only terminates its sequence of phases after completing a phase $Y > 0$ where the resulting array d_Y is identical to the previous array d_{Y-1}. We note that even if it was allowed to continue, all subsequent phases would by definition perform the same operations as phase Y and ultimately generate arrays identical to d_{Y-1} as well. Thus for all positive integers $y > Y$ and all $\tilde{\nabla}$, we say $d_y[\tilde{\nabla}] = d_Y[\tilde{\nabla}]$ since that would be the value assigned if the algorithm was permitted to continue.

Lemma 3.4.2 *No X-Path valid derivation tree has a y-th level root node with X-Path profile $\tilde{\nabla}_{\text{Node}}$ and weight $w_{\text{Node}} < d_y[\tilde{\nabla}_{\text{Node}}]$.*

Proof In some sense, this means the X-Path Algorithm must perform at least as well as any X-Path valid derivation tree. This can be shown via a proof by contradiction. If the lemma is false, there must exist some counterexample X-Path valid derivation tree VDT$'$ that has a y'-th level root node with X-Path profile $\tilde{\nabla}'_{\text{Node}}$ and weight $w_{\text{Node}} < d_{y'}[\tilde{\nabla}'_{\text{Node}}]$. Consider the counterexample with the smallest possible value of y'.

Firstly, y' cannot be 0, because VDT$'$ would consist of a single leaf node and it would have $w_{\text{Node}} = d_0[\tilde{\nabla}'_{\text{Node}}] = d_{y'}[\tilde{\nabla}'_{\text{Node}}]$. Therefore, the root node of VDT$'$ must be an internal node.

Thus the root node of VDT$'$ must have exactly two child nodes. As before, let $\tilde{\nabla}_1$, w_1, and $\hat{P}_1$, and $\tilde{\nabla}_2$, w_2, and $\hat{P}_2$ denote the X-Path profiles, weights, and X-Path instances of the child nodes respectively. Then $\tilde{\nabla}_1 \oplus \tilde{\nabla}_2 = \tilde{\nabla}'_{\text{Node}}$, and $w'_{\text{Node}} = \mathtt{M}(w_1, w_2, \tilde{\nabla}_1, \tilde{\nabla}_2, \tilde{\nabla}'_{\text{Node}})$ where the $\mathtt{M}$-function of the X-Path Algorithm is used.

However, observe that each child node is the root node of a subtree which is also a X-Path valid derivation tree, and since we are considering the counterexample with the smallest possible value of y', it must be the case that $w_1 \geq d_{y'-1}[\tilde{\mathtt{V}}_1]$ and $w_2 \geq d_{y'-1}[\tilde{\mathtt{V}}_2]$. Then since the $\mathtt{M}$-function is monotonically increasing with the first two parameters, $w'_{\text{Node}} \geq \mathtt{M}(d_{y'-1}[\tilde{\mathtt{V}}_1], d_{y'-1}[\tilde{\mathtt{V}}_2], \tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2, \tilde{\mathtt{V}}'_{\text{Node}})$. Furthermore, by definition, $d_{y'}[\tilde{\mathtt{V}}'_{\text{Node}}] \leq \mathtt{M}(d_{y'-1}[\tilde{\mathtt{V}}_1], d_{y'-1}[\tilde{\mathtt{V}}_2], \tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2, \tilde{\mathtt{V}}'_{\text{Node}})$. This means $w'_{\text{Node}} \geq d_{y'}[\tilde{\mathtt{V}}'_{\text{Node}}]$. Thus we have proven that no counterexample X-Path valid derivation tree can exist. ∎

Let $\tilde{\mathtt{V}}_{\text{Node}}$, w_{Node}, and $\hat{P}_{\text{Node}}$ be the X-Path profile, weight, and X-Path instance associated with a node in an X-Path valid derivation tree. As defined above, it must be the case that $\hat{P}_{\text{Node}}$ is a $\tilde{\mathtt{V}}_{\text{Node}}$ instance. In order to describe how each array entry $d_y[\tilde{\mathtt{V}}]$ in the X-Path Algorithm relates to $W^*(\tilde{\mathtt{V}})$, it is also useful to describe the relationship between w_{Node} and $\hat{P}_{\text{Node}}$:

Lemma 3.4.3 *For any X-Path valid derivation tree* VDT*, if a tree node has weight w_{Node} and X-Path instance $\hat{P}_{\text{Node}}$, then $\hat{P}_{\text{Node}}$ has at most w_{Node} hidden colours.*

Proof If this lemma is false, there must exist a counterexample node which has the lowest level in VDT. This counterexample node cannot be a leaf node, because $\hat{P}_{\text{Node}}$ would then consist of at most one edge and would by definition have 0 hidden colours.

Thus this counterexample node must be an internal node with exactly two child nodes. Let $\tilde{\mathtt{V}}_{\text{Node}}$ denote the X-Path profile of this counterexample node, and let $\tilde{\mathtt{V}}_1$, w_1, and $\hat{P}_1$, and $\tilde{\mathtt{V}}_2$, w_2, and $\hat{P}_2$ denote the X-Path profiles, weights, and X-Path instances of the child nodes respectively. Note that because we are considering the lowest-level counterexample, $\hat{P}_1$ and $\hat{P}_2$ must have at most w_1 and w_2 hidden colours respectively.

By Observation 3.3.1, P_{Node} has at most $w_1 + w_2 + \sum_{\mathtt{c} \in \mathtt{S}_{\hat{\mathtt{A}}}} (\mathtt{H}(\mathtt{c}, \tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2, \tilde{\mathtt{V}}_{\text{Node}}))$ hidden colours. However, this is equivalent to $\mathtt{M}(w_1, w_2, \tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2, \tilde{\mathtt{V}}_{\text{Node}}) = w_{\text{Node}}$. Thus no counterexample nodes can exist. ∎

Lemma 3.4.4 *For all phases $y \geq 0$ and all X-Path profiles $\tilde{V}$, if $d_y[\tilde{V}]$ is a finite value, there must exist a $\tilde{V}$ instance with at most $d_y[\tilde{V}]$ hidden colours.*

Proof By Lemma 3.4.1 there must be an X-Path valid derivation tree whose root node has the same X-Path profile $\tilde{\mathtt{V}}$ and weight at most $d_y[\tilde{\mathtt{V}}]$. Then by Lemma 3.4.3 the root node of this X-Path valid derivation tree must be associated with a $\tilde{\mathtt{V}}$ instance with at most $d_y[\tilde{\mathtt{V}}]$ hidden colours. ∎

This effectively means that $d_y[\tilde{\mathtt{V}}] \geq W^*(\tilde{\mathtt{V}})$ for all $y \geq 0$ and all $\tilde{\mathtt{V}}$, based on the definition of $W^*(\tilde{\mathtt{V}})$. Thus we have shown that the X-Path Algorithm derives an upper bound on $W^*(\tilde{\mathtt{V}})$ for every X-Path profile $\tilde{\mathtt{V}}$.

As defined in Section 3.3, the X-Path Algorithm outputs $d_Y[\tilde{\mathtt{V}}_{\text{output}}]$, where Y is the terminal phase and $\tilde{\mathtt{V}}_{\text{output}}$ is a 1-Path profile with $\tilde{\mathtt{V}}_{\text{output}}[1] = s'$ and $\tilde{\mathtt{V}}_{\text{output}}[2] = t'$. Then as previously mentioned, because $\mathtt{S}_{s'}$ and $\mathtt{S}_{t'}$ are empty sets, $W^*(\tilde{\mathtt{V}}_{\text{output}}) = W'(\tilde{\mathtt{V}}_{\text{output}}) = W'(s',t')$, so the X-Path Algorithm outputs an upper bound on $W'(s',t')$. Although for the general case it is not clear how tight this upper bound is, the remainder of this thesis will discuss the various settings and values of X for which constant factor approximations can be achieved by the X-Path Algorithm.

Chapter 4

Valid Derivation Tree Construction

In Chapter 3, we introduced the Multi-Path Algorithms for the Minimum-Vertex-Colour Path Problem. Furthermore, we defined the concept of X-Path valid derivation trees and proved Lemma 3.4.1 and Lemma 3.4.2 hold for all Multi-Path Algorithms. Thus for any positive integer $X > 0$, the output of the X-Path Algorithm is equivalent to the minimum root node weight over all X-Path valid derivation trees for the X-Path profile $\tilde{\mathbb{V}}_{\text{output}}$, where $\tilde{\mathbb{V}}_{\text{output}}$ is a 1-Path profile with $\tilde{\mathbb{V}}_{\text{output}}[1] = s'$ and $\tilde{\mathbb{V}}_{\text{output}}[2] = t'$. Note that this means any $s' - t'$ path, treated as an X-Path instance, is a realization of the X-Path profile $\tilde{\mathbb{V}}_{\text{output}}$.

In particular, we showed that the output of the X-Path Algorithm must always be an upper bound on $W'(s',t')$, the minimum over $s' - t'$ paths of the number of distinct colours used. This means all X-Path valid derivation trees for $\tilde{\mathbb{V}}_{\text{output}}$ must also have root node weights which are upper bounds on $W'(s',t')$. In this chapter, we continue our performance analysis of the X-Path Algorithm by investigating the relationship between the structure of an $s' - t'$ path P, the structure of an X-Path valid derivation tree VDT for P, and the weight of the root node of VDT.

4.1 Timeline Representation of Paths

In previous work by Bereg and Kirkpatrick [2], paths in a restricted version of the Barrier Resilience Problem were visually represented as well-structured timelines, and this timeline representation was heavily exploited to achieve some of their results. We note that for the Minimum-Vertex-Colour Path Problem, it is also useful to have timeline representations, and many of the subsequent concepts and proofs involving timeline representations are refinements of similar concepts and proofs suggested by Bereg and Kirkpatrick [2].

Given any $s' - t'$ path P, the timeline representation of P is a timeline for some entity walking along P from s' to t'. For simplicity, we will only consider the case where P is an acyclic $s' - t'$ path, noting the fact that there must always be an acyclic Minimum-Colour $s' - t'$ path in every problem instance. This makes the subsequent analysis much simpler, but we note that many of the arguments could be generalized to work for cyclic paths as well.

The basic structure of the timeline representation of P is a horizontal chain of vertices on a two-dimensional plane, corresponding to the vertices encountered by the entity as it walks along the path P, such that the left endpoint of the chain is the vertex s' and the right endpoint of the chain is the vertex t'. This basic structure can be augmented with information about which vertices share colours via the addition of a set of coloured arcs connecting pairs of vertices with the same colour. We refer to each arc as a *bracket*, and a set of arcs is referred to as a *bracketing*.

Note that sharing a colour is a transitive relation, so a bracketing may connect pair of $\mathtt{c}$-coloured vertices indirectly via a series of $\mathtt{c}$-coloured brackets rather than directly through a single $\mathtt{c}$-coloured bracket. In addition, observe that it is also a reflexive relation, so every $\mathtt{c}$-coloured vertex is trivially connected to itself via a series of 0 brackets. Thus a bracketing is considered to be a *full bracketing* if it contains exactly one spanning tree of $\mathtt{c}$-coloured brackets connecting all of the $\mathtt{c}$-coloured vertices for each colour $\mathtt{c}$, with no additional brackets. Note that by this definition, there may be multiple different full bracketings for P, but they are all *connection-equivalent* in the sense that for every colour $\mathtt{c}$, every pair of $\mathtt{c}$-coloured vertices is connected by $\mathtt{c}$-coloured brackets in one full bracketing if and only if the pair is also connected in the other full bracketings. A *partial bracketing* is then defined as any subset of a full bracketing.

4.2 The Discount Bracketing of a Valid Derivation Tree

In Chapter 3, the performance analysis of the X-Path Algorithm showed that the output of the X-Path Algorithm must always be an upper bound on $W'(s',t')$. The proof of this statement relies on Lemma 3.4.3, which claims that given any node in an X-Path valid derivation tree, if the node has weight w_{Node} and X-Path instance $\hat{P}_{\text{Node}}$, w_{Node} is an upper bound on the number of distinct hidden colours for $\hat{P}_{\text{Node}}$. However, it does not assert anything about how tight this upper bound must be. In effect, although we proved that the output of the X-Path Algorithm must be an upper bound on $W'(s',t')$, we also could not assert anything about how tight this upper bound must be. Thus in order to give a more thorough performance analysis, we would like a stronger claim, one which can define the exact value of w_{Node} rather than simply stating that it is an upper bound on another value.

In this section, we note that this is possible if we assign each node a bracketing. As before, let P be an acyclic $s'-t'$ path, and let VDT be an X-Path valid derivation tree for P. Then we assign each node of VDT a bracketing B_{Node} in ascending order as follows:

- If the node is a leaf node, assign it a bracketing containing no brackets.

- If the node has exactly two child nodes, let $\tilde{\mathrm{V}}_{\text{Node}}$ denote the X-Path profile of the node and let $\tilde{\mathrm{V}}_1$ and $\tilde{\mathrm{V}}_2$ denote the X-Path profiles of its two child nodes. Similarly, let B_1 and B_2 denote the bracketings of the two child nodes.

  First, set B_{Node} to be the union of B_1 and B_2. Then for every colour c, let j be the c-coloured vertex in $\tilde{\mathrm{V}}_{\text{Node}}$ which is earliest in the sense that an entity walking along P from s' to t' would encounter j first. If no such vertex exists, let j be the earliest c-coloured vertex which is disappearing in the sense that it is in $(\tilde{\mathrm{V}}_1 \cup \tilde{\mathrm{V}}_2)$ but not in $\tilde{\mathrm{V}}_{\text{Node}}$. Then for each c-coloured disappearing vertex i, if $i \neq j$, add to B_{Node} a c-coloured bracket with i and j as its endpoints. Note that this means a bracket is added for each c-coloured disappearing vertex unless it is the earliest c-coloured disappearing vertex and there exists no c-coloured vertex in $\tilde{\mathrm{V}}_{\text{Node}}$.

In other words, the following describes how B_{Node} is constructed:

```
$B_{\text{Node}} \leftarrow B_1 \cup B_2$
**for all** $\mathtt{c} \in \mathtt{S}_{\hat{\mathtt{A}}}$ **do**
    $j \leftarrow$ the earliest $\mathtt{c}$-coloured vertex in $\tilde{\mathtt{V}}_{\text{Node}}$
    **if** $j = \text{NULL}$ **then**
        $j \leftarrow$ the earliest $\mathtt{c}$-coloured vertex in $(\tilde{\mathtt{V}}_1 \cup \tilde{\mathtt{V}}_2) - \tilde{\mathtt{V}}_{\text{Node}}$
    **for all** $i \in (\tilde{\mathtt{V}}_1 \cup \tilde{\mathtt{V}}_2) - \tilde{\mathtt{V}}_{\text{Node}}$ **do**
        **if** $i$ is $\mathtt{c}$-coloured **then**
            **if** $i \neq j$ **then**
                $b \leftarrow$ $\mathtt{c}$-coloured bracket with endpoints $i$ and $j$
                $B_{\text{Node}} \leftarrow B_{\text{Node}} \cup \{b\}$
```

Observation 4.2.1 *All brackets in B_{Node} only have endpoints in the set of vertices in $\hat{P}_{\text{Node}}$.*

Proof If this observation is false, consider the lowest counterexample node in VDT. As defined above, it cannot be a leaf node since the bracketing of a leaf node must have no brackets. Thus it must be an internal node with exactly two child nodes. Let B_1 and B_2 be the bracketings of these two child nodes, and observe that since these child nodes are not counterexample nodes, the brackets of B_1 and B_2 can only have endpoints in the set of vertices in $\hat{P}_{\text{Node}}$. Then from the construction procedure of B_{Node} above, it is impossible for any bracket in B_{Node} to have an endpoint not in the set of vertices in $\hat{P}_{\text{Node}}$. ∎

Observation 4.2.2 *B_{Node} does not contain any bracket which has both endpoints in $\tilde{\mathtt{V}}_{\text{Node}}$.*

Proof If this observation is false, consider the lowest counterexample node in VDT. As before, it cannot be a leaf node since leaf node bracketings cannot contain brackets. Thus it must be an internal node with exactly two child nodes. Combining Observation 4.2.1 with the fact that its child nodes must not be counterexample nodes, it follows that no bracket in either of the child node bracketings can have both endpoints in $\tilde{\mathtt{V}}_{\text{Node}}$. Then from the construction procedure of B_{Node} above, it is impossible for any bracket in B_{Node} to have both endpoints in $\tilde{\mathtt{V}}_{\text{Node}}$. ∎

Observation 4.2.3 *Let B_1 and B_2 be the bracketings for a pair of nodes which share a parent node. $B_1 \cap B_2 = \{\}$.*

Proof Follows directly from Observation 4.2.1 and Observation 4.2.2. ∎

Now for any node in VDT with weight w_{Node}, X-Path instance $\hat{P}_{\text{Node}}$, and bracketing B_{Node}, we define the following functions:

$$\text{Total}(\hat{P}_{\text{Node}}, \mathtt{c}) = \text{ Number of } \mathtt{c}\text{-coloured vertices in } \hat{P}_{\text{Node}}$$

$$\text{Exposed}(\hat{P}_{\text{Node}}, \mathtt{c}) = \text{ Number of } \mathtt{c}\text{-coloured path endpoints in } \hat{P}_{\text{Node}}$$

Then for any set of colours $\mathtt{S}$, we will extend these functions as follows:

$$\text{Total}(\hat{P}_{\text{Node}}, \mathtt{S}) = \sum_{\mathtt{c} \in \mathtt{S}} (\text{Total}(\hat{P}_{\text{Node}}, \mathtt{c}))$$

$$\text{Exposed}(\hat{P}_{\text{Node}}, \mathtt{S}) = \sum_{\mathtt{c} \in \mathtt{S}} (\text{Exposed}(\hat{P}_{\text{Node}}, \mathtt{c}))$$

Lemma 4.2.4 $w_{\text{Node}} = \text{Total}(\hat{P}_{\text{Node}}, S_{\hat{\mathtt{A}}}) - \text{Exposed}(\hat{P}_{\text{Node}}, S_{\hat{\mathtt{A}}}) - |B_{\text{Node}}|$.

Proof If this lemma is false, there must exist some counterexample nodes in VDT. Let Node_{low} denote the lowest level counterexample node in VDT, and let w_{low}, $\hat{P}_{\text{low}}$, $\tilde{\mathtt{V}}_{\text{low}}$ and B_{low} denote the weight, X-Path instance, X-Path profile, and bracketing of Node_{low} respectively. Note that because its a counterexample node, $w_{\text{low}} \neq \text{Total}(\hat{P}_{\text{low}}, \mathtt{S}_{\hat{\mathtt{A}}}) - \text{Exposed}(\hat{P}_{\text{low}}, \mathtt{S}_{\hat{\mathtt{A}}}) - |B_{\text{low}}|$.

First, consider the case where Node_{low} is a leaf node. Observe that all vertices used by $\hat{P}_{\text{low}}$ must be path endpoints, so $\text{Total}(\hat{P}_{\text{Node}}, \mathtt{S}_{\hat{\mathtt{A}}}) - \text{Exposed}(\hat{P}_{\text{Node}}, \mathtt{S}_{\hat{\mathtt{A}}})$ must be equal to 0. Furthermore, B_{low} must contain no brackets, so $|B_{\text{Node}}|$ is also equal to 0. Finally, w_{low} must also be 0 when Node_{low} is a leaf node. This means $w_{\text{low}} = \text{Total}(\hat{P}_{\text{low}}, \mathtt{S}_{\hat{\mathtt{A}}}) - \text{Exposed}(\hat{P}_{\text{low}}, \mathtt{S}_{\hat{\mathtt{A}}}) - |B_{\text{low}}|$, so Node_{low} cannot be a leaf node.

Thus Node_{low} must be an internal node with exactly two child nodes which are not counterexample nodes. Let w_1 and w_2 denote the weights, $\hat{P}_1$ and $\hat{P}_2$ denote the X-Path instances, $\tilde{\mathtt{V}}_1$ and $\tilde{\mathtt{V}}_2$ denote the X-Path profiles, and B_1 and B_2 denote the bracketings of the two child nodes respectively. Then as defined in Chapter 3,

$w_{\text{low}} = w_1 + w_2 + \sum_{c \in S_{\hat{A}}} (\mathtt{H}(c, \tilde{V}_1, \tilde{V}_2, \tilde{V}_{\text{low}})) = \sum_{c \in S_{\hat{A}}} (\text{Total}(\hat{P}_1, c) + \text{Total}(\hat{P}_2, c) - \text{Exposed}(\hat{P}_1, c) - \text{Exposed}(\hat{P}_2, c) + \mathtt{H}(c, \tilde{V}_1, \tilde{V}_2, \tilde{V}_{\text{low}})) - |B_1| - |B_2|$.

Next, let $\text{Dis}(c, \tilde{V}_1, \tilde{V}_2, \tilde{V}_{\text{low}})$ denote the number of disappearing c-coloured vertices, such that for any colour c, $\text{Total}(\hat{P}_1, c) + \text{Total}(\hat{P}_2, c) - \text{Exposed}(\hat{P}_1, c) - \text{Exposed}(\hat{P}_2, c) = \text{Total}(\hat{P}_{\text{low}}, c) - \text{Exposed}(\hat{P}_{\text{low}}, c) - \text{Dis}(c, \tilde{V}_1, \tilde{V}_2, \tilde{V}_{\text{low}})$. Then observe that based on the bracketing construction procedure above, when constructing B_{low} for Node_{low}, the number of c-coloured brackets added for disappearing vertices must be equal to $\text{Dis}(c, \tilde{V}_1, \tilde{V}_2, \tilde{V}_{\text{low}}) - \mathtt{H}(c, \tilde{V}_1, \tilde{V}_2, \tilde{V}_{\text{low}})$. In fact, $|B_{\text{low}}| - |B_1| - |B_2| = \text{Dis}(c, \tilde{V}_1, \tilde{V}_2, \tilde{V}_{\text{low}}) - \mathtt{H}(c, \tilde{V}_1, \tilde{V}_2, \tilde{V}_{\text{low}})$ because of Observation 4.2.3 and Observation 4.2.2.

Finally, by substituting these equations into our expression for w_{low}, we derive that $w_{\text{low}} = \sum_{c \in S_{\hat{A}}} (\text{Total}(\hat{P}_{\text{low}}, c) - \text{Exposed}(\hat{P}_{\text{low}}, c)) - |B_{\text{low}}|$, contradicting the fact that Node_{low} denotes a counterexample node. Thus the lemma must be true. ∎

Furthermore, since s' and t' have no colours, $\text{Exposed}(P, S_{\hat{A}}) = 0$. With that in mind, we refer to the bracketing assigned to the root node as VDT's discount bracketing B_{discount}, since the weight of the root node of VDT must be equivalent to $\text{Total}(P, S_{\hat{A}}) - |B_{\text{discount}}|$. Finally, we end this section with the following lemmas that are useful for proving bounds on the size of B_{discount}.

Lemma 4.2.5 *For each colour c and each pair of c-coloured vertices i and j in P, if there exists a pair of X-Path profiles $\tilde{V}_1$ and $\tilde{V}_2$ associated with any pair of nodes which share a parent in* VDT *such that $\tilde{V}_1 \cup \tilde{V}_2$ contains both i and j, then i and j are connected by c-coloured brackets in B_{discount}.*

Proof We first show that for each colour c and each pair of c-coloured vertices i and j in P, if there exists an X-Path profile $\tilde{V}$ associated with any node in VDT such that $\tilde{V}$ contains both i and j, then i and j are connected by c-coloured brackets in B_{discount}. This is proved via contradiction as follows:

Consider the highest node $\text{Node}_{\text{high}}$ in VDT, where $\text{Node}_{\text{high}}$ is a counterexample node in the sense that its X-Path profile $\tilde{V}_{\text{high}}$ contains a pair of c-coloured vertices i and j, but i and j are not connected by c-coloured brackets in B_{discount}. Because s' and t' have no colours assigned to them, $\text{Node}_{\text{high}}$ cannot be the root node. Thus $\text{Node}_{\text{high}}$ must have a parent node which is not a counterexample node. This means either i or j must not be present in $\tilde{\mathbb{V}}_{\text{parent}}$, the X-Path profile of the parent node. In other words, i or j must be a disappearing vertex.

However, by the discount bracketing construction procedure, there must be a $\mathtt{c}$-coloured bracket added to B_{discount} that connects each $\mathtt{c}$-coloured disappearing vertex to either the first $\mathtt{c}$-coloured vertex in $\tilde{\mathbb{V}}_{\text{parent}}$ or the $\mathtt{c}$-coloured disappearing vertex closest to the start of P. If they are connected to the first $\mathtt{c}$-coloured vertex in $\tilde{\mathbb{V}}_{\text{parent}}$, by transitivity and the fact that the parent node is not a counterexample node it must be the case that i and j are connected by $\mathtt{c}$-coloured brackets in B_{discount}. Otherwise, i and j must both be disappearing vertices, and observe that they must still be connected by $\mathtt{c}$-coloured brackets in B_{discount} in this case. As a result no counterexample node can exist.

Thus for each colour $\mathtt{c}$ and each pair of $\mathtt{c}$-coloured vertices i and j in P, if there exists an X-Path profile $\tilde{\mathbb{V}}$ associated with any node in VDT such that $\tilde{\mathbb{V}}$ contains both i and j, then i and j are connected by $\mathtt{c}$-coloured brackets in B_{discount}. Then consider the pair of X-Path profiles $\tilde{\mathbb{V}}_1$ and $\tilde{\mathbb{V}}_2$ described in the lemma. The lemma immediately follows from applying this result on their nodes and the shared parent node, and the fact that if both i and j are disappearing vertices for the parent node, $\mathtt{c}$-coloured brackets would be added that connect i and j. ∎

4.3 Pre-emptive Detection Approach

Let P be an acyclic $s' - t'$ path and let B^* be a partial bracketing on P. We would like to prove whether there exists an X-Path Algorithm valid derivation tree VDT for P such that the root node of VDT has weight at most $\text{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B^*|$. If this can be achieved for every bracketing B^*, then the performance analysis of the X-Path Algorithm would be much simpler given Lemma 3.4.2 and Lemma 3.4.1.

In this section, for any positive integer $X \geq 2$, we present the Pre-Emptive Detection Approach for constructing an X-Path Algorithm valid derivation tree VDT_{PDA} for P such that the root node of VDT_{PDA} has weight at most $\text{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B^*|$ when given a partial bracketing B^* on P. This approach uses the following construction procedure:

1. For each edge in P, create a level-0 leaf node with X-Path profile set such

that the edge is a realization of the X-Path profile. Note that this X-Path profile by definition would also be a 1-Path profile.

2. For each bracket in B^*, let i and j be the vertices at the endpoints of the bracket. Create a level-0 leaf node with X-Path profile set to the vertex 4-tuple (i,i,j,j). Note that by definition, this X-Path profile is also a 2-Path profile.

3. For each pair of parentless nodes whose X-Path profiles are 1-Path profiles, if there exists a vertex $\hat{u}$ such that the following constraints hold:

   - The vertex 2-tuple of one of the 1-Path profiles starts with $\hat{u}$, while the other ends with $\hat{u}$.
   - $\hat{u}$ is not the endpoint of any bracket in B^*.

   create a node which is the parent of both nodes, and assign it the 1-Path instance which combines the 1-Path instances of both nodes such that $\hat{u}$ is no longer a path endpoint. Then assign it the 1-Path profile that is realized by this 1-Path instance.

   Repeat this step until there are no applicable pairs of nodes. Note that the order in which pairs of nodes are considered does not matter. Ultimately, for each subpath P_{sub} of P, if P_{sub} starts and ends at bracket endpoints of B^* and does not contain any other bracket endpoints of B^*, a unique node must have been created such that its X-Path profile is a 1-Path profile that is realized by the 1-Path instance P_{sub}.

4. Finally, find some method of combining the X-Path instances of the remaining parentless nodes to complete VDT_{PDA} such that the root node's X-Path instance is the 1-Path instance P. Note that this must be done without violating the constraints on valid derivation trees described in Section 3.4. In particular, note that this means we cannot create any additional leaf nodes whose X-Path instances contain edges.

Note that while the first three steps can be carried out for any bracketing, the last step will fail on some bracketings. With this in mind, we define a set of bracketings called *X-tree-admissible bracketings*, which includes a partial bracketing B^* if and only if the construction procedure of the Pre-emptive Detection Approach outlined above succeeds in constructing an X-Path valid derivation tree with respect to B^*.

Lemma 4.3.1 *If B^* is an X-tree-admissible-bracketing, the weight of the root node of* VDT_{PDA} *is at most* $\text{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B^*|$.

Proof For any colour $\mathtt{c}$ and any $\mathtt{c}$-coloured bracket b in B^*, observe that both its endpoints are in the X-Path profile of the node created for it in the second step. Thus by Lemma 4.2.5, the discount bracketing of VDT_{PDA} must connect its endpoints via $\mathtt{c}$-coloured brackets. Effectively, the discount bracketing connects everything that is connected by B^*, which is only possible if the discount bracketing is at least as large as B^* since B^* is a partial bracketing. Then by Lemma 4.2.4, the root node weight must be at most $\text{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B^*|$.

Thus in some sense, by the second step of the construction procedure we have already pre-emptively detected that the root node weight must be at most $\text{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B^*|$, before the root node is even created. Note that by Lemma 3.4.2, the X-Path Algorithm cannot perform worse than VDT_{PDA}, so its output must also be at most $\text{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B^*|$.

However, given any partial bracketing B^*, it is difficult to directly determine whether it is an X-tree-admissible bracketing. Instead, in this section we will begin by defining a set of bracketings and then prove that these bracketings are 2-tree-admissible bracketings via induction. Then in later chapters, these 2-tree-admissible bracketings will be used in some proofs involving constant factor approximations achieved by the 2-Path Algorithm.

First, let a bracket b_{empty} be considered an *empty bracket* with respect to a partial bracketing B^* if no bracket in B^* has an endpoint that lies in between the endpoints of b_{empty}. Then let a bracket b_{DA1} be considered *doubly-adjacent* to a bracket b_{DA2} in a partial bracketing B_* if there exists two distinct subpaths of P such that:

- Each subpath either starts at an endpoint of b_{DA1} and ends an endpoint of b_{DA2}, or starts at an endpoint of b_{DA2} and ends at an endpoint of b_{DA1}.

- Each endpoint of b_{DA1} is either the start or end of one subpath.
- No bracket in B^* has an endpoint that lies in within each subpath.

Next, given any full bracketing B_{full} on P, there exists $2^{|B_{\text{full}}|}$ partial bracketings which are subset of B_{full}. Out of these partial bracketings, we inductively define a set $\hat{B}$ of bracketings as follows:

1. The partial bracketing composed of no brackets is in $\hat{B}$.
2. Let b_{empty} be an empty bracket with respect to a partial bracketing B_{induct}. If B_{induct} is in $\hat{B}$, $B_{\text{induct}} \cup \{b_{\text{empty}}\}$ is also in $\hat{B}$.
3. Let b_{DA1} be a bracket which is doubly-adjacent to some bracket b_{DA2} in a partial bracketing B_{induct}. If B_{induct} is in $\hat{B}$, $B_{\text{induct}} \cup \{b_{DA1}\}$ is also in $\hat{B}$.
4. $\hat{B}$ is the smallest set which satisfies the above.

We will prove that all bracketings in $\hat{B}$ are 2-tree-admissible bracketings. For simplicity, the following arguments are presented for the simple case where there are no coincident bracket endpoints in B_{full}. In other words, there is no vertex on P that is used as the endpoint of more than one bracket in B_{full}. However, we note that they can be generalized to apply for all full bracketings by specifying an arbitrary ordering for any coincident bracket endpoints, so given any two bracket endpoints one must be strictly on one side of the other. We begin with the base case:

Lemma 4.3.2 *If B_{induct} contains no brackets, B_{induct} is a 2-tree-admissible bracketing.*

Proof If B_{induct} contains no brackets, then the second step in the construction procedure of the Pre-emptive Detection Approach would create no nodes. Then observe that the third step would already complete the tree. Thus B_{induct} must be a 2-tree-admissible bracketing. ∎

Lemma 4.3.3 *If B_{induct} is a 2-tree-admissible bracketing, $B_{\text{induct}} \cup \{b_{\text{empty}}\}$ must be a 2-tree-admissible bracketing as well.*

Proof Since B_{induct} is a 2-tree-admissible bracketing, the last step of the construction procedure of the Pre-emptive Detection Approach outlined above must have succeeded and produced a 2-Path Algorithm valid derivation tree $\text{VDT}_{B_{\text{induct}}}$.

Let P_{empty} be the subpath of P that starts from one endpoint of b_{empty} and ends at the other endpoint of b_{empty}. Since b_{empty} is an empty bracket with respect to B_{induct}, there must be no bracket endpoints of B_{induct} that lie within P_{empty}. Then observe that in the construction procedure outlined above, P_{empty} must have a set of level-0 nodes representing its edges created for $\text{VDT}_{B_{\text{induct}}}$ in the first step, and these edges must have been merged into a single path P_{induct} that has P_{empty} as a subpath by the third step. By construction, P_{induct} must have its endpoints either at some bracket endpoints of B_{induct} or at s' or t', and in $\text{VDT}_{B_{\text{induct}}}$ there must be a node $\text{Node}_{\text{induct}}$ whose X-Path instance is P_{induct}.

Consider applying the Pre-emptive Detection Approach to construct a tree for $B_{\text{induct}} \cup \{b_{\text{empty}}\}$. After the third step of the construction procedure outlined above, in place of $\text{Node}_{\text{induct}}$ there must be instead:

- A node with an X-Path instance composed of a length-0 path at each endpoint of b_{empty}.
- A node with an X-Path instance composed of a single path from the start of P_{induct} to the start of P_{empty}.
- A node with an X-Path instance composed of a single path P_{empty}
- A node with an X-Path instance composed of a single path from the end of P_{empty} to the end of P_{induct}.

Observe that given the rules on X-Path instance combinations defined in Section 3.1, the X-Path instances associated with these four nodes can be combined sequentially to form a new node $\text{Node}_{\text{empty}}$ whose X-Path Instance is P_{induct}. Then observe that the only difference between the resulting set of parentless nodes and the set of parentless nodes after the third step of the construction procedure applied for constructing $\text{VDT}_{B_{\text{induct}}}$ is that $\text{Node}_{\text{induct}}$ has been replaced by $\text{Node}_{\text{empty}}$, but both nodes share the same X-Path instance $\hat{P}_{\text{induct}}$ and consequently the same X-Path profile.

Then since the last step must have succeeded for the latter set of parentless nodes, it must clearly succeed for the former set as well since the X-Path instance of every corresponding parentless node is identical. Thus we have shown that $B_{\text{induct}} \cup \{b_{\text{empty}}\}$ must be a 2-tree-admissible bracketing as well. ▮

Lemma 4.3.4 *If B_{induct} is a 2-tree-admissible bracketing, then $B_{\text{induct}} \cup \{b_{DA1}\}$ must be a 2-tree-admissible bracketing as well.*

Proof As before, since B_{induct} is a 2-tree-admissible bracketing, the last step of the construction procedure of the Pre-emptive Detection Approach must have succeeded and produced a 2-Path Algorithm valid derivation tree $\text{VDT}_{B_{\text{induct}}}$.

Since b_{DA2} is in B^*, the second step of the construction procedure must have created a node Node_{DA2} whose X-Path instance is a 2-Path instance composed of a length-0 path at each endpoint of b_{DA2}.

Consider applying the Pre-emptive Detection Approach to construct a tree for $B_{\text{induct}} \cup \{b_{DA1}\}$. After the second step of the construction procedure outlined above, the only difference to the set of parentless nodes would be the addition of another node whose X-Path instance is a 2-Path instance composed of a length-0 path at each endpoint of b_{DA1}. Then, because b_{DA1} is doubly-adjacent to b_{DA2}, observe that by definition the X-Path instances of the two nodes for these two brackets can be combined with the 1-Path instances representing some edges of P for some leaf nodes to form a node Node_{DA} whose X-Path instance is a 2-Path instance composed of two paths, each containing an endpoint of b_{DA2}.

If we perform the third step after Node_{DA} is created, observe that there is a one-to-one mapping from the resulting set of parentless nodes to the set of parentless nodes after the third step of the construction procedure applied for constructing $\text{VDT}_{B_{\text{induct}}}$, such that the only difference is that the absolute positions and sizes of some X-Path instances have changed. However, which path endpoints are shared by each pair of X-Path instances remains exactly the same. Thus as before, we observe that the last step of the construction procedure for the Pre-emptive Detection Approach must succeed for the former set as well. Thus we have shown that $B_{\text{induct}} \cup \{b_{DA1}\}$ must be a 2-tree-admissible bracketing as well. ▮

Thus by induction, all bracketings in $\hat{B}$ are 2-tree-admissible bracketings.

4.4 Sweep Span-Exploiting Approach

In this section, we present an alternative approach to constructing X-Path valid derivation trees that is useful for some proofs later on in this thesis. Note that as before, the goal is to construct an X-Path Algorithm valid derivation tree $\mathrm{VDT}_{\mathrm{SSA}}$ for an acyclic $s' - t'$ path P such that the root node of $\mathrm{VDT}_{\mathrm{SSA}}$ has weight at most $\mathrm{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B^*|$ when given a partial bracketing B^* on P. However, unlike the previous section where we began by defining a construction procedure and then proceeded to find bracketings for which the procedure would succeed, in this section we will first define a target set of bracketings, then describe the construction procedure of the Sweep Span-Exploiting Approach and prove that it always succeeds for the target bracketings.

To describe this target set of bracketings, we begin by defining the *bracket-span* of a colour $\mathtt{c}$ with respect to a bracketing B on P as the set of all vertices on P that are between the endpoints of at least one $\mathtt{c}$-coloured bracket in B, including the vertices which are at the endpoints of $\mathtt{c}$-coloured brackets. Then every partial bracketing B^* on P is a target bracketing if and only if every vertex on P is in the bracket-spans of at most X distinct colours with respect to B^*.

Given any target bracketing B^*, the gist of the Sweep Span-Exploiting Approach is to recursively combine the leftmost remaining edge of P into an X-Path instance, in some sense sweeping across the timeline from left to right until it reaches t'. Note that this means the X-Path instance at any point in this recursion must always be composed of X consecutive paths with no edges in between them, in the sense that the vertex at the end of each path is the start of the next path. This in turn means that the number of distinct vertices which are endpoints of the X-Path instance at any point in the recursion is at most $X+1$. This sweep approach is reflected in the valid derivation tree by a long chain of nodes, each the parent of the last, with each node representing a point in the recursion. This is more formally described as follows:

First create a set of level-0 leaf nodes for every edge in P as before, which is needed to ensure that the root node can have P as its X-Path instance. Next, the start of the long chain of nodes is created as a level-0 leaf node whose X-Path instance is composed of X length-0 paths from s' to s'. Then to describe the procedure used to extend the chain in each step of the recursion, let Node_{end} denote the node at the current end of the chain, and let $\tilde{\mathtt{V}}_{\text{end}}$ and $\hat{P}_{\text{end}}$ denote its X-Path profile and X-Path instance respectively. Then let P_1 to P_X denote the X paths of $\hat{P}_{\text{end}}$ in sequential order. As previously mentioned, we shall maintain the invariant that these X paths are consecutive with no edges in between them. In other words, for any integer $1 \leq x < X$, the end of P_x must be the same vertex as the start of P_{x+1}, and $\tilde{\mathtt{V}}_{\text{end}}[2x] = \tilde{\mathtt{V}}_{\text{end}}[2x+1]$. Furthermore, $\tilde{\mathtt{V}}_{\text{end}}[1]$ must always be s'. In light of this fact, let $\mathtt{S}_{P_x}$ denote the set of colours assigned to the vertex at the end of P_x, such that $\mathtt{S}_{P_x} = \mathtt{S}_{\tilde{\mathtt{V}}_{\text{end}}[2x]}$.

Next, let P_{X+1} denote the leftmost remaining edge of P not within $\hat{P}_{\text{end}}$, and similarly let $\mathtt{S}_{P_{X+1}}$ denote the set of colours assigned to the vertex at the end of P_{X+1}. Note that we must have created a level-0 leaf node $\text{Node}_{\text{leaf}}$ whose X-Path instance is the 1-Path instance P_{X+1}. The next node in the chain is constructed such that its X-Path instance is formed via the combination of the X-Path instances from Node_{end} and $\text{Node}_{\text{leaf}}$. Observe that this combination can be performed with no problems because the end of P_X must be the same vertex as the start of P_{X+1}. In fact, there even exists multiple choices for the resulting X-Path instance, since any two consecutive paths in the $X+1$ paths P_1 to P_{X+1} could be concatenated to form an X-Path instance.

Let $\text{BS}(\mathtt{c})$ be true if the end of P_{X+1} is within the bracket-span of a colour $\mathtt{c}$. We shall choose the X-Path instance that results from concatenating P_x and P_{x+1} for the smallest positive integer x where each colour assigned to the end of P_x is either assigned to the end of another path, or the end of the chain is no longer within the bracket-span of the colour. More formally, this is the smallest positive integer x for which $(\mathtt{c} \in \mathtt{S}_{P_x}) \to (\neg\text{BS}(\mathtt{c}) \vee \exists_{1 \leq j \leq X+1}((x \neq j) \wedge \mathtt{c} \in \mathtt{S}_{P_j}))$. We observe that such a path must always exist given our precondition that every point on P is known to be within the bracket-spans of at most X distinct colours with respect to the bracketing B^*. This is because any counterexample would require every one of the $X+1$ paths to be assigned a unique colour whose bracket-span covers the end of the chain, which is not possible when every point on P is within the bracket-spans of at most X distinct colours.

Then for every colour $\mathtt{c}$, consider the bracket-span of $\mathtt{c}$ with respect to B^*. As defined above, the bracket-span is the set of vertices in P that are within at least one $\mathtt{c}$-coloured bracket, and B^* does not need to be a full bracketing. As a result, the bracket-span may be composed of multiple disjoint intervals.

Observation 4.4.1 *If the end of P_{X+1} is the start of one of the disjoint intervals that form the bracket-span of $\mathtt{c}$, the end of P_{X+1} must be assigned the colour $\mathtt{c}$.*

Proof As defined above, the bracket-span of $\mathtt{c}$ includes the vertices at the endpoints of $\mathtt{c}$-coloured brackets. The start of each of these i disjoint intervals must be an endpoint of a $\mathtt{c}$-coloured bracket, and $\mathtt{c}$-coloured brackets must have $\mathtt{c}$-coloured vertices as their endpoints. ∎

Observation 4.4.2 *If the end of P_{X+1} is within one of the disjoint intervals that form the bracket-span of $\mathtt{c}$ but not the start of the interval, there must exist a positive integer $x' \leq X$ such that the end of $P_{x'}$ is assigned the colour $\mathtt{c}$.*

Proof By Observation 4.4.1, when the sweep reached the start of this interval, the X-Path instance constructed must have a path which has $\mathtt{c}$ assigned to its end. As stated above, for the new node in the chain we choose the X-Path instance that results from concatenating P_x and P_{x+1} for the smallest positive integer x where each colour assigned to the end of P_x is either assigned to the end of another path, or the sweep is no longer within the bracket-span of the colour. Since the sweep could not have left the bracket-span of $\mathtt{c}$ from the start of this interval to its current position, observe that whenever we choose a path P_x with $\mathtt{c}$ assigned to its end, there must be another path in the new X-Path instance which also has $\mathtt{c}$ assigned to its end. Thus when sweep reaches the current position, there must exist a positive integer $x' \leq X$ such that the end of $P_{x'}$ is assigned the colour $\mathtt{c}$. ∎

Finally, when the last edge is merged into the sweep chain, we create another leaf node with an X-Path instance which is a 1-Path instance composed of a single 0-length path at t'. We then create the root node with an X-Path instance which is the 1-Path instance formed via the combination of the leaf node's 1-Path instance and the X-Path instance from the end of the sweep chain. Note that because of the leaf nodes we created in this procedure, this 1-Path instance must be the $s' - t'$ path P.

Now for every colour $\mathtt{c}$, let Intervals$(\mathtt{c})$ denote the number of disjoint intervals that compose the bracket-span of $\mathtt{c}$, and let Unconnected$(\mathtt{c})$ denote the number of $\mathtt{c}$-coloured vertices in P which are not used as an endpoint of any $\mathtt{c}$-coloured bracket in B^*. Then B^*, being a partial bracketing, must have at least Intervals$(\mathtt{c})+$ Unconnected$(\mathtt{c})-1$ fewer $\mathtt{c}$-coloured brackets than any full bracketing, since every full bracketing by definition contains a spanning tree of $\mathtt{c}$-coloured brackets connecting all $\mathtt{c}$-coloured vertices. Next, consider the $\mathtt{c}$-coloured brackets in the discount bracketing B_{discount} of the X-Path valid derivation tree constructed by this approach.

Lemma 4.4.3 *Let i be a $\mathtt{c}$-coloured vertex that is within one of the disjoint intervals that form the bracket-span of $\mathtt{c}$ but not the start of the interval. B_{discount} must contain $\mathtt{c}$-coloured brackets which connect i to a preceding $\mathtt{c}$-coloured vertex.*

Proof By Observation 4.4.2, when the sweep reaches i in the sense that i is the end of P_{X+1}, there must exist a positive integer $x' \leq X$ such that the end of $P_{x'}$ is assigned the colour $\mathtt{c}$. Then by Lemma 4.2.5, B_{discount} must contain $\mathtt{c}$-coloured brackets which connect i to the end of $P_{x'}$, which by definition is a preceding $\mathtt{c}$-coloured vertex. ∎

Lemma 4.4.4 *For each of the disjoint intervals composing the bracket-span of $\mathtt{c}$, all $\mathtt{c}$-coloured vertices within the interval, including the start and end of the interval, are connected by $\mathtt{c}$-coloured brackets in B_{discount}.*

Proof Follows immediately from Lemma 4.4.3 and the fact that being connected by $\mathtt{c}$-coloured brackets in B_{discount} is a transitive relation. ∎

This means B_{discount} has at most Intervals$(\mathtt{c})+$Unconnected$(\mathtt{c})-1$ fewer $\mathtt{c}$-coloured brackets than any full bracketing. As a result, B_{discount} must have at least as many $\mathtt{c}$-coloured brackets as B^* for every colour $\mathtt{c}$, meaning B_{discount} has at least as many brackets as B^*. This allows us to state the main result of this section:

As defined at the start of this section, every partial bracketing B^* on P is a target bracketing if and only if every vertex on P is in the bracket-spans of at most X distinct colours with respect to B^*.

Lemma 4.4.5 *For any target bracketing B^*, the Sweep Span-Exploiting Approach constructs an X-Path Algorithm valid derivation tree for P has root node weight at most* $\mathrm{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B^*|$.

Proof As noted in Section 4.2, the weight must be $\mathrm{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B_{\mathrm{discount}}|$. We just showed that B_{discount} has at least as many brackets as B^*, so $|B_{\mathrm{discount}}| \geq |B^*|$ and it immediately follows that the weight is at most $\mathrm{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B^*|$. ∎

Chapter 5

Planar Bracketing Construction

In the previous chapter, we showed that if a partial bracketing B^* on an acyclic $s' - t'$ path P has certain properties, we can prove via tree construction approaches and Lemma 3.4.2 that the output of X-Path Algorithm is at most $\text{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B^*|$. Thus one way to guarantee the output is a close approximation is by considering an acyclic Minimum Colour $s' - t'$ path P, and proving the existence of an almost full bracketing B^* on P which has the properties required for one of the tree construction approaches to succeed.

Unfortunately, while an acyclic Minimum-Colour $s' - t'$ path P can always be guaranteed to exist, it is not clear whether such a bracketing would exist in general problem instances of the Minimum-Vertex-Colour Path Problem. Nevertheless, as noted in Chapter 2, problem instances of the Minimum-Vertex-Colour Path Problem that result from the reduction from the Barrier Resilience Problem actually have additional structural constraints on the graph. In this chapter, we show that these structural constraints can actually be exploited to construct well-structured full bracketings. In particular, given any acyclic Minimum-Colour $s' - t'$ path P, we can construct a full bracketing which is considered *planar* in the sense that all the brackets in the bracketing can be drawn in the timeline representation without any two brackets intersecting and without any bracket intersecting the horizontal chain of vertices. These planar full bracketings are then exploited in later chapters to prove that the Multi-Path Algorithms guarantee constant factor approximations for the Barrier Resilience Problem.

5.1 Sensor Intersection Orientations

As noted in Chapter 2, when using the reduction from the Barrier Resilience Problem to the Minimum-Vertex-Colour Path Problem, every $s' - t'$ path on the graph $\hat{\mathtt{A}}$ corresponds to a $S - T$ path through $\mathtt{A}$. In addition, observe that like the other optimal path problems discussed in this thesis, there must always exist an acyclic $S - T$ path P which is *resilience-optimal*, in the sense that the number of distinct sensors intersected by P is exactly the resilience of the barrier, and P would correspond to an acyclic Minimum-Colour $s' - t'$ path on the graph $\hat{\mathtt{A}}$. As before, we shall restrict our attention to these acyclic resilience-optimal paths, as they exhibit additional properties which we can exploit.

Before we describe the procedure for constructing a planar full bracketing on P, we shall first assign sensor intersections of *P relative orientations*, describing their relationship with other intersections of the same sensor. These relative orientations are used to guide the construction procedure described in the later sections. Note that because the reduction to the Minimum-Vertex-Colour Path Problem removes every sensor that covers s' or t', we will only assign these relative orientations to intersections of sensors which do not cover the start or the end of P. This means every sensor intersection must have a unique entry point and a unique exit point on the boundary of the associated sensor.

Let a *sensor-boundary-walk* be defined as a walk on the boundary of a sensor. Each intersection of the sensor is considered to be associated with exactly two sensor-boundary-walks. Each of these two sensor-boundary-walks starts from the entry point of the intersection and ends at the exit point, but one sensor-boundary-walk goes around the sensor in a clockwise manner and while the other goes around in a counter-clockwise manner. In particular, each of these sensor-boundary-walk is considered to be *path-intersected* if any other sensor intersection has its entry or exit point within the sensor-boundary-walk.

Each sensor intersection is assigned a relative orientation based on whether its two sensor-boundary-walks are path-intersected. If both sensor-boundary-walks are path-intersected, the sensor intersection is considered to be an *interior* intersection and is assigned a *dual* orientation. Otherwise, it is considered a *peripheral* intersection. Then for each peripheral intersection, if both sensor-boundary-walks are not path-intersected, it is assigned a *null* orientation. Otherwise, it must have exactly one path-intersected sensor-boundary-walk, and a *positive* orientation is assigned if the counter-clockwise sensor-boundary-walk is path-intersected, while a *negative* orientation is assigned if the clockwise sensor-boundary-walk is path-intersected.

In particular, we observe that an acyclic resilience-optimal $S-T$ path which makes only peripheral intersections, henceforth referred to as a *peripheral traversal*, is significantly easier to handle than an acyclic resilience-optimal $S-T$ path which makes even a single interior intersection, henceforth referred to as an *interior traversal*. Thus the next few sections will describe how a planar full bracketing is constructed for a peripheral traversal, and we leave the details on how a planar full bracketing can be constructed for an interior traversal for Section 5.5.

5.2 Forbidden Configurations for Peripheral Traversals

In this section, we describe the additional constraints on peripheral traversals which make the planar full bracketings easier to construct. Begin by considering an arbitrary peripheral traversal P. The folded timeline of P can be thought of as a sequence of vertices representing the faces intersected by P from S to T. For any sensor s_i that is intersected by P more than once, any vertex in this sequence is in exactly one of three distinct states. Either it is not inside s_i, or it is within a positive intersection of s_i, or it is within a negative intersection of s_i. Note that it cannot be within an intersection of s_i which has null or dual orientation because it is a multiply-intersected sensor and P is a peripheral traversal.

Then for any pair of distinct multiply-intersected sensors, each vertex can be described as being in exactly one of nine distinct states with respect to these two sensors. Any sequence of vertices can then be thought of as a sequence of states. Let a sequence of such states be called a *timeline configuration*, and let us define a timeline U as containing a timeline configuration if there exists two distinct multiply-intersected sensors associated with the timeline U such that the sequence of states represented by the timeline configuration is a subsequence of the sequence of states associated with the timeline U with respect to these two sensors.

From this definition, we observe that certain timeline configurations can never be contained by a peripheral traversal through an arrangement of disk sensors. In fact, we will argue that this is true for any arrangement of arbitrarily-shaped sensors as long as the following axioms Axiom 5.2.1 and Axiom 5.2.2 hold.

Axiom 5.2.1 *Each sensor in the arrangement defines a single detection region on the two dimensional plane that contains no holes. In other words, for any two points p_1 and p_2 and any sensor s_i, if the points are both inside or both outside s_i, there exists a path from p_1 to p_2 that does not intersect the boundary of s_i.*

For any pair of sensors s_i and s_j, let an *intersection region* $\mathtt{I}_{i,j}$ be defined as a region on the two dimensional plane that satisfies the following constraints:

- $\mathtt{I}_{i,j}$ is a subregion of the intersection of s_i and s_j.
- $\mathtt{I}_{i,j}$ contains no holes.
- $\mathtt{I}_{i,j}$ is bounded only by the boundaries of s_i and s_j.

Let $\mathtt{P}_i$ be the set of points on the boundary of $\mathtt{I}_{i,j}$ that are also on the boundary of s_i. Let $\mathtt{P}_j$ be the corresponding set for s_j. We say that s_i is *boundary-isolated* with respect to $\mathtt{I}_{i,j}$ if for any two points p_1 and p_2 inside the set $(\mathtt{P}_i - \mathtt{P}_j)$, it is possible to walk from p_1 to p_2 using only points in $\mathtt{P}_i$. In other words, there exists a path from p_1 to p_2 that contains only points which are on the boundaries of both s_i and $\mathtt{I}_{i,j}$ simultaneously. s_i is considered *fully boundary-isolated* with respect to s_j if it is boundary-isolated for every intersection region. Note that if no such region exists, such as when s_i and s_j are fully disjoint, we will still consider s_i to be fully boundary-isolated with respect to s_j. Then if both s_i and s_j are fully boundary-isolated with respect to each other, s_i and s_j are considered to have a *boundary-segregated intersection*.

Axiom 5.2.2 *The arrangement is pairwise boundary-segregated, in the sense that every pair of sensors has a boundary-segregated intersection.*

Observe that any arrangement of disk sensors clearly satisfies both axioms. Furthermore, Axiom 5.2.1 allows us to keep the same definitions for sensor intersection orientations. Let a timeline configuration be considered *realizable* if there exists a timeline U that both contains the timeline configuration and represents a peripheral traversal through an arrangement subject to Axiom 5.2.1 and Axiom 5.2.2. It is considered *unrealizable* or *impossible* otherwise. Then there exists three different transformation methods on timeline configurations that preserve realizability. These transformation methods are:

Mirror Image If the mirror image of the arrangement is examined, clearly if the peripheral traversal associated with U existed in the original arrangement, the traversal still exists for this new arrangement. In the new configuration associated with the traversal, the order of points and sensors remains the same, but all orientations are reversed.

Label Swap If we swap the labels of the two sensors, clearly if there existed the peripheral traversal associated with U existed in the original arrangement, the traversal still exists for the new arrangement. In the new configuration associated with this path, the order of points and orientations remains the same, but all points are covered by the opposite sensor instead.

Reverse Path If the peripheral traversal associated with U is reversed, clearly the traversal still exists and must still enter the same set of sensors the same number of times. In the new configuration associated with this path, all points are still covered by the same sensors as before, but the order of points and all orientations are reversed.

Observe that this definition also means that if a timeline configuration is not realizable, no supersequence of it can be realizable either, since the timeline U that contains such a realizable supersequence must clearly contain the original timeline configuration as well.

In this section, we examine ten timeline configurations, which are considered distinct in the sense that no timeline configuration is a supersequence or transformation of another. These configurations are shown in Figure 5.1. Each black line represents a subchain of vertices of a timeline. The colours red and blue represent two distinct sensors, and coloured shorter lines are drawn above a vertex to show that it is within a positive intersection of a sensor associated with the colour, and

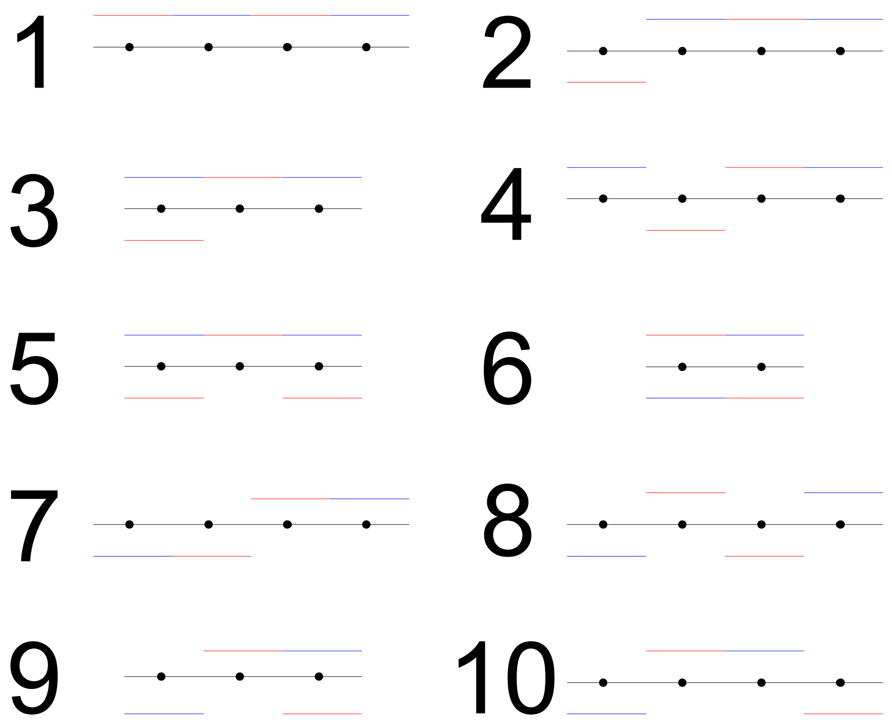

Figure 5.1: 10 Forbidden Configurations

drawn below for negative intersections. In each of these ten timeline subchains, a timeline configuration is formed by the states of the vertices.

We shall define Forbidden Configurations as every timeline configuration that is a result of some sequence of transformations from these ten. As their name implies:

Theorem 5.2.3 *Forbidden Configurations are unrealizable.*

Proof See Section A.1. ∎

5.3 Timeline Unfolding for Peripheral Traversals

The planar full bracketing construction procedure begins with a process we refer to as *timeline unfolding*. Given any peripheral traversal P, the timeline representation of P described in Section 4.1 is referred to as the *folded timeline* of P. In particular, given any vertex i on the folded timeline of P, if i is $\mathtt{c}$-coloured for some colour $\mathtt{c}$, i must be within an intersection of the sensor which corresponds to the colour $\mathtt{c}$. Thus we assign each colour of i the orientation of the associated sensor intersection. The timeline unfolding process uses the folded timeline to create an *unfolded timeline* using these orientations:

1. For each vertex i other than s' in the horizontal chain of vertices from s' to t' in the folded timeline, create a vertex labelled $-i$, and split the colours assigned to i between i and $-i$ such that all colours with negative orientations are transferred to $-i$.

2. Connect all of the vertices to form a horizontal chain of vertices from $-t'$ to s' to t' such that:

   - $-t'$ is on the left end of the chain.
   - s' is in the exact center of the chain.
   - t' is on the right end of the chain.
   - All negative vertices are to the left of s'.
   - The chain is palindromic in the sense that the sequence of vertices encountered from $-t'$ to s' must be encountered in reverse order from s' to t'.

Much like the folded timeline, the unfolded timeline can be augmented with brackets connected vertices of the same colour. In addition, this timeline unfolding process is fully reversible, in the sense that unfolded timelines can be refolded to form folded timelines. In fact, this can be performed even after the addition of brackets, since bracket endpoints can be transferred to and from negative vertices along with their colours.

However, for the unfolded timeline, we will add another constraint on bracketings: no part of any bracket may be drawn below the horizontal chain of vertices. Thus a planar full bracketing on an unfolded timeline is a full bracketing where every bracket can be drawn above the horizontal chain such that no two brackets intersect and no bracket intersects the horizontal chain. The motivation for adding

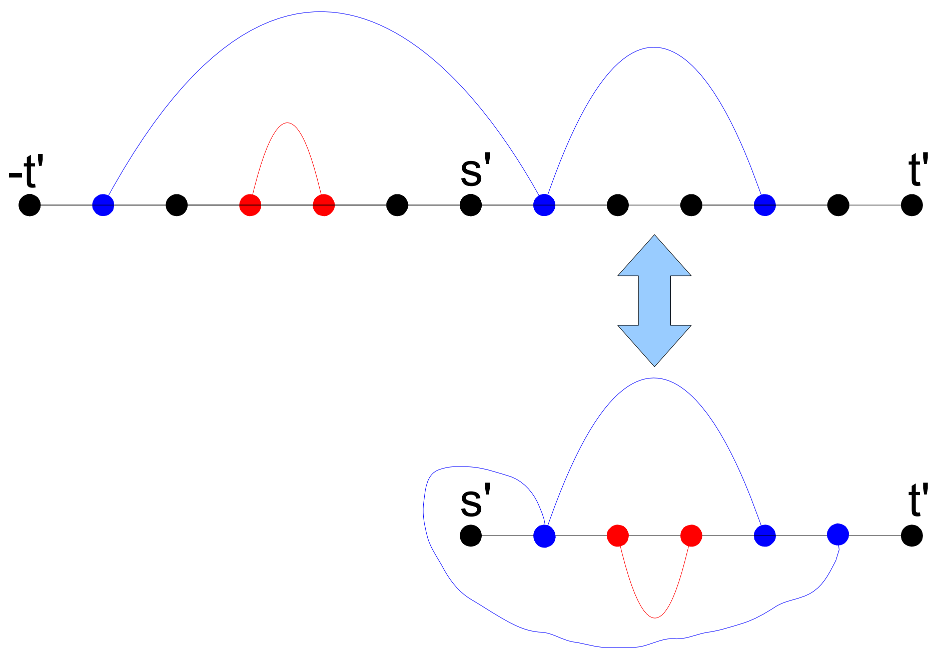

Figure 5.2: Example of a planar full bracketing on an unfolded timeline and the corresponding planar full bracketing on the folded timeline after timeline refolding.

this constraint is that the timeline refolding process would now preserve the planarity of bracketings, as illustrated by Figure 5.2.

As a result, the problem of constructing a planar full bracketing for a folded timeline is now reduced to the problem of constructing a planar full bracketing on the unfolded timeline. Furthermore, observe that because of this additional constraint, a bracketing on an unfolded timeline is planar if and only if there does not exist a pair of brackets whose endpoints are interleaved such as in Figure 5.3. Thus it becomes much simpler to determine whether a bracketing is planar.

Finally, observe that the unfolded timeline of P can also be thought of as a sequence of vertices representing faces in the sensor arrangement. With respect to each sensor, each vertex on the unfolded timeline is considered to be in one of two states: either it is within the sensor, or it is not. Note that this corresponds exactly

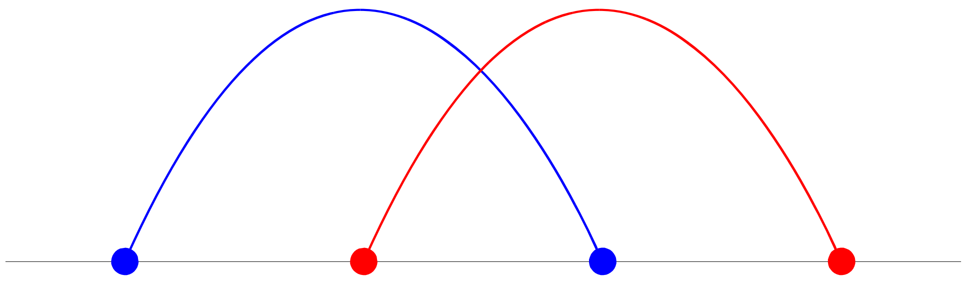

Figure 5.3: A pair of intersecting brackets with interleaved endpoints.

to whether the vertex is assigned the colour associated with the sensor. Thus for any pair of sensors, each vertex is in one of four distinct states, and any sequence of vertices could be thought of as a sequence of such states. Then let a sequence of such states be called an *unfolded timeline configuration*, and an unfolded timeline U contains an unfolded timeline configuration if there exists a pair of sensors such that the sequence of states represented by the timeline configuration is a subsequence of the sequence of states associated with U with respect to this pair of sensors.

The main motivation for the reduction to the unfolded timeline is the *Unfolded Forbidden Configuration*, which is the unfolded timeline configuration described as $\alpha_{-\beta}\beta_{-\alpha}\alpha_{-\beta}\beta_{-\alpha}$, where $\alpha_{-\beta}$ denotes a vertex on the unfolded timeline that is within an intersection of sensor α but not within an intersection of sensor β. We observe that any folded timeline which does not contain a Forbidden Configuration described in Section 5.2 cannot have an unfolded timeline that contains the Unfolded Forbidden Configuration. In other words, all possible timeline configurations on the folded timeline that could unfold to form the Unfolded Forbidden Configuration are Forbidden Configurations, or supersequences of Forbidden Configurations. The proof of this is included in Section A.2.

Thus the unfolded timeline of the peripheral traversal P cannot contain the Unfolded Forbidden Configuration. This is fortunate because as mentioned above, a bracketing on an unfolded timeline cannot be planar if there exists a pair of brackets whose endpoints are interleaved such as in Figure 5.3, which is inevitable in a

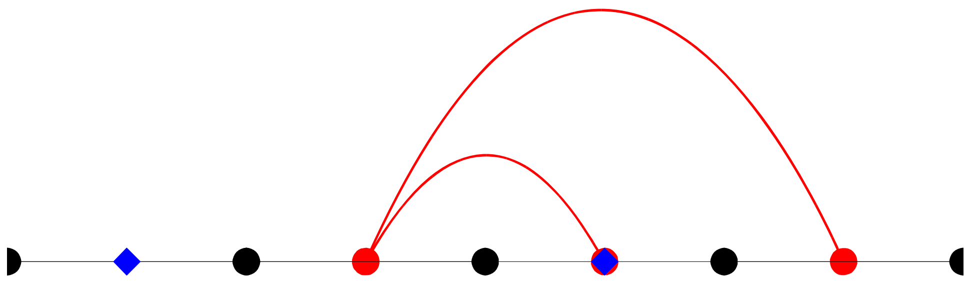

Figure 5.4: An example illustrating why carelessly constructing brackets can render planar full bracketings Impossible.

full bracketing if the unfolded timeline contains the Unfolded Forbidden Configuration.

5.4 Planar Bracketing Construction for Peripheral Traversals

The goal of this section is to prove that all peripheral traversals have a full bracketing which is planar in the sense that no two brackets intersect. In Section 5.3, we reduced the problem to constructing a planar full bracketing for an unfolded timeline. Then given that this unfolded timeline does not contain the Unfolded Forbidden Configuration, we can now describe the procedure for constructing this planar full bracketing. Note that even with these constraints, constructing a planar full bracketing is not a trivial task, as carelessly constructing brackets can easily render planar full bracketings impossible. This is illustrated in Figure 5.4, where the red brackets constructed make it impossible to connect the blue vertices without violating planarity.

We begin by representing the unfolded timeline as a two-dimensional boolean matrix. The rows of this matrix represent the different colours and the columns represent the vertices in the horizontal chain, with each cell $[c, r]$ marked if and only if the vertex associated with column c is assigned the colour associated with row r. Brackets can then be defined to connect the different columns and have endpoints

0	1	0	0	0	1	0	0	0
0	0	0	1	0	1	0	1	0

Table 5.1: Matrix Representation of the Timeline shown in Figure 5.4.

1	0	1	0
0	1	0	1

Table 5.2: First of the two possible 2 by 4 matrices of alternatingly marked cells representing the Unfolded Forbidden Configuration.

only at columns. For instance, Table 5.1 illustrates the matrix representation of the timeline shown in Figure 5.4, where the first row is associated with blue and the second is associated with red.

First off, there is one important constraint on the matrices derived from the constraints on timelines described in Section 5.2:

Observation 5.4.1 *Any matrix derived from an unfolded timeline without the Unfolded Forbidden Configuration must not contain a 2 by 4 submatrix of alternatingly marked cells that represents the Unfolded Forbidden Configuration. In other words, it must not have Table 5.2 or Table 5.3 as submatrices.*

Proof Immediately follows by construction and by definition. ∎

We note that in previous work by Bereg and Kirkpatrick [2], for some restricted problem instances of the Barrier Resilience Problem, they showed that all acyclic resilience-optimal paths can be represented as unfolded timelines, and planar full bracketings could be constructed for these unfolded timelines by recursively splitting the problem into two subproblems. The following procedure is a refinement of their procedure, and within this chapter we will show that our planar full bracketing construction procedure can successfully construct planar full bracketings for

0	1	0	1
1	0	1	0

Table 5.3: Second of the two possible 2 by 4 matrices of alternatingly marked cells representing the Unfolded Forbidden Configuration.

all problem instances of the Barrier Resilience Problem which are reduced to the Minimum-Vertex-Colour Path Problem.

Let a *Splittable Column* be defined as a column c_{split} in the matrix other than the first or last column such that each row of the matrix is in at least one of the following states:

- The row has no marked cells to the right of c_{split}.
- The row has no marked cells to the left of c_{split}.
- The row has a marked cell in c_{split}.
- The row has marked cells in both the first and last columns.

Let M be the matrix representation of an unfolded timeline for a peripheral traversal P. As M cannot have Table 5.2 or Table 5.3 as submatrices, we observe that all submatrices of M must either have at most two columns, or contain at least one Splittable Column. The proof of this observation is included in Appendix B. This allows us to solve the problem of constructing a planar full bracketing for M as follows:

1. Let c_{first} be the first column of M, and let c_{last} be the last column of M. For each row r in M, if r has marked cells in both c_{first} and c_{last}, construct an r-coloured bracket with endpoints at the vertices corresponding to c_{first} and c_{last}.
2. If M has more than two columns, let c_{split} be a Splittable Column of M. Note that if M contains multiple Splittable Columns, any one of them would suffice. Then let M_1 be the submatrix of M that only has the columns between c_{first} and c_{split} inclusive, and let M_2 be the submatrix of M that only has the columns between c_{split} and c_{last} inclusive. Note that both M_1 and M_2 have all the rows of M. Solve the subproblems of constructing planar full bracketings for M_1 and M_2.

Lemma 5.4.2 *If the bracket construction procedure above is applied to a matrix M, the bracketing constructed is planar.*

Proof As observed in Section 5.3, a bracketing on an unfolded timeline is planar if and only if there does not exist a pair of brackets whose endpoints are interleaved such as in Figure 5.3. The bracket construction procedure above only constructs brackets in the first step, and these brackets only have endpoints in the first and last columns. Furthermore, when the problem is split into two subproblems in the second step, the matrices of both subproblems only share the c_{split} column. Thus it is impossible to find a pair of brackets whose endpoints are interleaved such as in Figure 5.3. ▮

Lemma 5.4.3 *For every colour r, if the bracket construction procedure above is applied to a matrix M, all r-coloured vertices of M would be connected by the r-coloured brackets constructed.*

Proof Note that this is trivially true if M contains less than two columns, since the number of vertices of M would be at most 1 and by definition any r-coloured vertex is considered to be connected to itself.

Then consider the case where M has exactly two columns, representing two consecutive vertices. Observe that in the first step of the bracket construction procedure, if both vertices are r-coloured, an r-coloured bracket must have been created to connect them. Thus all r-coloured vertices must be connected.

Finally, consider the case where M has more than two columns. Because the second step solves the subproblems for M_1 and M_2, all r-coloured vertices in M_1 must be connected, and all r-coloured vertices in M_2 must be connected. If either M_1 or M_2 does not contain any r-coloured vertices, then all r-coloured vertices in M are already connected.

If both M_1 and M_2 contain r-coloured vertices, then by the definition of Splittable Columns, either the vertex corresponding c_{split} is r-coloured, or the vertices corresponding to c_{first} and c_{last} are r-coloured. If the vertex corresponding c_{split} is r-coloured, M_1 and M_2 must contain this r-coloured vertex, so all r-coloured vertices in M are already connected.

Alternatively, if the vertices corresponding to c_{first} and c_{last} are r-coloured, the first step would have constructed an r-coloured bracket connecting these two vertices. Then since the r-coloured vertex corresponding to c_{first} is in M_1 while the

the r-coloured vertex corresponding to c_{last} is in M_2, by the transitive nature of the relation, all r-coloured vertices in M must be connected. ∎

Note that this bracket construction procedure can sometimes create more brackets than a full bracketing. However, any subset of a planar bracketing must clearly be planar as well. Thus to derive a planar full bracketing, for every colour r it suffices to choose a spanning tree of r-coloured brackets connecting all r-coloured vertices and remove all other r-coloured brackets.

In particular, note that we can always choose a spanning tree of r-coloured brackets connecting all r-coloured vertices such that every pair of consecutive r-coloured vertices is directly connected by an r-coloured bracket, since ultimately every pair of consecutive vertices must form one of the subproblems solved in the recursion.

Theorem 5.4.4 *Any unfolded timeline U that does not contain the Unfolded Forbidden Configuration must have a planar full bracketing.*

Proof Immediately follows from Lemma 5.4.2 and Lemma 5.4.3. ∎

Corollary 5.4.5 *There exists a planar full bracketing for any folded timeline U of a peripheral traversal.*

Proof Follows directly from Theorem 5.2.3 and Theorem 5.4.4. ∎

5.5 Planar Bracketings for Interior Traversals

In Section 5.4, we showed that planar full bracketings could always be constructed for peripheral traversals due to the absence of the Unfolded Forbidden Configuration on their unfolded timelines. However, for interior traversals, the presence of interior sensor intersections allows the folded timeline to contain the Forbidden Configurations discussed in Section 5.2. Consequently, even if we attempt to construct an unfolded timeline for an interior traversal, the Unfolded Forbidden Configuration may be present and thus the previous arguments cannot be directly applied to construct a planar full bracketing.

In this section, we present a method to get around this issue and construct a planar full bracketing for any interior traversal P. Note that as before, we will assume that the start and end of P are not covered by any sensors, so each sensor intersection is uniquely associated with an entry point and an exit point on the boundary of the sensor.

We begin by splitting every sensor in the arrangement along every sensor intersection made by P. This split is performed such that for each sensor intersection, two subsensors are created which overlap only along the sensor intersection such that the sensor intersection becomes two perfectly overlapping sensor intersections to the two subsensors. The immediate consequence of this sensor split is that there are no longer any interior sensor intersections made by P, since every interior sensor intersection is effectively split into two perfectly overlapping sensor intersections of the two subsensors, where one sensor intersection is positive and the other is negative. In other words, P is transformed into a peripheral traversal on the arrangement of subsensors.

However, this raises another problem: the subsensors are not disk shaped. In fact, it is difficult to describe the exact shape of each subsensor since the path P could be quite arbitrarily shifted around while maintaining the same sequence of faces intersected. Nevertheless, we can prove the following:

Lemma 5.5.1 *Both Axiom 5.2.1 and Axiom 5.2.2 hold for this arrangement of subsensors.*

Proof First, observe that Axiom 5.2.1 must hold since the P is an acyclic path and we only split sensors along P.

Then we can show via a proof by contradiction that Axiom 5.2.2 must hold as well. If Axiom 5.2.2 does not hold, there must exist a pair of subsensors s_α and s_β that do not have a boundary-segregated intersection. Let α and β be the original sensors which s_α and s_β were derived from respectively.

By definition, this means there exists an intersection region $\mathtt{I}$ on the two dimensional plane that satisfies the following constraints:

- $\mathtt{I}$ is a subregion of the intersection of s_α and s_β.
- $\mathtt{I}$ contains no holes.

- $\mathtt{I}$ is bounded only by the boundaries of s_α and s_β.
- Either s_α or s_β is not boundary-isolated with respect to $\mathtt{I}$.

Let $\mathtt{P}_{s_\alpha}$ be the set of points on the boundary of $\mathtt{I}$ that are also on the boundary of s_α. Let $\mathtt{P}_{s_\beta}$ be the corresponding set for s_β.

Without loss of generality, assume s_α is not boundary-isolated with respect to $\mathtt{I}$. Then there must exist two points p_1 and p_2 inside the set $(\mathtt{P}_{s_\alpha} - \mathtt{P}_{s_\beta})$ such that it is impossible to walk from p_1 to p_2 using only points in $\mathtt{P}_{s_\alpha}$. By definition, this means that each of the two ways to walk from p_1 to p_2 along the boundary of $\mathtt{I}$ must use some points inside the set $(\mathtt{P}_{s_\beta} - \mathtt{P}_{s_\alpha})$.

Then observe that since every sensor is split along P, any point inside the set $(\mathtt{P}_{s_\alpha} - \mathtt{P}_{s_\beta})$ must have been on the boundary of α and inside β. Similarly, any point inside the set $(\mathtt{P}_{s_\beta} - \mathtt{P}_{s_\alpha})$ must have been on the boundary of β and inside α. Inevitably, this means α and β cannot have a boundary-segregated intersection, since the points inside the above sets must also exist in the same relative order in some intersection region of α and β.

However, since α and β are the original disk-shaped sensors, they must have a boundary-segregated intersection. This contradiction proves that Axiom 5.2.2 holds for the arrangement of subsensors. ▮

Corollary 5.5.2 *There exists a planar full bracketing for any folded timeline U of a interior traversal.*

Proof Let U' be the corresponding folded timeline through the arrangement of subsensors after we split every sensor at every sensor intersection. By Lemma 5.5.1 and Theorem 5.2.3, there are no Forbidden Configurations in U', meaning the unfolded timeline of U' would not contain the Unfolded Forbidden Configuration. Thus by Theorem 5.4.4 there exists a planar full bracketing for U'. Then observe that this planar full bracketing for U' can be used as a planar full bracketing for U by replacing every colour corresponding to a subsensor with the colour corresponding to the original sensor. ▮

Thus for any problem instance of the Minimum-Vertex-Colour Path Problem derived from the reduction from the Barrier Resilience Problem, we can state the following:

Corollary 5.5.3 *The folded timeline of any acyclic $s' - t'$ path has a planar full bracketing.*

Proof Follows directly from Corollary 5.5.2 and Corollary 5.4.5. ∎

Chapter 6

Resilience Approximations for Intersection-Limited Problem Scenarios

As stated in Chapter 2, the Barrier Resilience Problem is to determine the resilience of a barrier, defined as the minimum over $S-T$ paths of the number of distinct sensors intersected. We note that the work by Bereg and Kirkpatrick [2] defined a similar measure referred to as thickness, which is defined as the minimum over $S-T$ paths of the number of sensor intersections. The difference between the resilience and the thickness is that the resilience counts each sensor in the arrangement at most once, whereas the thickness counts each sensor once for each intersection of the sensor.

Thus in general, the thickness may be arbitrarily higher than the resilience. However, there exist many restricted versions of the Barrier Resilience Problem where the thickness actually serves as a constant factor approximation of the resilience.

For ease of notation, let an acyclic $S-T$ path P be defined as an $(\mathtt{m},\mathtt{i})$-*traversal* if P intersects each sensor at most $\mathtt{m}$ times and at most $\mathtt{i}$ of these intersections are interior intersections. Note that by this definition, P is a peripheral traversal if $\mathtt{i}=0$.

A intersection-limited problem scenario is a set of Barrier Resilience Problem instances for which there exists some finite integer $\mathtt{m}$ such that the existence of a resilience-optimal $(\mathtt{m},\mathtt{i})$-traversal for an arbitrary integer $\mathtt{i}$ is guaranteed. Thus for these problem instances, the thickness must be a $\mathtt{m}$-approximation of the resilience. We are motivated to study these intersection-limited problem scenarios because the work by Bereg and Kirkpatrick [2] has proven the following to be intersection-limited problem scenarios:

Theorem 6.0.4 (Bereg and Kirkpatrick [2]) *In the special case where sensors in* $\mathtt{A}$ *are unit disks and the distance between S and T is greater than* $2\sqrt{3}$*, there exists a resilience-optimal* $(2,0)$*-traversal.*

Theorem 6.0.5 (Bereg and Kirkpatrick [2]) *In the case where sensors in* $\mathtt{A}$ *are unit disks, there exists a resilience-optimal* $(3,0)$*-traversal.*

In this chapter, we will analyze these intersection-limited problem scenarios and explore the constant factor approximations guaranteed by the 2-Path Algorithm. In particular, we will improve on the approximation factors achieved in the work by Bereg and Kirkpatrick [2] for unit disk sensors.

We begin by defining any bracket as a *mini-bracket* if its two endpoints are two consecutive vertices on the path. Then given any intersection-limited problem scenario, let P be one of the resilience-optimal $(\mathtt{m},\mathtt{i})$-traversals that must exist. Following the reduction from the Barrier Resilience Problem to the Minimum-Vertex-Colour Path Problem described in Chapter 2, we note that by Corollary 5.5.3, we can create a planar full bracketing on the folded timeline of P. In particular, as noted in Section 5.4, we can choose a planar full bracketing B_{full} such that every pair of consecutive vertices that share a colour $\mathtt{c}$ must be directly connected by a $\mathtt{c}$-coloured mini-bracket in B_{full}. Let B_{mini} denote the set of mini-brackets in B_{full}, and let B_{rem} be the set of brackets in $B_{\text{full}} - B_{\text{mini}}$.

In Section 6.1, we will show how to split the brackets in B_{rem} into two well-structured bracketings B_1 and B_2. Then in Section 6.2, we will show that both $B_1 \cup B_{\text{mini}}$ and $B_2 \cup B_{\text{mini}}$ are 2-tree-admissible bracketings, and discuss the approximation factors guaranteed by the 2-Path Algorithm.

6.1 Splitting the Brackets into Two Partial Bracketings

In this section, we will split the brackets of B_{rem} into two well-structured partial bracketings B_1 and B_2. Before describing the actual splitting procedure, we must first define some terms that allow us to describe the structure of the brackets.

As B_{full} is a planar bracketing, by definition there must be some way to draw the brackets of B_{full} on the folded timeline such that no pair of brackets intersect and no bracket intersects the horizontal chain of vertices. Given this definition, observe that when drawing a bracket to connect a pair of vertices, the ends of the bracket could be drawn such that they approach the vertex from the bottom or from the top. Let an endpoint be considered *negative* if it approaches its vertex from below, and *positive* otherwise. As there are two endpoints, this allows us to classify the brackets into 4 distinct types.

Let the first endpoint of a bracket be the endpoint that is closer to the start of the path. Then let a *wraparound bracket* be defined as any bracket whose two endpoints approach their vertices from opposite directions. For each wraparound bracket, if the first endpoint is negative, the bracket is considered a *clockwise wraparound bracket*. Otherwise it is considered a *counter-clockwise wraparound bracket*. For the non-wraparound brackets, if both endpoints are positive, the bracket is considered a *positive bracket*, whereas if both endpoints are negative, the bracket is considered a *negative bracket*.

Next, because our goal is to eventually show some subsets of B_{full} are 2-tree-admissible bracketings, we shall define a total ordering for any set of coincident bracket endpoints. In particular, observe that the drawing of the planar bracketing already defines a partial ordering, in the sense that some of the endpoints must be ordered in certain patterns to maintain planarity. For instance, if two positive brackets shares the same vertex at their first endpoint, one bracket must be drawn such that it lies between the other bracket and the horizontal chain of vertices. This inner bracket must have its first endpoint after the first endpoint of the outer bracket, otherwise the planarity of the drawing would be violated. We will assign each set of coincident bracket endpoints some total ordering which matches the partial ordering defined by the drawing.

Then let *direct interference* be a binary relation on the set of wraparound brack-

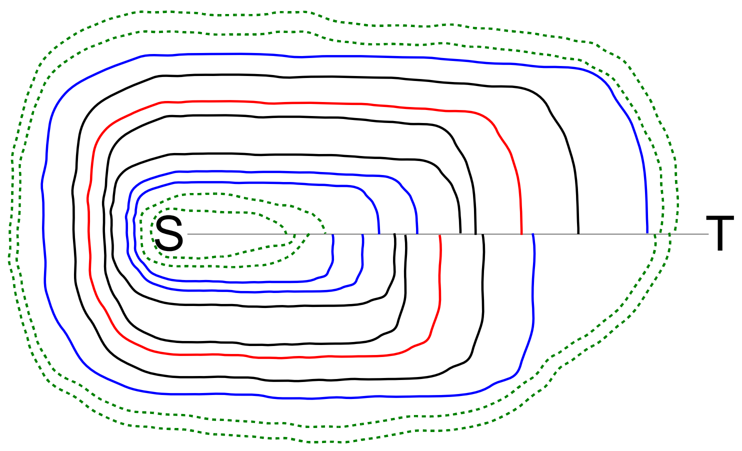

Figure 6.1: Illustration of the Interference Relation between Wraparound Brackets in a Planar Bracketing

ets such that for any pair of wraparound brackets b_i and b_j, b_i is considered to directly interfere with b_j if there is an endpoint of b_i in between the endpoints of b_j on the folded timeline. Then let *interference* be the transitive closure of direct interference, in the sense that any pair of wraparound brackets b_i and b_j interfere if and only if there exists a sequence of wraparound brackets from b_i to b_j such that each wraparound bracket in the sequence directly interferes with its neighbours.

This is illustrated in Figure 6.1, where the brackets have been coloured with respect to their relation to the red bracket. All black brackets are considered to be directly interfering with the red bracket, as they have an endpoint between the endpoints of the red bracket. Note that by definition, they are also considered to be interfering with the red bracket. All blue brackets are not directly interfering with the red bracket since they do not have an endpoint between the endpoints of the red bracket. However, all blue brackets are still considered to be interfering with the red bracket since there exists a sequence of directly interfering wraparound brackets from each blue bracket to the red bracket. In contrast, all the dashed green brackets do not interfere with the red bracket at all, as no such sequence of directly interfering wraparound brackets exists.

A few observations can be made about the structure of these wraparound brackets:

Observation 6.1.1 *For any two wraparound brackets b_i and b_j, if the first endpoint of b_j is after the first endpoint of b_i, the last endpoint of b_j must be after the last endpoint of b_i.*

Observation 6.1.2 *For any two wraparound brackets b_i and b_j, if b_i is clockwise while b_j is counter-clockwise, b_i cannot interfere with b_j.*

The proofs of these observations is included in Section C.1.

We will begin the splitting procedure by assigning all positive brackets to B_1 and all negative brackets to B_2. Then we will divide the folded timeline into maximal non-overlapping intervals, henceforth referred to as *Total Interference Blocks*, under the following constraints:

- Each wraparound bracket has both of its endpoints in the same Total Interference Block.
- Each pair of wraparound brackets in the same Total Interference Block must interfere with each other. Note that by Observation 6.1.2, this means no Total Interference Block may contain both a clockwise wraparound bracket and a counter-clockwise wraparound bracket.
- Each pair of wraparound brackets in different Total Interference Blocks must not interfere with each other.

Note that if there are no wraparound brackets, there would only be one Total Interference Block that spans the entire timeline.

Let the brackets of B_{rem} be ordered by increasing distance between S and their first endpoint. Then for each Total Interference Block that contains wraparound brackets, we will split the wraparound brackets between the bracketings B_1 and B_2 as follows:

1. Let b^* denote the first wraparound bracket in the Total Interference Block. If b^* is clockwise, assign b^* to B_2. Otherwise, assign b^* to B_1.
2. For each remaining bracket b_{DI} in the Total Interference Block, if b_{DI} directly interferes with b^*, assign b_{DI} to the same bracketing as b^*.

3. If there remains a wraparound bracket in the Total Interference Block, by definition it must interfere with b^*. Assign the first remaining wraparound bracket to the bracketing that does not contain b^*. Then let b^* denote this wraparound bracket instead and repeat from the previous step.

We argue that B_1 and B_2 are well-structured bracketings, because if we consider each of them independently, we can make the following observations:

Observation 6.1.3 *For any pair of wraparound brackets b_i and b_j, if b_i interferes with b_j, b_i must directly interfere with b_j.*

Observation 6.1.4 *No non-wraparound bracket that contains endpoints from both a clockwise wraparound bracket and a counter-clockwise wraparound bracket.*

Observation 6.1.5 *If a non-wraparound bracket contains a pair of bracket endpoints corresponding to a pair of wraparound brackets, the pair of wraparound brackets must directly interfere with each other.*

The proofs for these observations are included in Section C.2.

6.2 Tree-Admissibility of the Partial-Bracketings

In this section, we will show that given B_1 and B_2 derived from the splitting procedure in Section 6.1, both $B_1 \cup B_{\text{mini}}$ and $B_2 \cup B_{\text{mini}}$ are 2-tree-admissible bracketings. For simplicity, we will only show this for $B_1 \cup B_{\text{mini}}$, but observe that a symmetric argument can be applied to show the same for $B_2 \cup B_{\text{mini}}$ as well.

First, observe that because each bracket b_i in B_{mini} is a mini-bracket which connects two consecutive vertices, for all bracketings B^* and all brackets $b_j \in B^*$, b_j cannot have an endpoint in between the endpoints of b_i, simply because there is no vertex in between the endpoints of b_i for b_j to use as an endpoint. As a result, all brackets in B_{mini} can be treated as empty brackets. Thus by Lemma 4.3.3, to show that $B_1 \cup B_{\text{mini}}$ is a 2-tree-admissible bracketing, it suffices to show that B_1 is a 2-tree-admissible bracketing.

Now consider just the wraparound brackets in B_1. Let α and β denote endpoints of the first and second wraparound brackets respectively and let κ denote the possibility of having other wraparound bracket endpoints in between, such as an endpoint of the third wraparound bracket.

Lemma 6.2.1 *The first two wraparound brackets can only be in configuration $\alpha\beta\kappa\alpha\beta$ or $\alpha\alpha\beta\kappa\beta$.*

Proof In order for a configuration to be possible, the following constraints apply:

- By definition the configuration must begin with α, since α is associated with the first wraparound bracket.
- The second endpoint of each wraparound bracket must be after the first endpoint by definition.
- The second β must be after the second α by Observation 6.1.1.
- There is no possibility of any wraparound bracket endpoints between the first α and the first β, since we are only considering wraparound brackets and β is associated with the second wraparound bracket by definition. In other words, there cannot be a κ between the first α and the first β.
- If the first β is before the second α, there is no possibility of any endpoints between the second α and the second β, since the splitting procedure in Section 6.1 would have assigned it to B_2 instead and thus it would not be in B_1. In other words, if the first β is before the second α, there cannot be a κ between the second α and the second β.

This leaves only $\alpha\beta\kappa\alpha\beta$ and $\alpha\alpha\beta\kappa\beta$. ∎

Next, let B_{wrap} be some subset of the wraparound brackets in B_1, and let B'_{wrap} be B_{wrap} without the first wraparound bracket in B_{wrap}.

Lemma 6.2.2 *If B'_{wrap} is a 2-tree-admissible bracketing, then B_{wrap} is a 2-tree-admissible bracketing as well.*

Proof By Lemma 6.2.1, there are only two valid configurations for the first two wraparound brackets.

- In the case of $\alpha\beta\kappa\alpha\beta$, the two wraparound brackets are doubly-adjacent such that Lemma 4.3.4 can be applied to prove B_{wrap} is a 2-tree-admissible bracketing as well.
- In the case of $\alpha\alpha\beta\kappa\beta$, the first wraparound bracket does not interleave with the second wraparound bracket, or with any of the other wraparound brackets by definition. This means it is actually an empty bracket. Thus Lemma 4.3.3 can be applied to prove B_{wrap} is a 2-tree-admissible bracketing as well.

Lemma 6.2.3 B_{wrap} *is a* 2*-tree-admissible bracketing.*

Proof By Lemma 6.2.2, B_{wrap} is a 2-tree-admissible bracketing if B'_{wrap} is a 2-tree-admissible bracketing. Thus we can reduce the problem to determining if B'_{wrap} is a 2-tree-admissible bracketing. However, B'_{wrap} itself is a subset of the wraparound brackets in B_1, with less brackets than B_{wrap}. Thus we can apply the reduction recursively until it is no longer possible to apply.

By definition, the reduction can be applied as long as the bracketing contains a wraparound bracket, so the resulting bracketing must have no wraparound brackets. Since there are no wraparound brackets, the bracketing is empty and by Lemma 4.3.2 the bracketing must be a 2-tree-admissible bracketing. Then must be the case that the original bracketing B_{wrap} is also a 2-tree-admissible bracketing. ∎

Now consider all of the brackets in B_1, including the non-wraparound brackets. Note that by the splitting procedure in Section 6.1, B_1 is not assigned any negative brackets, so all non-wraparound brackets in B_1 are positive brackets. Furthermore, observe that any positive bracket cannot interleave with the wraparound brackets except on their negative endpoints, since a positive bracket containing a positive endpoint of a wraparound bracket inevitably violates planarity.

Then with respect to B_1, divide the timeline into maximal non-overlapping intervals, henceforth referred to as *Direct Interference Blocks*, under the following constraints:

- Each wraparound bracket has both of its endpoints in the same Direct Interference Block.

- Each pair of wraparound brackets in the same Direct Interference Block must directly interfere with each other.

- Each pair of wraparound brackets in different Direct Interference Blocks must not interfere with each other.

- Each positive bracket has both of its endpoints in the same Direct Interference Block. This is possible because by Observation 6.1.5, any pair of wraparound brackets with negative endpoints in the same positive bracket must directly interfere with each other, so they must be in the same Direct Interference Block and there is never a need to have a positive bracket overlap more than one Direct Interference Block.

Note that if a positive bracket contains no negative endpoints, it can still be put in a Direct Interference Block associated with some set of wraparound brackets without violating any of the constraints.

The problem at hand is to prove that B_1 is a 2-tree-admissible bracketing. With this division, we can reduce the problem into the subproblems of proving that the set of brackets in each Direct Interference Block is a 2-tree-admissible bracketing. To see why this is the case, consider what it means for the set of brackets in a Direct Interference Block to be a 2-tree-admissible bracketing. It means that the tree construction procedure described in Section 4.3 would succeed for the set of brackets. In particular, observe that after the third step of the procedure, if we remove the nodes whose X-Path profiles contain either s' or t', the construction procedure would still succeed, except the root node of the tree would have an X-Path instance which corresponds to the subpath associated with the Direct Interference Block instead of the entire path. Nevertheless, it would still have a discount bracketing that has at least as many brackets as the set of brackets in the Direct Interference Block.

Then observe that if the set of brackets in each Direct Interference Block is a 2-tree-admissible bracketing, we can construct a tree for the subpath associated with each Direct Interference Block as described above. Since Direct Interference Blocks are non-overlapping intervals, these subpaths can be sequentially combined to form the entire path. In other words, we can create a larger tree VDT for the whole path that uses the valid derivation trees of the Direct Interference Blocks as subtrees. Then observe that the discount bracketing of VDT must be contain the discount bracketings of all Direct Interference Blocks, thus it must have at least as many brackets as the set of brackets in B_1. In particular, observe that because it contains the valid derivation trees of the Direct Interference Blocks as subtrees, it must be possible to construct VDT by following the construction procedure described in Section 4.3 applied to B_1. Thus B_1 must be a 2-tree-admissible bracketing.

Now, let B_{DIB} denote the set of brackets in a Direct Interference Block.

Lemma 6.2.4 *B_{DIB} is a 2-tree-admissible bracketing.*

Proof Note that by applying Lemma 4.3.3 and Lemma 4.3.4, we can reduce the problem of determining whether B_{DIB} is a 2-tree-admissible bracketing to whether B_{DIB} with some empty or doubly-adjacent bracket removed is a 2-tree-admissible bracketing. In particular, observe that Lemma 4.3.3 and Lemma 4.3.4 can be recursively applied to keep removing brackets.

Let B^* be the resulting bracketing after Lemma 4.3.3 and Lemma 4.3.4 can no longer be applied. In other words, B_{DIB} must be a 2-tree-admissible bracketing if B^* is a 2-tree-admissible bracketing, and neither Lemma 4.3.3 nor Lemma 4.3.4 can be applied to further remove brackets from B^*.

Assuming B^* still contains positive brackets, consider the smallest positive bracket b' in B^*, where bracket size is defined by the number of endpoints inside the bracket.

b' cannot be an empty bracket, otherwise it would have been removed by Lemma 4.3.3.

b' cannot contain any positive endpoints, since it is the smallest positive bracket by definition and there cannot be a wraparound bracket with a positive endpoint inside a positive bracket without violating planarity.

b' cannot contain any negative endpoints associated with negative brackets, since the splitting procedure in Section 6.1 does not assign any negative brackets to B_1.

b' cannot contain exactly one negative wraparound endpoint, since b' would then be doubly-adjacent to the wraparound bracket, allowing Lemma 4.3.4 to be applied.

Thus b' must contain multiple consecutive negative wraparound endpoints. Since the wraparound brackets are part of the same Direct Interference Block, they must directly interfere with one another by Observation 6.1.5 and be either all clockwise or all counter-clockwise by Observation 6.1.4.

Assume they are all clockwise. Consider the wraparound brackets b_i and b_j associated any two consecutive negative wraparound endpoints. There can be no positive brackets in between the first endpoints of b_i and b_j, since by the splitting procedure in Section 6.1 there are no negative endpoints in that region and thus any positive brackets there would have been removed by recursive application of Lemma 4.3.3. By Observation 6.1.1, there can also be no other wraparound positive endpoints between the first endpoints of b_i and b_j since their negative endpoints are consecutive.

In the absence of planarity violations required by the fact that the brackets were derived from the planar full bracketing B_{full}, it must then be the case that b_i and b_j are doubly-adjacent, since they are directly interfering consecutive wraparound brackets and no other C_1-coloured brackets have any endpoints between the first endpoints of b_i and b_j or between the last endpoints of b_i and b_j. Then one of these wraparound brackets should have been removed by application of Lemma 4.3.4.

A similar argument also leads a contradiction if we assume instead that all the wraparound brackets are counter-clockwise. Since we reach a contradiction in both scenarios, it must then be the case that B^* does not contain any positive brackets. In other words, B^* only contains wraparound brackets and no other brackets.

Then by Lemma 6.2.3, B^* is a 2-tree-admissible bracketing. Thus B_{DIB} is a 2-tree-admissible bracketing. ∎

Lemma 6.2.5 *B_1 is a 2-tree-admissible bracketing.*

Proof As we have shown that the set of brackets in each Direct Interference Block is a 2-tree-admissible bracketing, this lemma immediately follows by using the reduction previously described. ∎

Lemma 6.2.6 *$B_1 \cup B_{\text{mini}}$ is a 2-tree-admissible bracketing.*

Proof By definition, the brackets of B_{mini} are all mini-brackets, which in some sense are empty brackets with respect to any bracketing since the endpoints of any mini-bracket is a pair of consecutive vertices. By Lemma 6.2.5, B_1 is a 2-tree-admissible bracketing. Thus by repeated application of Lemma 4.3.3, $B_1 \cup B_{\text{mini}}$ is a 2-tree-admissible bracketing. ∎

As previously mentioned, a symmetric argument can be used to show that $B_2 \cup B_{\text{mini}}$ is a 2-tree-admissible bracketing as well.

Next, recall that P, the path associated with these bracketings, was defined as a resilience-optimal $(\mathtt{m}, \mathtt{i})$-traversal. Now let VDT be a 2-Path valid derivation tree for P, and let B_{discount} be the discount bracketing of VDT. By Lemma 4.2.4, the weight of the root node of VDT is equivalent to $\text{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B_{\text{discount}}|$.

Observe that if $|B_{\text{discount}}| = |B_{\text{full}}|$, the weight would be equivalent to the resilience since P is a resilience-optimal $S-T$ path and B_{full} is a full bracketing. On the other hand, if $|B_{\text{discount}}| = |B_{\text{mini}}|$, by our choice of B_{full}, the weight would be equivalent to the number of sensor intersections made by P. Then since P is a $(\mathtt{m}, \mathtt{i})$-traversal, the weight would be a $\mathtt{m}$-approximation of the resilience.

Theorem 6.2.7 *For any intersection-limited problem scenario where the existence of a resilience-optimal $(\mathtt{m}, \mathtt{i})$-traversal is guaranteed, the 2-Path Algorithm guarantees a $\frac{m+1}{2}$-approximation of the resilience.*

Proof As before, let $\rho(\mathtt{A}, S, T)$ denote the resilience. As previously mentioned, $\text{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B_{\text{full}}| = \rho(\mathtt{A}, S, T)$, while $\text{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B_{\text{mini}}| \leq \mathtt{m} \times \rho(\mathtt{A}, S, T)$. Thus B_{rem}, which was defined as $B_{\text{full}} - B_{\text{mini}}$, must have cardinality at most $(\mathtt{m} - 1) \times \rho(\mathtt{A}, S, T)$.

Since B_1 and B_2 were derived by splitting the brackets of B_{rem}, observe that the sets of brackets defined by B_1, B_2, and B_{mini} are all disjoint in the sense that no bracket is in more than one of these sets, and either B_1 or B_2 must contain at least half of the brackets in B_{rem}.

Without loss of generality, let B_1 be the bracketing with at least half of the brackets in B_{rem}. Since $B_1 \cup B_{\text{mini}}$ is a 2-tree-admissible bracketing, Lemma 4.3.1 asserts that there must exist a 2-Path valid derivation tree VDT_1 for P, such that the weight of the root node of VDT_1 is at most $\text{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B_1 \cup B_{\text{mini}}|$, which is $\text{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B_{\text{mini}}| - |B_1|$ since B_1 and B_{mini} are disjoint. In particular, observe that this must be equivalent to $\text{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B_{\text{full}}| + |B_2|$, which in turn is equivalent to $\rho(\mathtt{A}, S, T) + |B_2|$.

Next, because B_1 contains at least half of the brackets in B_{rem}, B_2 must contain at most half of the brackets in B_{rem}. As a result, $|B_2| \leq \frac{\mathtt{m}-1}{2} \times \rho(\mathtt{A}, S, T)$. Thus $\rho(\mathtt{A}, S, T) + |B_2| \leq \frac{m+1}{2} \times \rho(\mathtt{A}, S, T)$. This means the weight of the root node of VDT_1 is a $\frac{m+1}{2}$-approximation of the resilience. Then by Lemma 3.4.2, it immediately follows that the 2-Path Algorithm guarantees a $\frac{m+1}{2}$-approximation of the resilience. ∎

Corollary 6.2.8 *In the special case where sensors in* $\mathtt{A}$ *are unit disks and the distance between S and T is greater than* $2\sqrt{3}$*, a* 1.5*-approximation of the resilience is guaranteed by the* 2*-Path Algorithm.*

Proof By Theorem 6.0.4, there exists a resilience-optimal $(2,0)$-traversal. Thus by Theorem 6.2.7, the 2-Path Algorithm guarantees a $\frac{2+1}{2} = 1.5$-approximation of the resilience. ∎

Corollary 6.2.9 *In the case where sensors in* $\mathtt{A}$ *are unit disks, a* 2*-approximation of the resilience is guaranteed by the* 2*-Path Algorithm.*

Proof By Theorem 6.0.5, there exists a resilience-optimal $(3,0)$-traversal. Thus by Theorem 6.2.7, the 2-Path Algorithm guarantees a $\frac{3+1}{2} = 2$-approximation of the resilience. ∎

Chapter 7

Resilience Approximations for Intersection-Unlimited Problem Scenarios

In the previous chapter, we defined an intersection-limited problem scenario as a set of Barrier Resilience Problem instances for which there exists some finite integer $\mathtt{m}$ such that the existence of a resilience-optimal $(\mathtt{m},\mathtt{i})$-traversal for an arbitrary integer $\mathtt{i}$ is guaranteed. We then showed that the 2-Path Algorithm guarantees a constant approximation of the resilience for any intersection-limited problem scenario.

For the remaining problem scenarios, henceforth referred to as intersection-unlimited problem scenarios, there is no finite integer $\mathtt{m}$ such that the existence of a resilience-optimal $(\mathtt{m},\mathtt{i})$-traversal for an arbitrary integer $\mathtt{i}$ is guaranteed. In order to describe what resilience-optimal $S-T$ paths exist, we shall without loss of generality rescale the inputs of the Barrier Resilience Problem such that the radius of the largest disk sensor is 1, and for some $0 \leq \xi < 1$, the radius of the smallest disk sensor is $1-\xi$. In other words, $\mathtt{A}$ becomes an arrangement of disk sensors with radii in the range $[1-\xi,1]$. Note that this means the case where all sensors are unit disks corresponds to the case where $\xi = 0$.

We observe that if the disk sensors are approximately equal-sized, we can still guarantee the existence of a resilience-optimal peripheral traversal. In other words,

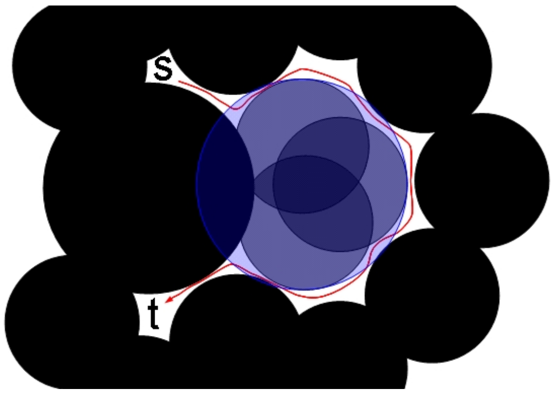

Figure 7.1: An example showing an arrangement of non-identical disk sensors where all resilience-optimal paths must intersect the blue sensor at least 4 times. Smaller sensors tangent to the blue sensor are used to force the red path to alternatingly enter and exit the blue sensor.

for small values of ξ, the existence of a resilience-optimal $(\infty, 0)$-traversal is guaranteed. In fact, we note that by packing arguments, there must exist a function $I(\xi)$ such that:

- The domain is $[0, 1)$.
- The codomain is the set of finite, positive integers.
- The existence of a resilience-optimal $(\infty, I(\xi))$-traversal is guaranteed if the radii of the disk sensors are in the range $[1-\xi, 1]$.

These claims are discussed in more detail in Appendix D. However, we note that if $\xi > 0$, the thickness is no longer guaranteed to be a constant factor approximation of the resilience. In particular, we prove the following:

Theorem 7.0.10 *For all $\xi > 0$ and all pairs of finite integers $\mathtt{m}$ and $\mathtt{i}$, there exists an arrangement $\mathtt{A}$ of disk sensors of radii in the range $[1-\xi, 1]$ and regions S and T such that there is no resilience-optimal $(\mathtt{m}, \mathtt{i})$-traversal.*

Proof This stems from the fact that resilience-optimal $S-T$ paths can be forced to intersect a sensor an indefinite number of times when the radii of disk sensors are different, even when they are almost equal. Thus in general we cannot guarantee the existence of a resilience-optimal $(\mathtt{m}, \mathtt{i})$-traversal for any finite integer $\mathtt{m}$.

The main reason why the slightest difference in disk sizes can force more than $\mathtt{m}$ intersections of a sensor for any integer $\mathtt{m}$ is that a smaller disk can force a path out of a sensor at a single point while not preventing it from returning immediately.

In essence, for any sensor s_i, we can place indefinitely many smaller disks inside s_i that only touch s_i's perimeter at a single point, which forces paths out of s_i. This fact combined with our ability to place indefinitely many outside disks tangent to s_i to force paths back into s_i means we can effectively force paths to intersect s_i an indefinite number of times to avoid all the smaller disks inside s_i and all the outside disks tangent to s_i.

Figure 7.1 illustrates an example of this. For clarity, the arrangement will only force 4 intersections of s_i, but extending this example to force indefinitely more intersections should be straightforward. In this figure, each circle represents a sensor and the largest blue circle represents sensor s_i. Black regions are used to indicate heavy sensor coverage. Observe that the smaller sensors in this figure which touch the boundary of s_i at a single point effectively force any resilience-optimal $S-T$ path to intersect s_i 4 times, since the path must avoid all the smaller sensors in order to be resilience-optimal.

This method of sensor placement can force any number of intersections as long as the disks are not exactly equal sized, since the smaller sensors will still only touch the boundary of s_i at a single point. This corresponds to the case where $\xi > 0$. In contrast, when $\xi = 0$, observe that because the disks are exactly equal sized, it is clearly impossible to place a disk the same size as s_i inside s_i such that it only touches the boundary of s_i at a single point. ∎

As a result, there is no finite integer $\mathtt{m}$ for which we can assume the existence of a resilience-optimal $S-T$ path P such that $\text{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B_{\text{mini}}| \leq \mathtt{m} \times \rho(\mathtt{A}, S, T)$, where B_{mini} denotes the largest partial bracketing composed of only mini-brackets. Thus the previous approach of using B_{mini} to get to within a constant factor of the resilience becomes futile. As such, we know of no polynomial-time constant-approximation of the resilience for such problem scenarios in the absence of additional constraints.

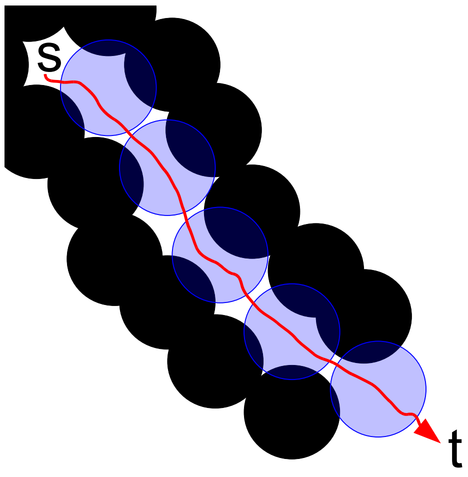

Figure 7.2: Arrangement $\mathtt{A}_1$.

Nevertheless, in the next section we will define a restricted version of the Barrier Resilience Problem, and the rest of this chapter will focus on proving that constant factor approximations are possible for this restricted version even in intersection-unlimited problem scenarios.

7.1 The Barrier Density-Constrained Resilience Problem

Before we define the restricted version of the Barrier Resilience Problem, we shall first describe the motivation behind the restrictions. The Barrier Resilience Problem as defined in Chapter 2 is usually of interest in applications when sensors are not completely reliable. The resilience indicates the minimum number of sensors that have to fail in order for it to be possible for entities to pass through the barrier without entering the region of any active sensor. In some sense this means the resilience is a measurement of the fault tolerance of a barrier, as higher resilience means that more sensors must fail for a path to be uncovered, which would mean the likelihood that the barrier would naturally fail decreases, since more sensors must fail at the same time. However, in some sense this way of defining barrier strength does not capture some of the key elements in natural settings.

This can be seen by considering two arrangements $\mathtt{A}_1$ and $\mathtt{A}_2$, both with barrier resilience 5 and some arbitrarily large number of sensors n. In $\mathtt{A}_1$, the optimal

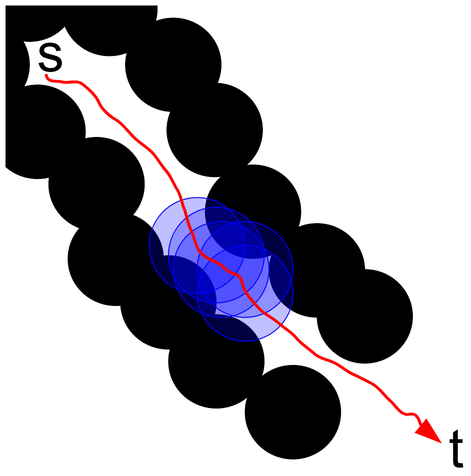

Figure 7.3: Arrangement $\mathtt{A}_2$.

path for traversing entities passes through 5 disjoint sensors, which means entities would be undetected if these 5 sensors failed. In $\mathtt{A}_2$, the optimal path for traversing entities passes through a point covered by 5 sensors but does not encounter sensors anywhere else, which also means the entities would be undetected if these 5 sensors fail. These are illustrated in Figure 7.2 and Figure 7.3, where black areas indicate regions which are densely covered by sensors, blue circles denote individual sensors, and the red path represents the optimal path.

Even though both arrangements have the same resilience and the same number of sensors, $\mathtt{A}_2$ is in some sense a stronger barrier because it forces entities to enter a region covered by 5 sensors at once, which could be advantageous in a number of different settings. This is because all 5 sensors must fail simultaneously in $\mathtt{A}_2$ in order for any traversing entity to make progress through the arrangement, whereas a sequence of failures of one sensor at a time in $\mathtt{A}_1$ would suffice to allow traversers to slowly progress through the arrangement undetected. Thus in some sense, the resilience as defined only measures the *cumulative* sensor coverage, it completely ignores the *instantaneous* sensor coverage.

Let a point or face be called c-detected if it is in the common intersection of at least c distinct sensors. Then a path is called c-evasive if it avoids all c-detected points. To capture a combination or tradeoff of cumulative and instantaneous barrier coverage, an arrangement of sensors is considered a (c,k)-barrier with respect to S and T if every $S-T$ path either (i) has a $(c+1)$-detected face, or (ii) intersects at least k distinct sensors. This allows us to distinguish between $\mathtt{A}_1$ and $\mathtt{A}_2$ as $\mathtt{A}_1$ is a $(1,5)$-barrier, while $\mathtt{A}_2$ is a $(1,\infty)$-barrier since there are no 2-evasive $S-T$ paths.

To this end, we define a generalization of resilience referred to as *Density-c* or D_c-resilience, which considers only $(c+1)$-evasive $S-T$ paths and has value ∞ if no such path exists. With this definition, the standard resilience is then logically equivalent to the D_c-resilience for all $c \geq n$, since all $S-T$ paths would be $(c+1)$-evasive. Also note that the D_0-resilience must logically be either 0 or ∞ and amounts to the somewhat trivial problem of determining if S and T share an uncovered face in the arrangement. Thus it suffices to consider values of c between 1 and n inclusive. Similarly, we shall consider a $(c+1)$-evasive $S-T$ path P to be *D_c-resilience-optimal* if the number of distinct sensors intersected by P is exactly the D_c-resilience.

We shall now formally define the Barrier Density-Constrained Resilience Problem. The input for this problem is a tuple $(\mathtt{A},S,T,c)$. As before, $\mathtt{A}$ is an arrangement of n disk sensors with radii in the range $[1-\xi,1]$, and S and T are two regions on the same plane. The additional parameter c is a positive integer between 1 and n inclusive. The problem is to determine the D_c-resilience, denoted $D_c\rho(\mathtt{A},S,T)$, which is the minimum over $(c+1)$-evasive $S-T$ paths of the number of distinct sensors intersected, including any sensors which cover the endpoints of the path.

Observe that to use polynomial-time approximation-preserving reduction from the Barrier Resilience Problem to the Minimum-Vertex-Colour Path Problem for the Barrier Density-Constrained Resilience Problem, it suffices to remove all vertices corresponding to $(c+1)$-detected faces. In other words, the reduction can be described as follows:

1. Assign each of the n sensors in $\mathtt{A}$ a unique colour, forming the set $\mathtt{S}_{\hat{\mathtt{A}}}$ of n distinct colours.

2. Construct a graph $\hat{\mathtt{A}} = (V,E)$ such that each face in $\mathtt{A}$ becomes a vertex in V and there exists an edge (i,j) in E if and only if the faces of i and j were adjacent in the arrangement. This construction process is illustrated in

Figure 2.1.

3. Assign each vertex i in V the set of colours $\mathtt{S}_i$ which correspond to the set of sensors covering i's associated face in $\mathtt{A}$.

4. Remove each vertex i if $|\mathtt{S}_i| > c$. Also remove all edges of vertex i.

5. For each pair of vertices s' and t' which correspond to faces intersecting S and T respectively:

   (a) Remove from $\hat{\mathtt{A}}$ all of the colours in $\mathtt{S}_{s'} \cup \mathtt{S}_{t'}$.

   (b) Solve the Minimum-Vertex-Colour Path Problem on $(\hat{\mathtt{A}}, s', t')$ to compute $W'(s',t')$, the number of distinct colours used by the Minimum-Colour $s' - t'$ path.

   (c) Readd all the colours removed.

6. Output the minimum over s' and t' of $W'(s',t') + |\mathtt{S}_{s'} \cup \mathtt{S}_{t'}|$.

In the following sections, we will show that if c and ξ are constant, there exists an integer X' such that X' is $O(c)$ and the X'-Path Algorithm guarantees a constant factor approximation of the D_c-resilience if the radii of disk sensors is in the range $[1-\xi, 1]$. As before, we will begin by considering the simpler case where a D_c-resilience-optimal peripheral traversal P is guaranteed, and we will show that in this case a 2-approximation of the D_c-resilience can be guaranteed.

7.2 The 2-Approximation of the D_1-Resilience

As in our previous approaches, the first step in our proof is construct a planar bracketing for the D_1-resilience-optimal peripheral traversal P. However, in this case, the sensor intersections are non-overlapping since the path is 2-evasive by definition. This means given the reduction to the Minimum-Vertex-Colour Path Problem, every vertex on P has at most one colour. We can exploit this structural feature to derive an even more well-structured planar full bracketing by changing our timeline bracket construction procedure to the following:

1. Construct an empty horizontal timeline with left endpoint S and right endpoint T, representing the path from S to T.

2. Unfold the timeline as described before in Section 5.3.

3. For each vertex i, if i has a colour $\mathtt{c}$ and there exist some other vertices to the right of i that are also $\mathtt{c}$-coloured, let j denote the closest of these vertices, and construct a $\mathtt{c}$-coloured bracket with endpoints at i and j.

4. Refold the timeline such that it once again has endpoints S and T.

The advantage we have gained here by modifying the way we select brackets is that we are now guaranteed to form one multi-bracket per colour on the unfolded timeline, where a multi-bracket is a series of brackets associated with the same colour that are adjacent and consecutive, effectively forming a comb-like structure where each vertex assigned the colour has a tooth of the comb connected to it. We must however prove that these multi-brackets form a planar full multi-bracketing, in the sense that no two multi-brackets intersect one another.

Lemma 7.2.1 *Given a D_1-resilience-optimal peripheral traversal P, the bracket construction method results in a planar full multi-bracketing for P.*

Proof We can show a proof by contradiction: Assume the bracket construction method presented above does result in some non-planar bracketing. Then there must exist a pair of brackets that intersects each other. However, as no vertex has more than one colour, this is only possible if the unfolded timeline contains the Unfolded Forbidden Configuration $A_{-B}B_{-A}A_{-B}B_{-A}$, such as illustrated in Figure 5.3. This is a contradiction because by Theorem 5.2.3, P does not contain Forbidden Configurations, which in turn means P cannot contain this Unfolded Forbidden Configuration. Thus it must be the case that this set of multi-brackets is planar. ∎

It is worth emphasizing that this only produces a planar bracketing because of the fact that such vertex has only one colour. Figure 7.4 shows an example where the comb multi-bracketing as described above can become non-planar when the vertices have more than one colour, even though the Unfolded Forbidden Configuration is still absent. Here the multi-brackets for red and blue make the green multi-bracket violate planarity, even though we observe that a planar full bracketing is possible simply by replacing one green bracket. Furthermore this path is

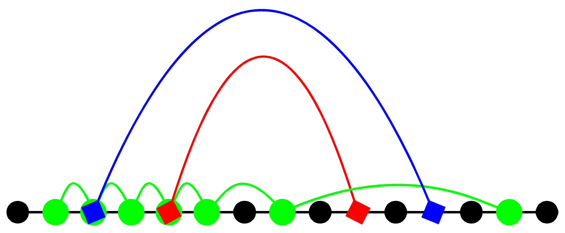

Figure 7.4: A counterexample timeline that cannot have a planar comb multi-bracketing.

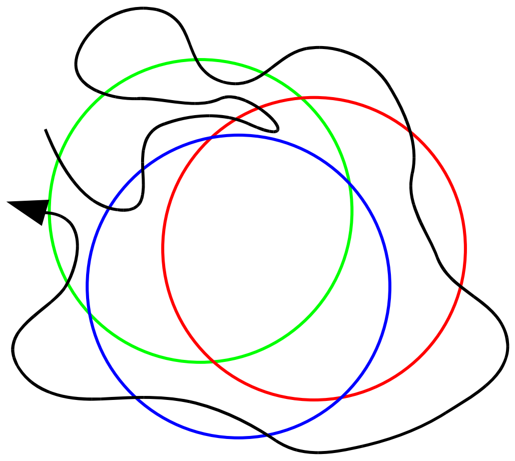

Figure 7.5: The path that realizes the counterexample timeline.

fully realizable, as shown in Figure 7.5. Note that the sensors are well-separated in the figure to emphasize the movement of the path, if the three sensors were drawn so that they overlap almost completely, the path despite appearances could be the optimal path.

Next, let b_i and b_j be two brackets on the unfolded timeline such that b_i and b_j have different colours. Note that since each vertex has at most one colour, b_i and b_j cannot have any coincident bracket endpoints. We consider b_i to *nest* b_j if the endpoints of b_j lie between the endpoints of b_i. Furthermore, b_i is considered the *parent bracket* of b_j if b_i nests b_j but does not nest any other bracket which nests

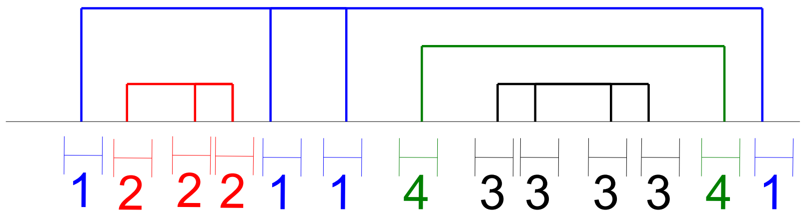

Figure 7.6: An example of a timeline with comb-like multi-brackets.

b_j.

Proposition 7.2.2 *All brackets in a multi-bracket share the same parent bracket.*

Proof Since the multi-brackets are planar and constructed to be comb-like structures, no two brackets in a multi-bracket may nest each other. Furthermore, because each vertex has at most one colour and the Unfolded Forbidden Configuration is absent, there cannot be other brackets with endpoints in between the brackets of this comb-like structure. Thus the parent bracket of any bracket in the multi-bracket cannot have either of its two endpoints within the multi-bracket. In that case, it must be the parent bracket of every other bracket in the multi-bracket as well. ∎

Thus we can clearly define the parent of a multi-bracket m_i as the multi-bracket m_j that contains the parent bracket of the brackets in m_i. For instance, in Figure 7.6, the multi-bracket for s_1 is the parent of the multi-bracket for s_2 and the multi-bracket for s_4 but not the multi-bracket for s_3, because the multi-bracket for s_3 has the multi-bracket for s_4 as its parent.

With these properties of multi-bracket nesting clearly defined, we proceed to greatly simplify the bracketing by removing or breaking all nesting and removing all wraparound brackets. Perhaps unexpectedly, this can actually be done at an amortized cost of only 1 bracket removed for each colour associated with the bracketing, via the following steps:

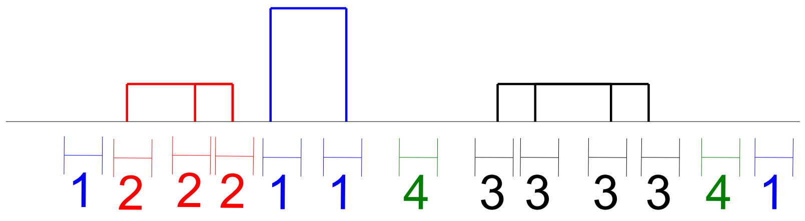

Figure 7.7: An example of a timeline with comb-like multi-brackets after unnesting.

1. Unfold the timeline such that it once again goes from $-T$ to T with center S.
2. To remove nesting, break all parent multi-brackets into sets of multi-brackets via the removal of brackets, and charge the cost of each break to the colour of a child multi-bracket that was nested before the breaking. Each child multi-bracket only needs 1 bracket removed to be unnested since all brackets in the child multi-bracket share the same parent bracket by Proposition 7.2.2. Furthermore, each colour can only have one multi-bracket, each child can only have one parent, and grandparents would charge to the parents instead of the child, so no colour is ever charged the cost of more than 1 bracket.
3. Refold the timeline such that it goes from S to $-T$ again.
4. Since the bracketing is planar, if there is more than one wraparound bracket, one of the wraparound brackets must nest the others since all wraparound brackets must have endpoints on both sides of S on the unfolded timeline. Thus after the previous steps there can only be at most one wraparound bracket left. Furthermore, in the original bracketing there must have been one multi-bracket that is not a child of any other multi-bracket, thus the colour of this multi-bracket has not been charged yet. Therefore we can remove this 1 last wraparound bracket and charge its cost to this colour.

Figure 7.7 shows the bracketing in Figure 7.6 after unnesting. Note that the number of brackets removed is never more than the total number of multi-brackets, and in this case the 3 brackets that were nesting the multi-brackets associated with s_2, s_3, and s_4 were removed. Also note that this can split each multi-bracket into several multi-brackets or even remove some multi-brackets entirely, such as how the multi-bracket for s_4 was removed.

Given a D_1-resilience-optimal peripheral traversal P, let B_{full} be the planar full multi-bracketing of P and let B^* be the bracketing that results from unnesting B_{full}.

Lemma 7.2.3 $\text{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B^*|$ *is a* 2*-approximation of the* D_1*-resilience.*

Proof By the D_1-resilience-optimality of P, $\text{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B_{\text{full}}| = D_1\rho(\mathtt{A}, s, t)$. In particular, the number of distinct colours used by brackets in B_{full} must be at most the D_1 resilience, so B^* must have at most $D_1\rho(\mathtt{A}, s, t)$ fewer brackets than B_{full}. Thus $\text{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B^*| \leq 2 \times D_1\rho(\mathtt{A}, s, t)$. ∎

Observation 7.2.4 *Every vertex on the folded timeline of* P *is in the bracket-spans of at most* 2 *distinct colours with respect to* B^**.*

Proof This can be shown via a proof by contradiction. Assume there is some point on the folded timeline that is within the bracket-spans of more than two distinct colours. If we unfold the timeline, this point becomes two points on the unfolded timeline. Since it is bracket-spanned by more than two distinct colours, at least one of the two resulting points must be bracket-spanned by at least two distinct colours on the unfolded timeline. However, since the multi-brackets are planar by Lemma 7.2.1, in order to be bracket-spanned by at least two distinct colours, there must be some nesting between the multi-brackets which contain this point. This is a contradiction because all nesting was removed. ∎

Theorem 7.2.5 *The* 2*-Path Algorithm guarantees a* 2*-approximation of the* D_1*-resilience if the existence of a* D_1*-resilience-optimal peripheral traversal is guaranteed.*

Proof By Observation 7.2.4, every vertex on the folded timeline of P is in the bracket-spans of at most 2 distinct colours with respect to B^*. Thus we can construct a 2-Path valid derivation tree using the Sweep Span-Exploiting Approach described in Section 4.4, and by Lemma 4.4.5 the weight of the root node of this tree would be at most $\mathrm{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B^*|$, which by Lemma 7.2.3 is a 2-approximation of the D_1-resilience. Then by Lemma 3.4.2, the theorem immediately follows. ■

7.3 The D_1−Resilience Reduction method

As the value of c increases, the timeline derived from the D_c-resilience-optimal peripheral traversal P can become very complex due to the potential for heavily overlapped sensor intersections. While a planar full bracketing can still be constructed as shown in Section 5.4, this planar full bracketing tends to be very complicated as well, with structure that is difficult to exploit. In this section, we present an alternative method to derive a 2-approximation for the D_c-Resilience where we abandon the idea of constructing a planar full bracketing as the first step. Instead, given the D_c-resilience-optimal peripheral traversal P and some integer k:

1. Construct multi-brackets with simple sequential arrangement of each bracket on the unfolded timeline of P. That is, for every colour $\mathtt{c}$, each $\mathtt{c}$-coloured vertex is connected directly to the $\mathtt{c}$-coloured vertex immediately before and the $\mathtt{c}$-coloured vertex immediately after, in a manner similar to how we constructed multi-brackets in the D_1-resilience scenario. Note that because the vertices may have multiple colours in the D_c-resilience scenario, this is no longer guaranteed to be a planar bracketing, as shown in Figure 7.4.

2. Divide the multi-brackets into k partial bracketings such that each colour only appears in one of the bracketings, and for each bracketing, each vertex on the unfolded timeline is only connected to brackets of one colour. This can be viewed as splitting the unfolded timeline into k unfolded timelines, each having the same properties as the unfolded timelines of the D_1-resilience scenario. Note that this step can fail if k is too small.

3. Break nesting in each bracketing as described in Section 7.2. Note that this still only has an amortized cost of 1 per colour, thus maintaining the approximation factor of 2 by Lemma 7.2.3.

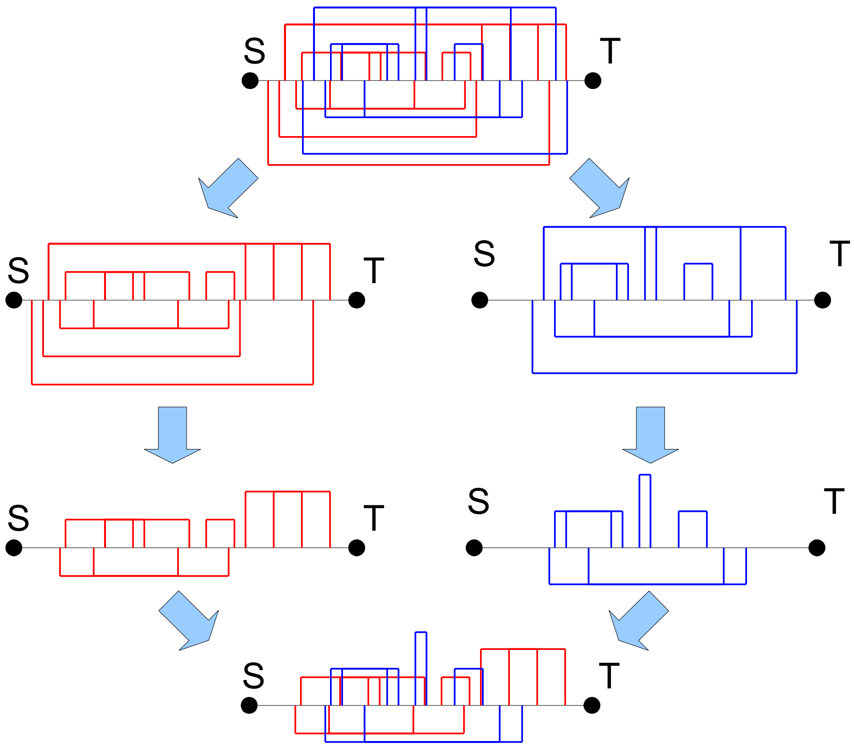

Figure 7.8: An example of the D_1-Resilience Reduction method.

4. Combine the resulting bracketings into one final bracketing which we denote by $B_{\text{final}}(k)$. Note that if the second step failed because k was too small, we'll say the bracketing $B_{\text{final}}(k)$ does not exist.

These steps are illustrated in Figure 7.8. Even though the final bracketing $B_{\text{final}}(k)$ can be non-planar, it is now easier to work with as shown in the following theorem:

Theorem 7.3.1 *If we can construct $B_{\text{final}}(k)$, the $2k$-Path Algorithm will guarantee a 2-approximation of the D_c-resilience.*

Proof When nesting is broken, the timeline for each of the k bracketings is such that every point on the folded timeline is within the bracket-spans of at most two colours associated with the bracketing, according to Observation 7.2.4. This means that when these bracketings are combined into $B_{\text{final}}(k)$, every point on the folded timeline will within the bracket-spans of at most $2k$ colours with respect to $B_{\text{final}}(k)$. Then by Lemma 4.4.5, we can construct a $2k$-Path valid derivation tree whose root node has weight at most $\text{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B_{\text{final}}(k)|$, which by a generalization of Lemma 7.2.3 is a 2-approximation of the D_c-resilience. Then by Lemma 3.4.2, the theorem immediately follows. ∎

However, observe that the X-Path Algorithm has time complexity polynomial in terms of n only if X is treated as a constant independent of n. In particular, if $k = n$, we can clearly use the above procedure to construct a bracketing $B_{\text{final}}(n)$, but then we would require the $2n$-Path Algorithm to get a 2-approximation, and this would not have a time complexity which is polynomial in terms of n. Thus we must show that there is some constant k independent of n for which we can use the above procedure to construct a bracketing $B_{\text{final}}(k)$.

7.4 The 2-Approximation of the D_2−Resilience

In this section, we consider the D_2-resilience for cases where a D_2-resilience-optimal peripheral traversal P is guaranteed. As previously mentioned, we shall show that there exists some constant k such that we can use the method described in Section 7.3 to construct a bracketing $B_{\text{final}}(k)$ on P.

This is very similar to a well-known problem: the Graph Vertex Colouring Problem. In the Graph Vertex Colouring Problem, the input is a graph of vertices and undirected edges. The algorithm is then asked to determine what is the minimum number of distinct colours required in order to assign each vertex exactly one colour under the constraint that no edge in the graph connects two vertices of the same colour. If an input graph can be coloured using k colours, it can be said that there exists a vertex k-colouring for it.

We can actually reduce this problem for D_2-Resilience into a Graph Vertex Colouring Problem by constructing a graph $G_2(\mathtt{A}) = (V_2(\mathtt{A}), E_2(\mathtt{A}))$ from the arrangement $\mathtt{A}$. The set of vertices $V_2(\mathtt{A})$ corresponds to the set of sensors and for every pair of vertices v_1 and v_2, an undirected edge is constructed connecting them if there exists a 3-evasive face in the arrangement that is covered by the sensors associated with those two vertices. Note that for the rest of this chapter, whenever we describe the colour of a sensor, we will be referring to the colour assigned by the solution to the Graph Vertex Colouring Problem, not the colour associated with the reduction to the Minimum-Vertex-Colour Path Problem.

Lemma 7.4.1 *If a vertex k-colouring on $G_2(\mathtt{A})$ exists, then $B_{\text{final}}(k)$ can be constructed using the method described in Section 7.3.*

Proof If there exists a vertex k-colouring on $G_2(\mathtt{A})$, then each sensor has been assigned exactly one colour, and we know that for every face in $\mathtt{A}$ that is covered by exactly 2 sensors, the colours of the 2 sensors are different since there must have been an edge between the associated vertices in $G_2(\mathtt{A})$.

Consider the method described in Section 7.3. It can only fail if it is not possible to divide the multi-brackets into k partial bracketings such that each sensor is associated with at most one of the bracketings, and for each bracketing, each vertex on the unfolded timeline is only connected to brackets of one sensor. However, consider splitting the multi-brackets into these k partial bracketings based on the colour assigned to each sensor, such that each of the k partial bracketings only contains brackets associated with sensors of one colour. Clearly, each sensor will be associated with at most one of the bracketings. Furthermore, since every 3-evasive face in $\mathtt{A}$ is covered by at most one sensor of each colour, each vertex on the unfolded timeline must be only connected to brackets of one sensor in each of the k partial bracketings. Thus $B_{\text{final}}(k)$ can be constructed using the method described in Section 7.3. ∎

Lemma 7.4.2 *For any arrangement $\mathtt{A}$ of disk sensors, if the vertices in $V_2(\mathtt{A})$ are positioned at the centers of their associated sensors and edges in $E_2(\mathtt{A})$ are drawn as straight line segments, the result must be a planar embedding of $G_2(\mathtt{A}) = (V_2(\mathtt{A}), E_2(\mathtt{A}))$, in the sense that none of the edges cross.*

Proof This can be shown via a proof by contradiction. Assume there is some arrangement $\mathtt{A}$ of disk sensors that results in a non-planar embedding of $G_2(\mathtt{A})$. Then there must exist some pair of crossing edges. Note that this pair of crossing edges does not rely on the existence of the other sensors in the graph, since the addition of other sensors can never add edges between existing sensors given the way edges are constructed. Thus removing all sensors other than the 4 involved in the pair of crossing edges must create a small counterexample with only 4 sensors that results in a non-planar embedding of a graph.

Let the edges be e_1 and e_2. An edge is only constructed if there exists a point in the arrangement that is covered by exactly the two sensors of the edge and no other sensors. Let p_1 and p_2 be these points for edges e_1 and e_2 respectively.

Construct the Voronoi diagram of the two points p_1 and p_2. Clearly this cleanly divides the 2-dimensional plane into 2 cells separated by a single straight line. Consider the vertices for e_1: they must be closer to p_1 than p_2, otherwise p_2 would be covered by their sensors and would fail the requirement that it is a point covered by only the sensors of e_2. Similarly, the vertices for e_2 must clearly be closer to p_2 than p_1. Thus we have that the vertices of each edge are strictly confined to the cell of the point for that edge. Then since these cells are separated by a single straight line, it is clearly impossible that these edges cross. Consequently, it is impossible to construct any counterexample arrangement that results in a non-planar embedding of a graph. ▮

Corollary 7.4.3 *For any arrangement* $\mathtt{A}$ *of disk sensors,* $G_2(\mathtt{A})$ *is a planar graph.*

Theorem 7.4.4 *The* 8*-Path Algorithm guarantees a* 2*-approximation of the* D_2*-resilience if the existence of a* D_2*-resilience-optimal peripheral traversal is guaranteed.*

Proof By Corollary 7.4.3 and the Four Colour Theorem, $G_2(\mathtt{A})$ must always be vertex 4-colourable. Then by Lemma 7.4.1 we can construct $B_{\text{final}}(4)$. Finally by Theorem 7.3.1, a 2-approximation can be obtained by using the 8-Path Algorithm. ▮

7.5 The 2-Approximation of the D_c-Resilience

For $c > 2$, the reduction to the Graph Vertex Colouring Problem requires that we construct a generalization of the $G_2(\mathtt{A})$ graph, since the D_c-resilience-optimal peripheral traversal P may now enter regions covered by more than two sensors. This graph, henceforth referred to as a $G_c(\mathtt{A})$ graph, is constructed similarly, with the set of vertices $V_c(\mathtt{A})$ being the set of sensors as before. The only difference is that when constructing edges, we now have to take into account the fact that a path can enter these regions covered by more than two sensors. Thus for every pair of vertices v_1 and v_2, an undirected edge is constructed connecting them if and only if there exists a point in the arrangement that is covered by the sensors associated with those two vertices and at most $c-2$ other sensors. Note that this means every region covered by c distinct sensors requires a clique of size c in $G_c(\mathtt{A})$.

By similar proofs, we can also generalize Lemma 7.4.1 for D_c-resilience.

Corollary 7.5.1 *If a vertex k-colouring on $G_c(\mathtt{A})$ exists, then $B_{\text{final}}(k)$ can be constructed using the method described in Section 7.3.*

Lemma 7.5.2 *If a sensor s_i is fully inside $c-1$ other sensors, it will only be connected to those $c-1$ other sensors in $G_c(\mathtt{A})$.*

Proof This can be shown via a proof by contradiction. Assume s_i is connected to a sensor s_j that is not one of the $c-1$ sensors that fully cover s_i. By definition, this means there is a region in $\mathtt{A}$ that is covered by s_i and s_j and at most $c-2$ other sensors. However, the $c-1$ sensors that fully cover s_i must fully cover this region as well, which is a contradiction. ∎

While we have managed to show that there still exists a reduction to Graph Vertex Colouring, $G_c(\mathtt{A})$ is no longer guaranteed to be planar, and thus no longer guaranteed to be vertex 4-colourable. For instance, Figure 7.9 shows an arrangement of 5 disk sensors. Consider $G_5(\mathtt{A})$. Since there is a region covered by all 5 sensors, there must be a clique of size 5 in $G_5(\mathtt{A})$. However, it is a well-known result that no graph containing a clique of size 5 is planar.

As before, we will aim to show that there exists a constant k for any constant c such that there always exists a k-colouring for the D_c-graph regardless of the size of n. In order to do so, we shall use a more complicated proof involving order-$(c-1)$ Voronoi diagrams with additive weights. These are generalizations of the standard Voronoi diagram such that each cell represents the set of points which share the same $c-1$ closest sites and the distance to each site is the standard Euclidean distance plus the weight of the site.

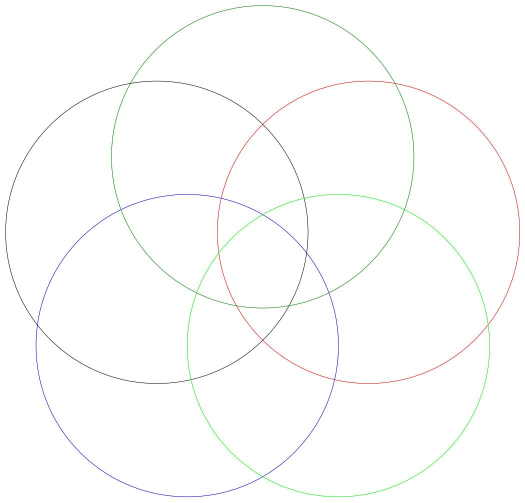

Figure 7.9: An example arrangement illustrating why $G_c(\mathtt{A})$ may be non-planar

We begin by stating a few properties about these Voronoi diagrams in general. Let $\mathbf{weightD}(\check{p}, s_i)$ and $\mathbf{EuclidD}(\check{p}, s_i)$ denote the weighted and Euclidean distances respectively from some point $\check{p}$ to some site s_i. Note that by definition the weighted distance from a site to itself is simply its weight, so $\mathbf{weightD}(s_i, s_i)$ also denotes the weight of a site s_i. We will also let $\mathbf{EuclidD}(\check{p}_1, \check{p}_2)$ denote the Euclidean distance from some point $\check{p}_1$ to some other point $\check{p}_2$. Lastly, for any site s_i, we'll say that it appears in some Voronoi diagram if and only if there exists some cell in that Voronoi diagram that lists s_i as one of its closest sites.

Let $\nu_{\tilde{c}}$ be some order-$\tilde{c}$ Voronoi diagram with additive weights where $\tilde{c}$ is some positive integer, and let $s_{\tilde{i}}$ and $s_{\tilde{j}}$ be some sites that are used in constructing $\nu_{\tilde{c}}$.

Lemma 7.5.3 *If there exists a cell in $\nu_{\tilde{c}}$ that has $s_{\tilde{i}}$ listed as one of its closest sites, site $s_{\tilde{i}}$ must be inside some cell that lists it as one of the closest sites.*

Lemma 7.5.4 *If there exists a cell in $\nu_{\tilde{c}}$ that does not contain the point associated with $s_{\tilde{i}}$ but has $s_{\tilde{i}}$ listed as one of its closest sites, all cells intersected by a line segment from this cell to $s_{\tilde{i}}$ must list $s_{\tilde{i}}$ as one of their closest sites.*

Lemma 7.5.5 *All cells in $\mathcal{V}_{\tilde{c}}$ that have $s_{\tilde{i}}$ listed as one of their closest sites must be connected with respect to $s_{\tilde{i}}$, in the sense that there exists a path between any pair of cells that never enters a cell that does not have $s_{\tilde{i}}$ listed as one of the closest sites.*

Lemma 7.5.6 *If $s_{\tilde{i}}$ and $s_{\tilde{j}}$ both appear in $\mathcal{V}_{\tilde{c}}$, there can only exist a point $\tilde{p}$ in $\mathcal{V}_{\tilde{c}}$ which uses $s_{\tilde{i}}$ and $s_{\tilde{j}}$ as part of its $\tilde{c}+1$ closest sites if there exists at least one edge in $\mathcal{V}_{\tilde{c}}$ which uses $s_{\tilde{i}}$ and $s_{\tilde{j}}$ as part of its set of $\tilde{c}+1$ closest sites.*

The proofs for these properties are included in Section E.1.

Lemma 7.5.7 (Rosenberger [18]) *Any order-$\tilde{c}$ Voronoi diagram of $\tilde{n}$ sites with additive weights in the two-dimensional plane has at most $(4\tilde{c}-2)(\tilde{n}-\tilde{c})$ vertices, $(6\tilde{c}-3)(\tilde{n}-\tilde{c})$ edges, and $(2\tilde{c}-1)(\tilde{n}-\tilde{c})+1$ cells.*

We shall now describe the procedure for transforming our arrangement into an order-$(c-1)$ Voronoi diagrams with additive weights. First, we create a boundary around the arrangement of sensors $\mathtt{A}$ by adding $c-1$ extra sensors with arbitrarily huge radii at each of three corners of a bounding triangle such that all $3c-3$ sensors overlap. The union of these extra sensors effectively surrounds $\mathtt{A}$, as illustrated in Figure 7.10. In other words, this boundary is being created such that by definition for some constant $d_{boundary}$, the following proposition is true:

Proposition 7.5.8 *Any points at any distance from $\mathtt{A}$ greater than $d_{boundary}$ in any direction would inevitably be closer to at least $c-1$ of these extra boundary sensors than to any of the original sensors.*

Then, without loss of generality, let the largest disc in $\mathtt{A}$ have radius equal to 1 via rescaling the arrangement. Let r_i denote the scaled radius of sensor i. Then construct an order-$(c-1)$ Voronoi diagram $\mathcal{V}_{c-1}(\mathtt{A})$ of sites with additive weights, where the center of every sensor s_i is treated as a site with weight $1-r_i$. In other words, the weighted distance from any point in the arrangement to some site is the actual Euclidean distance plus the weight of the site. Similarly, let $\mathcal{V}_c(\mathtt{A})$ denote the order-c Voronoi diagram with the same sites and weights. Note that because every sensor s_i is uniquely associated with exactly one site and exactly one vertex

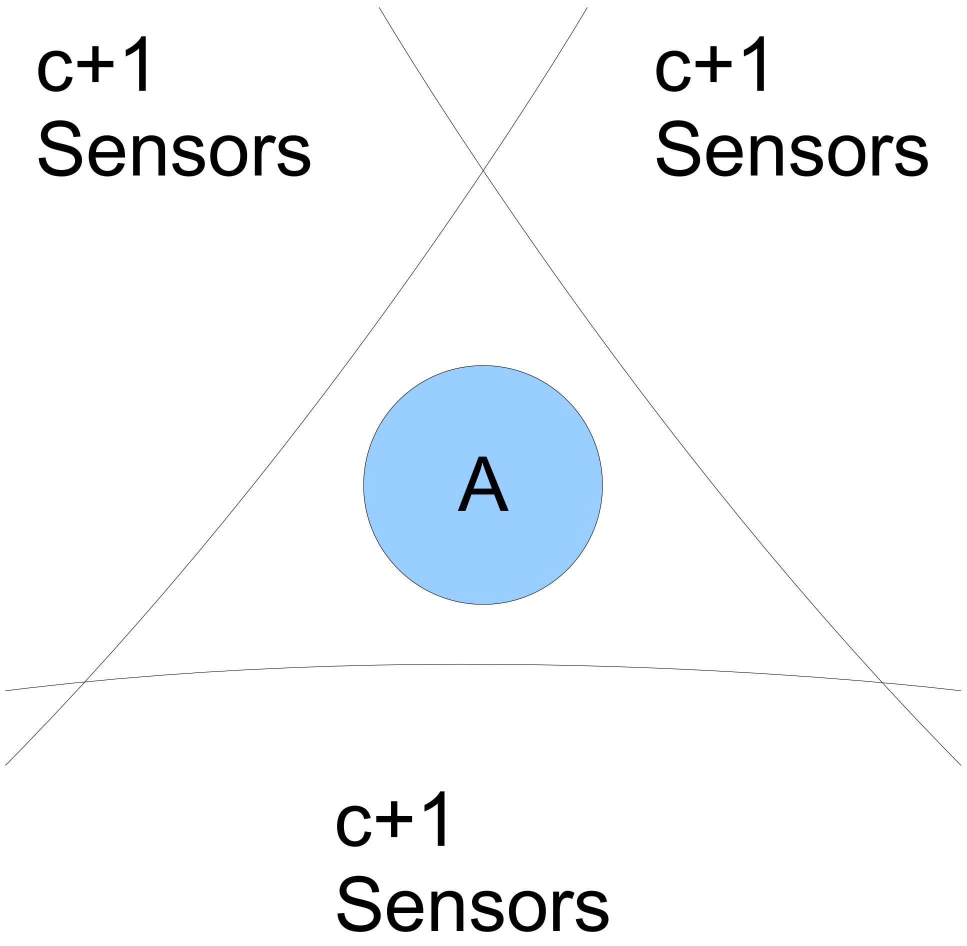

Figure 7.10: Illustration of how extra sensors are added to form a boundary around an arrangement A before the order-$(c-1)$ Voronoi diagram is constructed.

in each arrangement, diagram or graph, we will use s_i to denote the sensor, site, or vertex depending on the context.

Lemma 7.5.9 *Any cell in $\mathcal{V}_{c-1}(\mathtt{A})$ that has a sensor in $\mathtt{A}$ listed as one of its $c-1$ closest sites must be bounded.*

Lemma 7.5.10 *In $\mathcal{V}_{c-1}(\mathtt{A})$, if a sensor s_i is not listed as one of the closest sites for any cell, it must be fully inside $c-1$ other sensors.*

Lemma 7.5.11 *Two vertices s_i and s_j that appear in $\mathcal{V}_{c-1}(\mathtt{A})$ can be connected by an edge in $G_c(\mathtt{A})$ only if there exists at least one edge in $\mathcal{V}_{c-1}(\mathtt{A})$ which uses their two associated sensors as part of the set of closest sites.*

The proofs of these statements are included in Section E.2.

Construct an undirected graph $G(\mathcal{V}_{c-1}(\mathtt{A}))$ with the same set of vertices as $G_c(\mathtt{A})$ and add undirected edges as follows, making sure to not add any edge more than once:

1. Start with some vertex v_i in $G(\mathcal{V}_{c-1}(\mathtt{A}))$ that appears in $\mathcal{V}_{c-1}(\mathtt{A})$, and find an edge in $\mathcal{V}_{c-1}(\mathtt{A})$ that has v_i listed as a closest site on exactly one side of the edge, henceforth referred to as the inside. Create $c-1$ edges in $G(\mathcal{V}_{c-1}(\mathtt{A}))$ that connect v_i to the vertices of $c-1$ sites on the outside. Note that if some of the sites correspond to the extra bounding sensors we added earlier, we don't have to add those edges in.

2. Find another edge in $\mathcal{V}_{c-1}(\mathtt{A})$ that is connected to the previous edge and also has v_i listed as a closest site on exactly one side of the edge. Create edges in $G(\mathcal{V}_{c-1}(\mathtt{A}))$ that connect v_i to the closest sites of the outside cell for this new edge.

3. Repeat the previous step until all such edges are found. This works because all cells that list v_i as their closest site must be connected by Lemma 7.5.5, and by Lemma 7.5.9 these cells are bounded in all directions, so all edges that have v_i listed as a closest site on exactly one side must always be in the form of a ring of edges.

4. Repeat from step 1 for a previously unused vertex in $G(\nu_{c-1}(\mathtt{A}))$ that appears in $\nu_{c-1}(\mathtt{A})$, until all vertices in $G(\nu_{c-1}(\mathtt{A}))$ that appear in $\nu_{c-1}(\mathtt{A})$ have been used.

5. For each vertex v_i that does not appear in $\nu_{c-1}(\mathtt{A})$, if it is fully covered by exactly $c-1$ sensors connect it to those $c-1$ sensors.

Lemma 7.5.12 *The set of edges in $G(\nu_{c-1}(\mathtt{A}))$ is a superset of the set of edges in $G_c(\mathtt{A})$.*

Proof By Lemma 7.5.11, two vertices that appear in $\nu_{c-1}(\mathtt{A})$ can be connected by an edge in $G_c(\mathtt{A})$ only if there exists at least one edge in $\nu_{c-1}(\mathtt{A})$ which uses their two associated sensors as part of the set of closest sites.

If this edge has either one of the two vertices only on one side, then $G(\nu_{c-1}(\mathtt{A}))$ must have an edge between the two vertices.

Otherwise, this means that the cells on both sides on this edge both list both vertices as part of their $c-1$ closest sites.

Consider a path from one of these cells to a cell on the boundary. By Lemma 7.5.9, this means it connects to a cell that does not list the sites associated with either vertex as part of the $c-1$ closest sites. In the absence of degeneracy, each edge on $\nu_{c-1}(\mathtt{A})$ can only replace 1 site at a time, and we know that this path to the boundary must be replacing the sites associated with the two vertices since the final cell does not have them. Thus at some point on the path, a cell that contains exactly one of the two vertices must be encountered. Then the edge on $\nu_{c-1}(\mathtt{A})$ between this cell and the previous one must be an edge that has one vertex listed as a closest site on only one side of the edge. Then $G(\nu_{c-1}(\mathtt{A}))$ must have an edge between the two vertices if they both appear in $\nu_{c-1}(\mathtt{A})$.

If one vertex does not appear in $\nu_{c-1}(\mathtt{A})$, then by Lemma 7.5.10 it must be fully covered by at least $c-1$ sensors. If it is fully covered by exactly $c-1$ sensors, in the last step we would have connected that site to those sensors. Note that if it is fully covered by more sensors, then by definition all points in that sensor are $(c+1)$-detected, so the $(c+1)$-evasive D_c-resilience optimal path cannot use it and we don't need to add any edges to $G_c(\mathtt{A})$ for it. By Lemma 7.5.2, the other vertex must belong to one of the $c-1$ sensors fully covering the vertex, so it must be that $G(\nu_{c-1}(\mathtt{A}))$ has an edge between the two vertices.

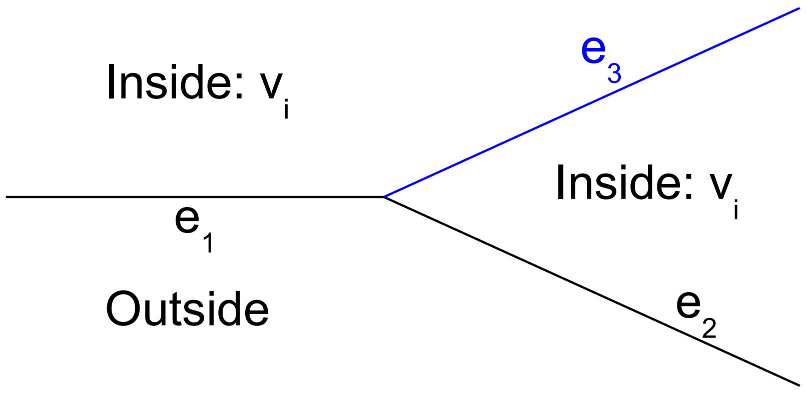

Figure 7.11: Illustration of how the second step could share the same outside cell.

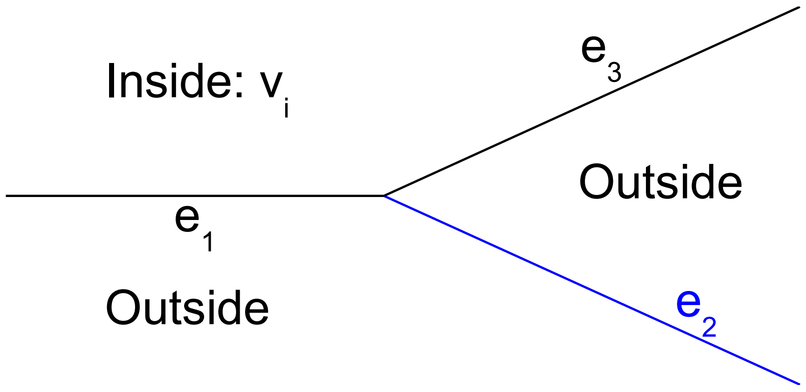

Figure 7.12: Illustration of how the second step could have a different but adjacent outside cell.

Thus all edges in $G_c(\mathtt{A})$ can exist only if they also exist in $G(v_{c-1}(\mathtt{A}))$, meaning the edges of $G(v_{c-1}(\mathtt{A}))$ form a superset of the edges in $G_c(\mathtt{A})$. ▮

As previously defined, n denotes the number of sensors in the arrangement $\mathtt{A}$. Since vertices in $G(v_{c-1}(\mathtt{A}))$ are derived from the sensors in $\mathtt{A}$, n is also the number of vertices in $G(v_{c-1}(\mathtt{A}))$.

Lemma 7.5.13 *$G(v_{c-1}(\mathtt{A}))$ has at most $n(c-1)+2(6c-9)(n+2c-2)$ edges.*

Proof All repetitions of the first step and the last step add $(c-1)$ edges times the number of vertices, so a total of $n(c-1)$ edges.

In the second step, we only use an edge that is connected to the previous one. This edge, being connected to the previous one, must either share the same outside cell, or use an outside cell that is adjacent to the previous outside cell. This is illustrated by Figure 7.11 and Figure 7.12, where e_1 is the previous edge and the current edge is the remaining black edge. Effectively this means the closest sites used by the outside cell are either identical to the previous outside cell or differ only by one. Thus we only need to add at most 1 extra edge to $G(\nu_{c-1}(\mathtt{A}))$ for this edge.

Furthermore, we chose edges defined to have a certain site be listed on exactly one side of the edge. Since the cells on both sides of the edge can only differ by one site, each edge can only be used for two distinct sites. Thus each edge can only be used twice in the second step. By Lemma 7.5.7, the maximum number of edges in $\nu_{c-1}(\mathtt{A})$ is at most $(6(c-1)-3)(n+3(c-1)-(c-1)) = (6c-9)(n+2c-2)$. Thus the second step and all its repetitions can add at most $2(6c-9)(n+2c-2)$ edges.

Combined, this is a total of at most $n(c-1)+2(6c-9)(n+2c-2)$ edges in $G(\nu_{c-1}(\mathtt{A}))$. ∎

Lemma 7.5.14 *There must exist a vertex in $G_c(\mathtt{A})$ which has degree at most $74c-110$, assuming $n \geq c$.*

Proof By Lemma 7.5.13, $G(\nu_{c-1}(\mathtt{A}))$ has at most $n(c-1)+2(6c-9)(n+2c-2)$ edges. By Lemma 7.5.12, the set of edges in $G(\nu_{c-1}(\mathtt{A}))$ is a superset of the set of edges in $G_c(\mathtt{A})$, so $G_c(\mathtt{A})$ has at most $n(c-1)+2(6c-9)(n+2c-2)$ edges.

Then the sum of vertex degrees in $G_c(\mathtt{A})$ is at most $2n(c-1)+4(6c-9)(n+2c-2)$. In turn this means there must exist a vertex in $G_c(\mathtt{A})$ which has degree at most $\frac{2n(c-1)+4(6c-9)(n+2c-2)}{n}$, which is less than $74c-110$ under the assumption that $n \geq c$ and $c > 1$. Note the latter assumption must be true or there would be no edges at all by definition. ∎

Lemma 7.5.15 *There exists a vertex $(74c-109)$-colouring for $G_c(\mathtt{A})$.*

Proof By Lemma 7.5.14, there exists a vertex which has degree less than $74c - 110$. Since we have $(74c - 109)$ distinct colours, this vertex can be coloured the remaining colour at the end with no problems. Thus we reduce the problem to one where this vertex doesn't exist.

However, the set of edges in this reduced problem must be a subset of the edges in the G_c graph derived from the arrangement with the associated sensor removed. Then once again by Lemma 7.5.14, there exists a vertex which has less than $74c - 110$ edges, which means we can reduce the problem again. Recursively reduce the problem in this manner until the graph has less than $74c - 110$ vertices, noting that the assumption that $n \geq c$ still holds. Then simply colour each vertex in the final graph a different colour. Since we can $(74c - 109)$-colour the final problem, the reductions state that the original graph $G_c(\mathtt{A})$ must have a vertex $(74c - 109)$-colouring as well. ∎

Theorem 7.5.16 *If the existence of a D_c-resilience-optimal peripheral traversal is guaranteed, a 2-approximation of the D_c-resilience is guaranteed by the $(148c - 218)$-Path Algorithm.*

Proof By Lemma 7.5.15, $G_c(\mathtt{A})$ must always be vertex $(74c - 109)$-colourable. Then by Corollary 7.5.1, we can construct $B_{\text{final}}(74c - 109)$. Finally by Theorem 7.3.1, a 2-approximation of the D_c-resilience is guaranteed by the $(148c - 218)$-Path Algorithm. ∎

Note that the $(148c - 218)$-Path Algorithm has time complexity is polynomial in terms of n if c is a constant independent of n. Additionally, we observe that this method allows us to derive a 2-approximation of the resilience under some constraints:

Let the *point density* of the the sensor arrangement, denoted **PD**$(\mathtt{A})$, be defined as the maximum over all points p of the number of sensors in $\mathtt{A}$ whose associated regions contain p.

Corollary 7.5.17 *If the existence of a resilience-optimal peripheral traversal is guaranteed and if there is a constraint on $\mathtt{A}$ that states $PD(\mathtt{A}) \leq k$ for some constant $k > 1$, a 2-approximation of the resilience is guaranteed by the $(148k - 218)$-Path Algorithm.*

Proof By the definition of $PD(\mathtt{A})$, no region in $\mathtt{A}$ can be covered by more than $PD(\mathtt{A})$ sensors, which means the resilience is equivalent to the $D_{PD(\mathtt{A})}$-resilience. Then by Theorem 7.5.16 and the constraint that $PD(\mathtt{A}) \leq k$, a 2-approximation of the resilience is guaranteed by the $(148k-218)$-Path Algorithm. ∎

Let the *sensor density* of the sensor arrangement, denoted **SD**($\mathtt{A}$), be defined as the maximum over all sensors s_i in $\mathtt{A}$ of the number of sensors in $\mathtt{A}$ whose associated regions overlap the region of s_i, including s_i itself. By definition this means **SD**($\mathtt{A}$) $\geq$ **PD**($\mathtt{A}$).

Lemma 7.5.18 *If there is a constraint on* $\mathtt{A}$ *that states* $SD(\mathtt{A}) \leq k$ *for some constant* $k > 1$, $G_c(\mathtt{A})$ *must be vertex k-colourable for any value of c.*

Proof By definition, $SD(\mathtt{A})$ is the maximum number of sensors in $\mathtt{A}$ that overlap any sensor in $\mathtt{A}$ on the two-dimensional plane, including itself. Thus $SD(\mathtt{A}) - 1$ is an upper bound on the degree of every vertex in $G_c(\mathtt{A})$ for all values of c, since two vertices cannot be connected if their sensors do not overlap in the arrangement. Thus $G_c(\mathtt{A})$ must be vertex k-colourable for any value of c. ∎

Corollary 7.5.19 *For any integer c, if the existence of a* D_c*-resilience-optimal peripheral traversal is guaranteed and if there is a constraint on* $\mathtt{A}$ *that states* $SD(\mathtt{A}) \leq k$ *for some constant* $k > 1$*, a* 2*-approximation of the* D_c*-resilience is guaranteed by a* 2*k-Path Algorithm.*

Proof By Lemma 7.5.18, $G_c(\mathtt{A})$ must always be vertex k-colourable. Then by Corollary 7.5.1 we can construct $B_{\text{final}}(k)$. Finally by Theorem 7.3.1, we can obtain a 2-approximation by using a $2k$-Path Algorithm. Note that this includes even the standard or D_n-resilience. ∎

7.6 The Constant Approximation of the D_c−Resilience

As defined in Section 7.1, the input of the Barrier Density-Constrained Resilience Problem is a tuple $(\mathtt{A}, S, T, c)$, where $\mathtt{A}$ is an arrangement of n disk sensors with radii in the range $[1-\xi, 1]$, S and T are two regions, and c is a positive integer between 1 and n inclusive.

In this section, we will use a combination of the techniques presented in previous sections to prove the main result of this thesis. We will show that for any problem instance of the Barrier Density-Constrained Resilience Problem, the Multi-Path Algorithms can always be used to guarantee a constant factor approximation of the D_c-resilience in time polynomial in terms of n, if c and ξ are treated as constants.

As previously mentioned, there must exist a function $I(\xi)$ such that if the D_c-resilience has a finite value, there must exist an acyclic D_c-resilience-optimal $(\infty, I(\xi))$-traversal P. First, we shall split every sensor along every intersection of P, using the same way we described in Section 5.5. Then given the timeline of P associated with the subsensors, our goal is to apply the D_1-resilience reduction method outlined in Section 7.3.

As before, we must show that there is some constant k independent of n for which we can construct $B_{\text{final}}(k)$. First construct the $G_c(\mathtt{A})$ graph for the original sensors in the arrangement as outlined in Section 7.5. By Lemma 7.5.15, there exists a vertex $(74c-109)$-colouring for $G_c(\mathtt{A})$. However, since we split the sensors into subsensors which could have overlapping regions, these subsensors have to be coloured too. This can be accomplished by observing that since every sensor was split along every intersection, subsensors derived from the same sensor must overlap only exactly on the intersections. In other words, if we consider just the subsensors of a single sensor, it is effectively a separation of the sensor into contiguous regions forming a map-like structure. Consequently, the subsensors for each sensor must have a 4-colouring by the Four Colour Theorem. Thus the set of all subsensors must have a $(296c-436)$-colouring. This means we can construct $B_{\text{final}}(296c-436)$.

Then note that since the bracket breaking procedure in Section 7.3 would remove one bracket per sensor involved in the bracketing, the fact that we split the sensors into subsensors means we are no longer guaranteed a 2-approximation. By definition, P makes at most $I(\xi)$ interior intersections per sensor, where $I(\xi)$ is a positive integer independent of n. Observe that if there are at most $I(\xi)$ interior intersections per sensor, while the number of subsensors per sensor is unbounded, the number of multiply-intersected subsensors per sensor must be at most $I(\xi)+1$. Thus the bracket breaking procedure would remove at most $I(\xi)+1$ brackets per sensor involved in the bracketing. Consequently, we have the following result:

Theorem 7.6.1 *A* $(I(\xi)+2)$*-approximation of the* D_c*-resilience is guaranteed by the* $(592c-872)$*-Path Algorithm.*

Proof By a generalization of the arguments in the proof of Theorem 7.3.1, we can prove that the output of the $(592c-872)$-Path Algorithm must be at most $\text{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B_{\text{final}}(296c-436)|$. Then since we broke at most $I(\xi)+1$ brackets per sensor to construct $B_{\text{final}}(296c-436)$, $\text{Total}(P, \mathtt{S}_{\hat{\mathtt{A}}}) - |B_{\text{final}}(296c-436)| \leq (I(\xi)+2) \times D_c\rho(\mathtt{A}, S, T)$. Thus a $(I(\xi)+2)$-approximation of the D_c-resilience is guaranteed by the $(592c-872)$-Path Algorithm. ▌

Thus to get a $(I(\xi)+2)$-approximation of the D_c-resilience, it suffices to use the $(592c-872)$-Path Algorithm. Note that this has time complexity polynomial in terms of n if c is a constant independent of n. Thus we have proven that the Multi-Path Algorithms can be used to achieve constant approximations of the D_c-resilience in time polynomial in n if c and ξ are treated as constants independent of n. We can also state generalizations of Corollary 7.5.17 and Corollary 7.5.19:

Corollary 7.6.2 *If there is a constraint on* $\mathtt{A}$ *that states* $PD(\mathtt{A}) \leq k$ *for some constant* $k > 1$*, a* $(I(\xi)+2)$*-approximation of the resilience is guaranteed by the* $(592k-872)$*-Path Algorithm.*

Corollary 7.6.3 *For any integer c, if there is a constraint on* $\mathtt{A}$ *that states* $SD(\mathtt{A}) \leq k$ *for some constant* $k > 1$*, a* $(I(\xi)+2)$*-approximation of the* D_c*-resilience is guaranteed by a* $8k$*-Path Algorithm.*

Chapter 8

Conclusion

In this thesis, we reduced the Barrier Resilience Problem for disk-shaped sensors to the Minimum-Vertex-Colour Path Problem, a variant of the Minimum-Colour Path Problem investigated by Yuan et al. [24]. We formulated and analyzed the Multi-Path Algorithms as approximation algorithms for the Minimum-Vertex-Colour Path Problem, and showed that they have time complexity polynomial in terms of the number of vertices if a constant number of paths is used [Theorem 3.3.2].

We showed that many restricted versions of the Barrier Resilience Problem, such as the case where sensors are unit disks, are intersection-limited problem scenarios, in the sense that we can guarantee the existence of an acyclic resilience-optimal $S-T$ path which intersects each sensor at most $\mathtt{m}$ times for some finite integer constant $\mathtt{m}$. For these problem scenarios, we showed that the 2-Path Algorithm guarantees a $\frac{m+1}{2}$-approximation of the resilience [Theorem 6.2.7]. In particular, this means that the 2-Path Algorithm, in time polynomial in terms of n, guarantees a 2-approximation of the resilience for any arrangement of unit disk sensors [Corollary 6.2.9]. This further tightens to a 1.5-approximation in the case where the distance between S and T is greater than $2\sqrt{3}$ [Corollary 6.2.8]. These results improve on the previous best-known approximation ratios of 3 and $\frac{5}{3}$ [2].

Next, we defined a generalization of the resilience referred to as Density-c or D_c-resilience, which is a measure of the resilience under the constraint that paths cannot enter regions which are covered by more than c distinct sensors. We then proved that for any constant c, there exists a constant X' such that the X'-Path Algorithm guarantees a 2-approximation of the D_c-resilience in time polynomial in n for any problem instance where there must exist an acyclic D_c-resilience-optimal peripheral traversal [Theorem 7.5.16]. Furthermore, we extended this to show that for any constant c, there exists a constant X' such that the X'-Path Algorithm guarantees a constant approximation of the D_c-resilience in time polynomial in n for any problem instance where there must exist an acyclic D_c-resilience-optimal $S-T$ path that intersects each sensor at most `i` times for some finite integer constant `i` [Theorem 7.6.1].

We observe that for any problem instance where the disk sensors have radii in range $[1-\xi, 1]$ for some constant $0 \leq \xi < 1$, if there is a $(c+1)$-evasive $S-T$ path, there must exist an acyclic D_c-resilience-optimal $S-T$ path that intersects each sensor at most `i` times for some finite integer constant `i`. In fact, for many problem instances, there must exist an acyclic D_c-resilience-optimal peripheral traversal, especially when ξ is small. Consequently, these results mean that the Multi-Path Algorithms can be used to guarantee constant approximations of the D_c-resilience for any arrangement of disk sensors with radii in range $[1-\xi, 1]$ for some constant $0 \leq \xi < 1$ and any constant c, which tightens to a 2-approximation under certain conditions. This is a significant result in that there has previously been no known polynomial-time algorithm that gives any constant-approximation of the resilience when disk sensors were of different sizes, and the D_c-resilience may be close to the resilience in many situations even for small c.

Furthermore, we defined two measures of barrier density, the point density which is the maximum number of sensors that covers any point, and the sensor density which is the maximum number of sensors that intersect any sensor, including itself. In practice, barrier coverage problems usually spread out sensors over a long strip of area, so their densities can actually be treated as constants. We showed that if the density of the arrangement for either one of these definitions can be treated as a constant, a constant approximation of the resilience can be derived in time polynomial in terms of n [Corollary 7.6.2 and Corollary 7.6.3].

Thus we have shown that for arrangements of disk sensors in a two-dimensional plane, the above constant-approximations are guaranteed by the Multi-Path Algorithms in time polynomial in n. However, as described, the exponents are too high to use in practice. One obvious direction for future research would be to make large optimizations to the algorithms to be able to actually deploy some of them. In fact, the approaches described in Section 4.3 and Section 4.4 suggest the existence of methods for significantly cutting down on the time complexity of the algorithm while maintaining the same approximation factors.

We also note that with respect to the Multi-Path Algorithms described in Chapter 3, the `M`-functions, which in some sense exploit rules on X-Path instance merging, are not the only way to produce new upper bounds for the number of hidden colours associated with an X-Path profile. In Appendix F, we define `D`-functions, which in some sense exploit rules on X-Path instance decomposition. We note that in some situations, these `D`-functions may cause the algorithm to perform significantly better. However, by their nature, these `D`-functions do not provide any benefit in the worst-case scenarios that we study in this thesis. Thus they are not used in any of the proofs above.

Another direction worth investigating is the notion of higher-order thicknesses and resiliences. Treating the standard thickness and resilience as the first-order, the second-order thickness counts the minimum number of times a path from S to T intersects a sensor while it is already inside at least 1 other sensor, and the second-order resilience is the number of sensors which must be removed in order for the second-order thickness to become 0, with higher orders defined similarly. This is a definition of barrier strength that is useful for real world applications where sensors have a high tendency to give false positives, thus requiring multiple sensors to give a positive signal before the we can believe there may really be an intruder in the vicinity.

While higher-order thickness can clearly be determined just as easily as first-order thickness by adjusting the edge weights in the dual graph of the arrangement, the higher-order resilience seems much harder to compute due to the choice of which sensor to remove from the arrangement. Of particular note is that there exist counterexamples which show that other than the bound on first-order resilience by first-order thickness, no other order of thickness or resilience forms any constant approximation on one another even in the case where sensors are unit disks.

Finally, it is worth exploring the scenario when sensors are on some other surface instead of a two dimensional plane. For instance, when sensors are on a torus, even if the sensor regions are unit disks, the forbidden configurations discussed in Section 5.2 are no longer unrealizable and the optimal path can in fact be forced by the arrangement to make sequences of sensor intersections that are supersequences of these configurations even without making any interior sensor intersections. However, it seems likely that there exists another set of forbidden configurations on a torus, as it seems impossible to be able to make an indefinite number of alternating positive peripheral intersections to any two sensors.

Bibliography

[1] A. Arora, R. Ramnath, E. Ertin, P. Sinha, S. Bapat, V. Naik, V. Kulathumani, H. Zhang, H. Cao, M. Sridharan, S. Kumar, N. Seddon, C. Anderson, T. Herman, N. Trivedi, M. Nesterenko, R. Shah, S. Kulkami, M. Aramugam, L. Wang, M. Gouda, Y. ri Choi, D. Culler, P. Dutta, C. Sharp, G. Tolle, M. Grimmer, B. Ferriera, and K. Parker. Exscal: elements of an extreme scale wireless sensor network. In *Embedded and Real-Time Computing Systems and Applications, 2005. Proceedings. 11th IEEE International Conference on*, pages 102 – 108, aug. 2005. doi:10.1109/RTCSA.2005.47. → pages 1

[2] 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. ISBN 978-3-642-05433-4. doi:10.1007/978-3-642-05434-1_5. URL http://dx.doi.org/10.1007/978-3-642-05434-1_5. → pages ii, 5, 6, 7, 25, 51, 58, 59, 100, 142

[3] W. Cao and T. He. Barrier coverage of wireless sensor networks based on clifford algebra. In *Computer Science and Computational Technology, 2008. ISCSCT '08. International Symposium on*, volume 2, pages 49 –52, dec. 2008. doi:10.1109/ISCSCT.2008.188. → pages 3

[4] A. DuttaGupta, A. Bishnu, and I. Sengupta. Optimisation problems based on the maximal breach path measure for wireless sensor network coverage. In S. Madria, K. Claypool, R. Kannan, P. Uppuluri, and M. Gore, editors, *Distributed Computing and Internet Technology*, volume 4317 of *Lecture Notes in Computer Science*, pages 27–40. Springer Berlin / Heidelberg, 2006. ISBN 978-3-540-68379-7. URL http://dx.doi.org/10.1007/11951957_3. → pages 3

[5] A. Duttagupta, A. Bishnu, and I. Sengupta. Maximal breach in wireless sensor networks: Geometric characterization and algorithms. In M. Kutylowski, J. Cichon, and P. Kubiak, editors, *Algorithmic Aspects of Wireless Sensor Networks*, volume 4837 of *Lecture Notes in Computer Science*, pages 126–137. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-77870-7. URL http://dx.doi.org/10.1007/978-3-540-77871-4_12. → pages 3

[6] H. Edelsbrunner and E. P. Mucke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph*, 9: 66–104, 1990. → pages 147

[7] U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4): 634–652, July 1998. ISSN 0004-5411. doi:10.1145/285055.285059. URL http://doi.acm.org/10.1145/285055.285059. → pages 7

[8] S. Kumar, T. H. Lai, and J. Balogh. On k-coverage in a mostly sleeping sensor network. In *Proceedings of the 10th annual international conference on Mobile computing and networking*, MobiCom '04, pages 144–158, New York, NY, USA, 2004. ACM. ISBN 1-58113-868-7. doi:10.1145/1023720.1023735. URL http://doi.acm.org/10.1145/1023720.1023735. → pages 1, 2

[9] 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. → pages 5, 6

[10] S. Kumar, T. H. Lai, and A. Arora. Barrier coverage with wireless sensors. *Wireless Networks*, 13:817–834, 2007. ISSN 1022-0038. URL http://dx.doi.org/10.1007/s11276-006-9856-0. 10.1007/s11276-006-9856-0. → pages 5, 6

[11] 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. doi:10.1145/1374618.1374673. URL http://doi.acm.org/10.1145/1374618.1374673. → pages 4

[12] L. Liu, X. Zhang, and H. Ma. Minimal exposure path algorithms for directional sensor networks. In *Global Telecommunications Conference,*

2009. GLOBECOM 2009. IEEE, pages 1–6, 30 2009-dec. 4 2009. doi:10.1109/GLOCOM.2009.5426228. → pages 3, 4

[13] S. Megerian, F. Koushanfar, G. Qu, G. Veltri, and M. Potkonjak. Exposure in wireless sensor networks: Theory and practical solutions. *Wireless Networks*, 8:443–454, 2002. ISSN 1022-0038. URL http://dx.doi.org/10.1023/A:1016586011473. 10.1023/A:1016586011473. → pages 3

[14] S. Megerian, F. Koushanfar, M. Potkonjak, and M. Srivastava. Worst and best-case coverage in sensor networks. *Mobile Computing, IEEE Transactions on*, 4(1):84 – 92, jan.-feb. 2005. ISSN 1536-1233. doi:10.1109/TMC.2005.15(410)4. → pages 3

[15] 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. doi:10.1109/INFCOM.2001.916633. → pages 1, 3

[16] 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. doi:10.1145/381677.381691. URL http://doi.acm.org/10.1145/381677.381691. → pages 3

[17] D. Mehta, M. Lopez, and L. Lin. Optimal coverage paths in ad-hoc sensor networks. In *Communications, 2003. ICC '03. IEEE International Conference on*, volume 1, pages 507 – 511 vol.1, may 2003. doi:10.1109/ICC.2003.1204228. → pages 3

[18] H. Rosenberger. Order-k voronoi diagrams of sites with additive weights in the plane. *Algorithmica*, 6:490–521, 1991. → pages 90

[19] D. Tian and N. D. Georganas. A coverage-preserving node scheduling scheme for large wireless sensor networks. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, WSNA '02, pages 32–41, New York, NY, USA, 2002. ACM. ISBN 1-58113-589-0. doi:10.1145/570738.570744. URL http://doi.acm.org/10.1145/570738.570744. → pages 2

[20] K.-C. Tseng and D. Kirkpatrick. On barrier resilience of sensor networks. In T. Erlebach, S. Nikoletseas, and P. Orponen, editors, *Algorithms for Sensor Systems*, volume 7111 of *Lecture Notes in Computer Science*, pages 130–144. Springer Berlin / Heidelberg, 2012. ISBN 978-3-642-28208-9. URL http://dx.doi.org/10.1007/978-3-642-28209-6_11. → pages 5, 6

[21] K.-C. R. Tseng. Resilience of wireless sensor networks. Master's thesis, The University of British Columbia, 2011. → pages 5

[22] G. Veltri, Q. Huang, G. Qu, and M. Potkonjak. Minimal and maximal exposure path algorithms for wireless embedded sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, SenSys '03, pages 40–50, New York, NY, USA, 2003. ACM. ISBN 1-58113-707-9. doi:10.1145/958491.958497. URL http://doi.acm.org/10.1145/958491.958497. → pages 3

[23] Y.-C. Wang, C.-C. Hu, and Y.-C. Tseng. Efficient placement and dispatch of sensors in a wireless sensor network. *Mobile Computing, IEEE Transactions on*, 7(2):262–274, feb. 2008. ISSN 1536-1233. doi:10.1109/TMC.2007.70708. → pages 2

[24] S. Yuan, S. Varma, and J. Jue. Minimum-color path problems for reliability in mesh networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 4, pages 2658 – 2669 vol. 4, march 2005. doi:10.1109/INFCOM.2005.1498549. → pages 5, 7, 100

[25] H. Zhang and J. C. Hou. Maintaining sensing coverage and connectivity in large sensor networks. *Ad Hoc and Sensor Wireless Networks*, 1:89–124, 2005. → pages 2

Appendix A

Proofs Regarding Forbidden Configurations

A.1 Folded Forbidden Configurations

In this appendix, we prove Theorem 5.2.3, restated here:

Theorem A.1.1 *Forbidden Configurations are unrealizable.*

As previously defined, Forbidden Configurations are every timeline configuration that is a result of some sequence of transformations from the ten shown in Figure A.1. Each black line represents a subchain of vertices of a timeline. The colours red and blue represent two distinct sensors, and coloured shorter lines are drawn above a vertex to show that it is within a positive intersection of a sensor associated with the colour, and drawn below for negative intersections. In each of these ten timeline subchains, a timeline configuration is formed by the states of the vertices. Without loss of generality, let the sensor associated with the red intervals be α and the sensor associated with the blue intervals be β.

A timeline configuration is considered realizable if there exists a timeline that both contains the timeline configuration and represents a peripheral traversal that goes through an arrangement of sensors subject to Axiom 5.2.1 and Axiom 5.2.2. As previously stated, any supersequence of an unrealizable timeline configuration

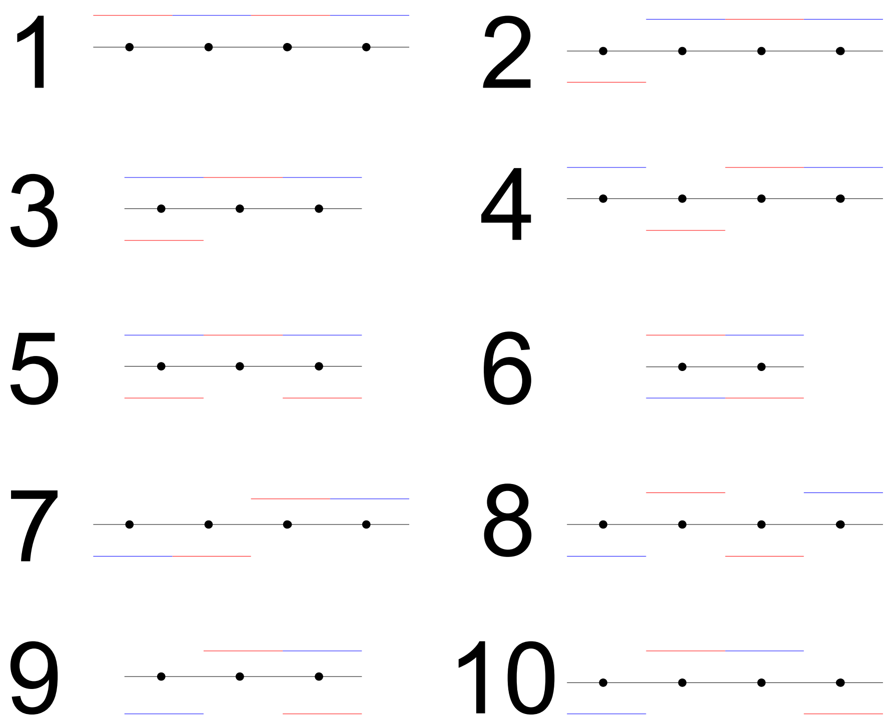

Figure A.1: 10 Forbidden Configurations

is also unrealizable. Thus we can focus on proving only the smallest sequences are unrealizable.

Let P be a peripheral traversal through some arrangement $\mathtt{A}$ subject to Axiom 5.2.1 and Axiom 5.2.2. The goal is to prove that P cannot contain any of the Forbidden Configurations. Note that in order to avoid confusion between intersections of pairs of sensors and sensor intersections made by P, all sensor intersections made by P are henceforth referred to as *sensor visits*.

Let arrangement $\mathtt{A}'$ be a duplicate of $\mathtt{A}$ but with the detection region of each sensor trimmed to include only its sensor visits and the subregion that a path could visit without creating any interior sensor visits, given that it already has all the sensor visits of P. As previously defined, in the absence of interior sensor visits, every sensor visit of a multiply-visited sensor has exactly one path-intersected sensor-boundary-walk and one non-path-intersected sensor-boundary-walk. Thus to derive each sensor in $\mathtt{A}'$, it suffices to take the boundary of the original sensor in $\mathtt{A}$ and replace each non-path-intersected sensor-boundary-walk with its associated sensor visit. Observe that this retains all the sensor visits of P in the same orientations and order, but with all sensor visits will now go along the boundaries of sensors in $\mathtt{A}'$.

Lemma A.1.2 *Axiom 5.2.1 holds for* $\mathtt{A}'$.

Proof Axiom 5.2.1 requires that each sensor in $\mathtt{A}'$ be associated with a single detection region on the two dimensional plane that contains no holes. In other words, for any two points p_1 and p_2 and any sensor s_i, if the points are both inside or both outside s_i, there exists a path from p_1 to p_2 that does not intersect the boundary of s_i.

By definition, P cannot intersect itself, so by construction it is impossible to create any holes in the detection region since that would require a self-intersecting sensor visit. Furthermore, no two sensor visits of the same sensor can intersect, so replacing each non-path-intersected sensor-boundary-walk with its associated sensor visit cannot result in a self-intersecting boundary. Lastly, observe that to split the detection region into multiple disconnected subregions, a non-path-intersected sensor-boundary-walk that fully covers a path-intersected sensor-boundary-walk is required. This is a contradiction since the non-path-intersected sensor-boundary-walk must clearly be path-intersected if it fully covers a path-intersected sensor-boundary-walk. Consequently, it is also impossible for the detection region to be split into multiple disconnected subregions. Thus it must be the case that each sensor in $\mathtt{A}'$ is associated with a single detection region that contains no holes. ∎

Lemma A.1.3 *Axiom 5.2.2 holds for* $\mathtt{A}'$.

Proof Axiom 5.2.2 requires that $\mathtt{A}'$ is pairwise boundary-segregated. In other words, every pair of sensors s'_i and s'_j in $\mathtt{A}'$ have a boundary-segregated intersection. This means s'_i and s'_j are boundary-isolated with respect to every intersection region for the pair, or no intersection region for the pair exists in the sense that s'_i and s'_j are associated with completely disjoint regions in $\mathtt{A}'$.

Now consider an intersection region $\mathtt{I}'_{i,j}$ for s'_i and s'_j. By definition, this is a region that is fully covered by s'_i and fully covered by s'_j. Then since the detection region of s'_i and s'_j are subregions of the sensors s_i and s_j in $\mathtt{A}$, it must be the case that $\mathtt{I}'_{i,j}$ is a subregion of some intersection region $\mathtt{I}_{i,j}$ for s_i and s_j. In addition, any point on the boundary of $\mathtt{I}'_{i,j}$ must either be on the boundary of $\mathtt{I}_{i,j}$ or on the path P, since the non-path-intersected sensor-boundary-walks are replaced by sensor visits in the construction process. Furthermore, as by definition s_i and s_j fully cover $\mathtt{I}_{i,j}$, each point in $\mathtt{I}_{i,j}$ that is also on P must be on the boundary of both s'_i and s'_j.

Consequently, when the non-path-intersected sensor-boundary-walks are replaced by their associated sensor visits, any point on the boundary of $\mathtt{I}'_{i,j}$ but not on the boundary of $\mathtt{I}_{i,j}$ must actually be on the boundary of both s'_i and s'_j. This allows us to prove that both s'_i and s'_j are boundary-isolated with respect to $\mathtt{I}'_{i,j}$ in a proof by contradiction as follows:

Let $\mathtt{P}'_i$ be the set of points on the boundary of $\mathtt{I}'_{i,j}$ that are also on the boundary of s'_i. Let $\mathtt{P}'_j$ be the corresponding set for s'_j, and let $\mathtt{P}_i$ and $\mathtt{P}_j$ be the corresponding sets for $\mathtt{I}_{i,j}$ and s_i and s_j.

By definition, if s'_i is not boundary-isolated with respect to $\mathtt{I}'_{i,j}$, there must be two points p'_1 and p'_2 inside the set $(\mathtt{P}'_i - \mathtt{P}'_j)$ such that it is not possible to walk from p'_1 to p'_2 using only points in $\mathtt{P}'_i$. However, we just showed that any points on the boundary of $\mathtt{I}'_{i,j}$ that are not on the boundary of $\mathtt{I}_{i,j}$ must be on the boundary of both s'_i and s'_j, so by definition these points cannot be inside the set $(\mathtt{P}'_i - \mathtt{P}'_j)$. Thus p'_1 and p'_2 must be points on the boundary of $\mathtt{I}_{i,j}$ and in $(\mathtt{P}_i - \mathtt{P}_j)$.

Then since Axiom 5.2.2 holds for $\mathtt{A}$, it must have been possible to walk from p'_1 to p'_2 using only points in $\mathtt{P}_i$. However, noting again that any points on the boundary of $\mathtt{I}'_{i,j}$ that are not on the boundary of $\mathtt{I}_{i,j}$ must be on the boundary of both s'_i and s'_j, it is impossible for any part of this walk to be replaced by a new boundary part that cannot be traversed using only points in $\mathtt{P}'_i$. Thus s'_i must be boundary-isolated with respect to $\mathtt{I}'_{i,j}$.

A similar argument shows the same result for s'_j, which means s'_i and s'_j must have a boundary-segregated intersection. Thus this proves that $\mathtt{A}'$ is pairwise boundary-segregated. ∎

Consequently, the problem is reduced to the following:

Let P be a peripheral traversal through some arrangement $\mathtt{A}$ subject to Axiom 5.2.1 and Axiom 5.2.2 where all sensor visits must go along the boundaries of sensors. The goal is to prove that P cannot contain any of the Forbidden Configurations.

To achieve this goal, first observe that for every pair of visits to the same sensor s_i, the two-dimensional plane can be split into 3 non-overlapping regions:

The Sensor Region The detection region associated with s_i.

The Interior Region The region bounded by the boundary of s_i and the subpath from the end of the first visit to the start of the second visit.

The Exterior Region The rest of the two-dimensional plane.

Note that by Axiom 5.2.1 and the fact that the path is a non-self-intersecting acyclic traversal, each of these defines a single region that contains no holes. We then make the following observations:

Let v_1 and v_2 be two arbitrary visits to some arbitrary sensor s_i where v_1 is earlier on the path P than v_2. Let $\text{Region}_{\text{sensor}}$, $\text{Region}_{\text{interior}}$, and $\text{Region}_{\text{exterior}}$ denote the sensor region, interior region, and exterior region respectively for the pair of visits v_1 and v_2. Finally, let P_{before}, P_{between}, and P_{after} denote the subpaths of P before v_1, between v_1 and v_2, and after v_2 respectively.

Observation A.1.4 *For each subpath P_{before} and P_{after}, if any point on the subpath is in* $\text{Region}_{\text{interior}}$, *no point on the subpath is in* $\text{Region}_{\text{exterior}}$.

Proof Since P is a peripheral traversal, it cannot intersect itself, and thus cannot connect points in $\text{Region}_{\text{interior}}$ and $\text{Region}_{\text{exterior}}$ by intersecting the subpath between v_1 and v_2. Consequently, the only way for P to connect points in $\text{Region}_{\text{interior}}$ to $\text{Region}_{\text{exterior}}$ is to intersect $\text{Region}_{\text{sensor}}$. However, for either P_{before} and P_{after} to do so would inevitably result in an interior visit to s_i, which is a contradiction because peripheral traversals by definition do not contain any interior visits. ∎

Let s_j be some sensor that neither has any visit v_3 that is the same orientation as v_1 and fully covers v_1, nor has any visit v_4 that is the same orientation as v_2 and fully covers v_2.

Observation A.1.5 *If part of the boundary of s_j is in* $\text{Region}_{\text{interior}}$*, no part of the boundary of s_j is in* $\text{Region}_{\text{exterior}}$.

Proof Assume s_j has part of its boundary in $\text{Region}_{\text{interior}}$ and part of its boundary in $\text{Region}_{\text{exterior}}$. By Axiom 5.2.1, these two parts must be connected together by the rest of the boundary of s_j. Furthermore, we know by construction that the boundary of s_j may go through the path at any point, only going along the path at the visits instead. This includes the subpath from the end of v_1 to the start of v_2. Consequently, the boundary of s_j must go through $\text{Region}_{\text{sensor}}$.

Now observe that if the subpath associated with v_1 is fully covered by the boundary of s_j going in between $\text{Region}_{\text{interior}}$ and $\text{Region}_{\text{exterior}}$, there must exist a visit v_3 to s_j that is of the same orientation as v_1 and fully covers v_1, contradicting the initial selection requirements for s_j. Thus the boundary of s_j going in between $\text{Region}_{\text{interior}}$ and $\text{Region}_{\text{exterior}}$ may not fully cover the subpath associated with v_1. By a similar argument, the same holds true for v_2. Then observe that this inevitably results in an intersection region $\mathtt{I}_{i,j}$ for s_i and s_j such that neither s_i nor s_j are boundary-isolated with respect to $\mathtt{I}_{i,j}$. Consequently, s_i and s_j do not have a boundary-segregated intersection.

However, this is a contradiction of Axiom 5.2.2. Thus it must be the case that s_j cannot have boundaries in both $\text{Region}_{\text{interior}}$ and $\text{Region}_{\text{exterior}}$. ∎

Now consider each of the ten configurations shown in Figure A.1. Let $\text{Region}_{\text{sensor}}$, $\text{Region}_{\text{interior}}$, and $\text{Region}_{\text{exterior}}$ denote the sensor region, interior region, and exterior region respectively for the first two visits of α.

First, observe that in each configuration, β does not have any visit which has the same orientation and fully covers either of these two visits to α. We will show that the configuration is unrealizable by proving the visits to β require β to have boundaries in both $\text{Region}_{\text{interior}}$ and $\text{Region}_{\text{exterior}}$, contradicting Observation A.1.5.

Then note that by Observation A.1.4, the path after the second visit of α cannot have points in both $\text{Region}_{\text{interior}}$ and $\text{Region}_{\text{exterior}}$. In fact, given a pair of visits to α with fixed orientations, it is possible to have paths which avoid $\text{Region}_{\text{interior}}$ after the pair and paths which avoid $\text{Region}_{\text{exterior}}$ after the pair. Let case *I* denote the case where *P* avoids $\text{Region}_{\text{interior}}$ after the pair, and let case *II* denote the case

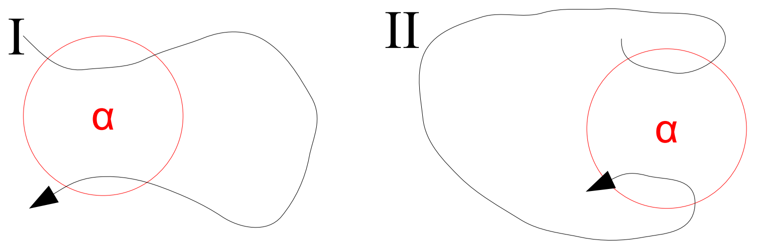

Figure A.2: The two cases for a pair of visits to α with the same orientation.

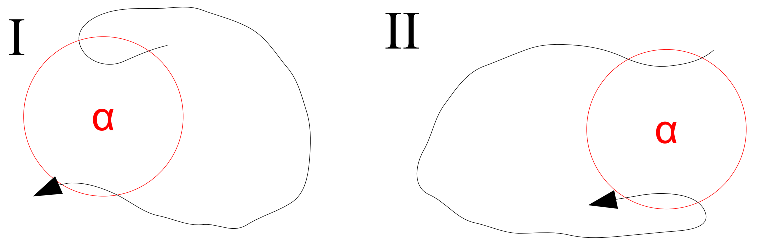

Figure A.3: The two cases for a pair of visits to α with opposite orientations.

where $\text{Region}_{\text{exterior}}$ is avoided instead. These cases are illustrated in Figure A.2 and Figure A.3 for a disk sensor α before the detection region is trimmed. However note that this disk shape is not required for our arguments to hold.

Note that due to the inherent symmetry between case *I* and case *II*, it suffices to just consider case *I* for each configuration, since similar arguments would achieve the same results for case *II*. Thus for each configuration, we shall assume that P avoids $\text{Region}_{\text{interior}}$ after the second visit of α in the configuration.

Now consider configuration 1. There is a positive visit of β in between the two visits of α, and by definition it is associated with a counter-clockwise path-intersected sensor-boundary-walk. Then given that the second visit of α is positive and P_{after} avoids $\text{Region}_{\text{interior}}$, β must have part of its boundary in $\text{Region}_{\text{interior}}$. However, there is also a visit of β after the second visit of α, and by again noting that P_{after} avoids $\text{Region}_{\text{interior}}$, this visit cannot be in $\text{Region}_{\text{interior}}$. Furthermore, this visit of β is not covered by any visit of α, thus this visit of β must be in $\text{Region}_{\text{exterior}}$. Consequently, β must have part of its boundary in $\text{Region}_{\text{interior}}$ and part of its boundary in $\text{Region}_{\text{exterior}}$. Thus we have shown that P containing configuration 1 would contradict Observation A.1.5, proving that configuration 1 is unrealizable.

Furthermore, observe that the above argument does not depend on the orientation of the second visit of β or the orientation of the first visit of β. In other words, the configuration remains unrealizable even if the orientations of either of those visits are reversed. Thus via the transformation methods presented in Section 5.2, this also proves that configurations 2 and 10 are unrealizable.

Next consider configuration 3. The first positive visit of β is overlapping the first negative visit of α, so by considering the counter-clockwise path-intersected sensor-boundary-walk and the second visit of α as before, it must be the case that β has part of its boundary in $\text{Region}_{\text{interior}}$. However, the second visit of β is in the subpath P_{after} which avoids $\text{Region}_{\text{interior}}$, and not covered by α. Thus it must be in $\text{Region}_{\text{exterior}}$, meaning β must have part of its boundary in $\text{Region}_{\text{interior}}$ and part of its boundary in $\text{Region}_{\text{exterior}}$, which proves configuration 3 is unrealizable.

Observe that this argument does not depend on the orientation of the second visit of β. Thus via transformation arguments configuration 9 is also unrealizable. Furthermore, even if the second visit of β was a positive visit which overlapped a negative visit of α such as in configuration 5, observe that it would still require β to have part of its boundary in $\text{Region}_{\text{exterior}}$ since the boundary of β cannot intersect the path associated with the negative visit of α and by Axiom 5.2.1 the boundaries of β must be connected. Thus configuration 5 is also unrealizable.

Now consider configuration 4. Given that P_{after} avoids $\text{Region}_{\text{interior}}$ and the pair of visits to α have opposite orientations, P_{before} must avoid $\text{Region}_{\text{exterior}}$. Then since the two visits of β are not covered by α visits and one is in P_{before} while the other is in P_{after}, β must have boundaries in $\text{Region}_{\text{interior}}$ and $\text{Region}_{\text{exterior}}$. Thus configuration 4 is also unrealizable. Observe that this argument does not depend on the orientation of the two visits of β, so via transformation arguments, configurations 7 and 8 are also unrealizable.

This leaves only configuration 6. Here the two visits of α have opposite orientations, and each is overlapped by a visit of β of opposite orientation. Given that P_{after} avoids $\text{Region}_{\text{interior}}$, it must then be the case that the first visit of β requires β to have part of its boundary in $\text{Region}_{\text{exterior}}$, while the second visit of β requires β to have part of its boundary in $\text{Region}_{\text{interior}}$. Thus configuration 6 is

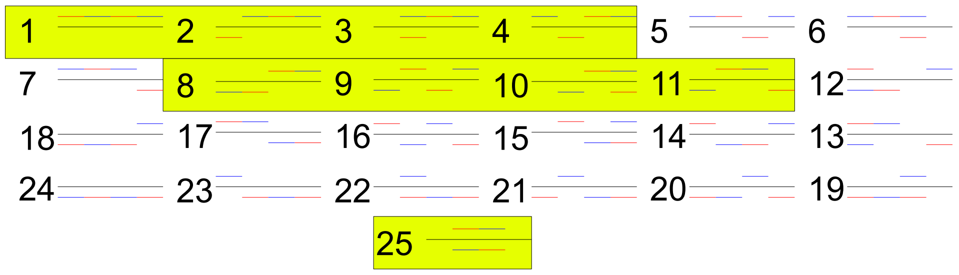

Figure A.4: All 25 folded forbidden configurations in the 2-visit scenario.

also unrealizable.

In conclusion, we have shown that these 10 configurations and all their transformations are unrealizable, proving Theorem 5.2.3.

A.2 Unfolded Forbidden Configurations

In this appendix, we will prove that for any timeline that does not contain Forbidden Configurations, the corresponding unfolded timeline does not contain the Unfolded Forbidden Configuration $\alpha_{-\beta}\beta_{-\alpha}\alpha_{-\beta}\beta_{-\alpha}$ of the 2-visit scenario. This is achieved via illustrating that all possible timeline configurations on the folded timeline that could unfold to form the Unfolded Forbidden Configuration are Forbidden Configurations, or supersequences of Forbidden Configurations. As before, it suffices to only consider the smallest sequences.

Starting with the base cases, in the 2-visit scenario there are 25 distinct folded timeline configurations that are the smallest sequences of visits that can form $\alpha_{-\beta}\beta_{-\alpha}\alpha_{-\beta}\beta_{-\alpha}$ in the unfolded timeline. These are shown in Figure A.4. Visits do not actually have to be consecutive, but they must be in the order presented. To distinguish these configurations from the ones in Section 5.2, we shall denote these configurations with a subscript u. Many of these configurations are extremely similar or symmetrical, and we can in fact show that by taking the mirror image and swapping the labels, clearly configurations 13_u to 24_u are transformations of configurations 1_u to 12_u, and thus we can ignore them and focus on showing proofs for the remaining 13 configurations. Furthermore, reversing the path and taking the mirror image reveals that 5_u to 7_u are transformations of 2_u to 4_u, and reversing the path shows that 12_u is a transformation of 10_u. Consequently, it suffices to consider

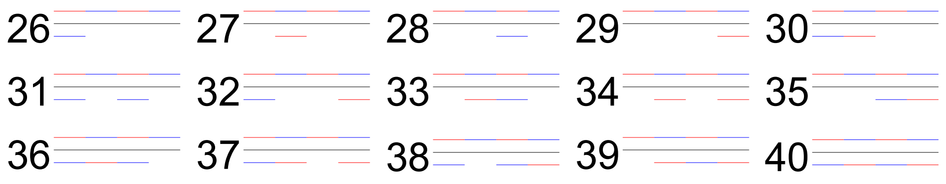

Figure A.5: All 15 extensions of configuration 1_u.

only configurations 1_u to 4_u, 8_u to 11_u, and 25_u, as shown in yellow in Figure A.4. To show that they are Forbidden Configurations, observe that they are identical to configurations 1 to 4, 7 to 10, and 6 respectively.

Then, to show that $\alpha_{-\beta}\beta_{-\alpha}\alpha_{-\beta}\beta_{-\alpha}$ would never appear in the unfolded timeline of timeline without Forbidden Configurations even when there are multiple visits per sensor, we have to further extend these base cases to include all possible situations with multiple visits. However, we don't have to show this for any supersequence of the base cases, since by definition a supersequence would contain a subsequence. In other words, the extra visits we add must not be in locations different from the set of locations presented by the base cases. Instead, they must be added to the base cases such that they are simultaneous with the old visits to not be a supersequence of it, while maintaining that unfolding the timeline can still result in $\alpha_{-\beta}\beta_{-\alpha}\alpha_{-\beta}\beta_{-\alpha}$. Consequently, this means for each visit in each base case, there is a possible extension by adding a visit of the opposite orientation to the opposite sensor at the same location.

Figure A.5 shows the 15 possible extensions of configuration 1_u. Out of these extensions, only configurations 31_u and 34_u are not supersequences of any of the 25 base cases. Furthermore configuration 31_u is a transformation of configuration 34_u by reversing the path, swapping the labels and flipping the arrangement. Thus it suffices to show that configuration 34_u is a Forbidden Configuration, or supersequence of Forbidden Configuration. This is achieved by observing that configuration 34_u is a supersequence of configuration 5.

Figure A.6 shows the 15 possible extensions of configuration 2_u. Out of these extensions, only 49_u and 52_u are not supersequences of any of the 25 base cases or their transformations. However, 49_u and 52_u are also supersequences of a transformation of configuration 5.

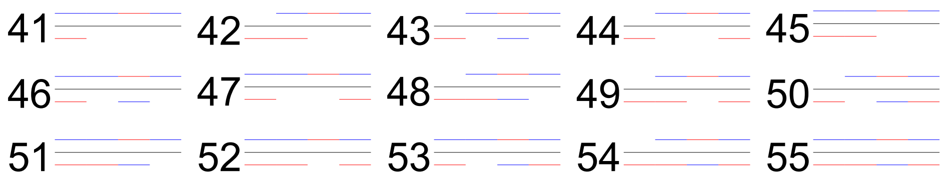

Figure A.6: All 15 extensions of configuration 2_u.

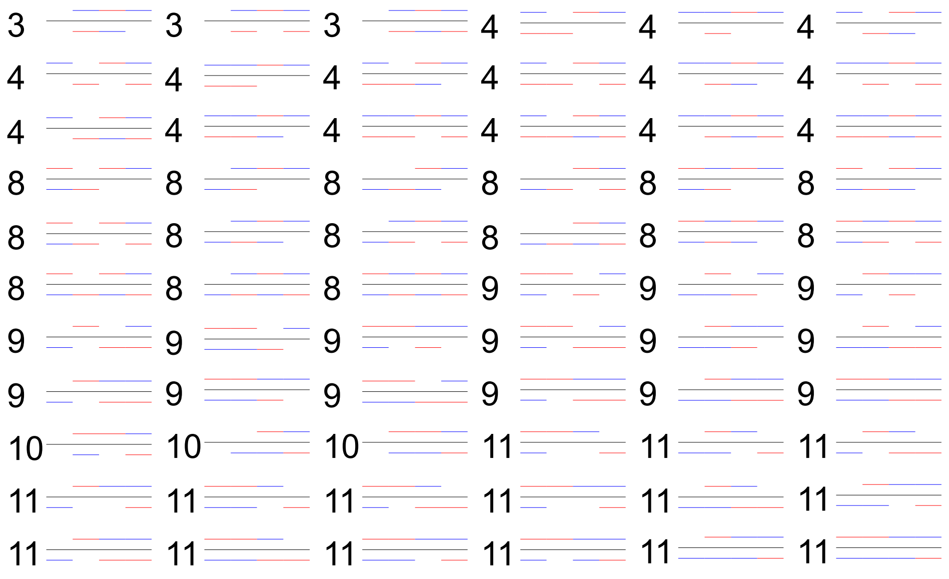

Figure A.7: All remaining extensions of the other base cases.

Figure A.7 shows all the remaining possible extensions. Note that configuration 25_u cannot be extended. Every one of them is supersequence of either a transformation or duplicate of a previously handled case. Thus it is proven that it is impossible to get the Unfolded Forbidden Configuration of $\alpha_{-\beta}\beta_{-\alpha}\alpha_{-\beta}\beta_{-\alpha}$ in the unfolded timeline of any timeline that does not contain Forbidden Configurations.

Appendix B

Proofs Regarding Boolean Matrices

In this appendix, we examine two-dimensional boolean matrices, whose cells either contain ones or zeros. Cells containing ones are referred to as marked cells, while cells containing zeros are considered unmarked. In particular, let M_{f1} and M_{f2} be the two 2 by 4 matrices of alternatingly marked cells illustrated in Table B.1 and Table B.2. These matrices will henceforth be referred to as Forbidden Submatrices.

As before, let a Splittable Column be defined as a column in the matrix other than the first or last column such that each row of the matrix is in at least one of the following states:

- The row has no marked cells to the right of the column.
- The row has no marked cells to the left of the column.
- The row has a marked cell in the column.
- The row has marked cells in the first and last columns.

1	0	1	0
0	1	0	1

Table B.1: First of the two possible 2 by 4 matrices of alternatingly marked cells representing the Forbidden Configuration.

0	1	0	1
1	0	1	0

Table B.2: Second of the two possible 2 by 4 matrices of alternatingly marked cells representing the Forbidden Configuration.

The aim of this appendix is to prove the following:

Theorem B.0.1 *Any matrix that does not have Table 5.2 or Table 5.3 as submatrices must either have at most two columns or a Splittable Column.*

Let M_∞ be an arbitrary non-empty boolean matrix with more than two columns and the following constraints:

- M_∞ must not contain a Forbidden Submatrix.
- Every column other than the first or last one must have at least one row that is not in the first three states. In other words, it has at least one row that has marked cells on both sides but an unmarked cell in the column.

We will then argue the following:

Theorem B.0.2 *There must be some row which has both the first and last columns marked.*

Let L and R be the number of columns and rows respectively, so the columns are numbered from 1 to L from left to right and the rows are numbered from 1 to R from top to bottom. Let a gap be defined as the unmarked cells between a pair of marked cells in the same row without any marked cells in between them. As rows can contain several marked cells, each row can contain several gaps, which generates confusion when we try to denote the start and ends of different gaps in a row.

Thus let us construct a new matrix M_∞^*, which duplicates every row in M_∞ arbitrarily many times such that each gap has one duplicate row representing it. In other words, for each gap in the original matrix M_∞, we add a copy of the row containing the gap to M_∞^*. Note that by this construction, each row in M_∞^* will be identical to some row in M_∞, and vice versa. As before, let L^* and R^* denote the number of columns and rows for M^*_∞, and note that $L^* = L$ since we are duplicating the rows in M_∞ without modifying them in any way.

Lemma B.0.3 *M^*_∞ must have the same two constraints as M_∞, and if M^*_∞ has a row which has both the first and last columns marked, M_∞ must also have a row which has both the first and last columns marked.*

Proof Assume M^*_∞ contains a Forbidden Submatrix. As illustrated in Table 5.2 and Table 5.3, the two rows in this submatrix cannot be duplicates. Thus it must be the case that M_∞ contains at least one copy of each row. Furthermore, reordering the two rows still results in a Forbidden Submatrix. Thus it must be the case that M_∞ also contains a Forbidden Submatrix. Since M_∞ must not contain a Forbidden Submatrix, it must then be the case that M^*_∞ does not contain a Forbidden Submatrix as well.

The second constraint requires that every column has at least one row with an unmarked cell in it and marked cells to the left and right. As each row in M_∞ is identical to some row in M^*_∞, every column that had at least one such row in M_∞ must still have at least one such row in M^*_∞.

By definition, each row in M^*_∞ is identical to some row in M_∞. If M^*_∞ has a row which has both the first and last columns marked, this row must be identical to some row in M_∞. Thus it must be that M_∞ has a row which has both the first and last columns marked. ∎

As every gap is now represented by one duplicate row, we can define the following notation without confusion:

- r^*_i denotes the i-th row in M^*_∞.
- **First**(r^*_i) denotes the number of the leftmost column that contains a marked cell in r^*_i.
- The start of the gap represented by r^*_i, denoted $G_s(r^*_i)$, is the number of the rightmost column that contains a marked cell in r^*_i before the gap.
- The end of the gap represented by r^*_i, denoted $G_e(r^*_i)$, is the number of the leftmost column that contains a marked cell in r^*_i after the gap.

- $\textbf{Last}(r_i^*)$ denotes the number of the rightmost column that contains a marked cell in r_i^*.

However, there is still insufficient structure to easily prove it contains a row which has both the first and last columns marked. Instead, we shall use the second constraint to derive another more structured matrix M'_∞ with L' and R' denoting the number of columns and rows, and with $L' = L$. As before, let r'_i denote the i-th row in M'_∞. The second constraint states that every column other than the first or last one must have at least one unmarked cell that has marked cells on the same row on the left and right. In terms of gaps, this means every column other than the first or last one must have some row which has a gap including that column. Thus there must exist a set of rows in M^*_∞ whose gaps combine to cover the entire matrix.

In particular, we want M'_∞ to be minimal and well-structured in some sense, so for each choice of gaps, we want to choose the gaps which span as much extra distance as possible. In other words, the procedure we shall use to construct M'_∞ is:

1. Consider column 2: The second constraint states that every column other than the first or last one must have at least one unmarked cell that has marked cells on the same row on the left and right. This means there must exist a non-empty set of rows in M^*_∞ that have $\textbf{First} = G_s = 1 < 2 < G_e \leq \textbf{Last}$. From this set, choose the row with the largest G_e to be r'_1 in M'_∞.

2. Consider the row in M^*_∞ that was just chosen as r'_1: unless there is some row which has both the first and last columns marked, it must be that $G_e(r'_1) \leq \textbf{Last}(r'_1) < L$.

3. Consider column $G_e(r'_1)$: As before, the second constraint states that every column other than the first or last one must have at least one unmarked cell that has marked cells on the same row on the left and right, so there must exist a non-empty set of rows in M^*_∞ that have $\textbf{First} \leq G_s < G_e(r'_1) < G_e \leq \textbf{Last}$. From this set, choose the row with the largest G_e to be r'_2 in M'_∞.

4. While the last row chosen does not have $\textbf{Last} = G_e = L$, repeat the previous step using the last row instead of r'_1 to add more rows to M'_∞.

The resulting structure of M'_∞ can be stated more formally as follows:

- **First**$(r'_1) = G_s(r'_1) = 1$ since r'_1 is chosen from a set of rows with **First** $=$ $G_s = 1$.
- **Last**$(r'_{R'}) = G_e(r'_{R'}) = L$ since the procedure only stops when this is true.
- $\forall_{1<i\leq R'} : G_s(r'_i) < G_e(r'_{i-1}) < G_e(r'_i)$ since r'_i is chosen from a set of rows with **First** $\leq G_s < G_e(r'_i) < G_e \leq$ **Last**.
- $\forall_{1<i\leq R'}, \forall_{1\leq j\leq R^*}$: if $G_s(r^*_j) < G_e(r'_{i-1}) < G_e(r^*_j)$ then $G_e(r^*_j) \leq G_e(r'_i)$, since we choose each r'_i as the row with the largest G_e when there are multiple candidates.

In addition, there is one more claim we can make as a result of the fact that we always choose the row with the largest G_e from the set of candidates:

Lemma B.0.4 $\forall_{1<i\leq R'} : G_s(r'_{i-1}) < G_s(r'_i)$

Proof This can be shown via a proof by contradiction. If this lemma is false, there must be some row r'_j such that $G_s(r'_{j-1}) \geq G_s(r'_j)$. Since we choose each row r'_i from the set which have **First** $\leq G_s < G_e(r'_{i-1}) < G_e \leq$ **Last**, it must be the case that $G_s(r'_{j-1}) < G_e(r'_{j-2}) < G_e(r'_{j-1})$ and $G_e(r'_{j-1}) < G_e(r'_j)$.

Combined, we get $G_s(r'_j) \leq G_s(r'_{j-1}) < G_e(r'_{j-2}) < G_e(r'_{j-1}) < G_e(r'_j)$. This means $G_s(r'_j) < G_e(r'_{j-2}) < G_e(r'_j)$. Then by definition, r'_j must have been part of the set of candidate rows when choosing r'_{j-1}. Furthermore, we have that $G_e(r'_{j-1}) < G_e(r'_j)$ and we know that we select the row with the largest G_e, so r'_j must have been chosen instead. Thus we reach a contradiction. ▮

Thus, M'_∞ is some submatrix of M^*_∞ that has the following properties:

- **First**$(r'_1) = G_s(r'_1) = 1$
- **Last**$(r'_{R'}) = G_e(r'_{R'}) = L$
- $\forall_{1<i\leq R'} : G_s(r'_{i-1}) < G_s(r'_i) < G_e(r'_{i-1}) < G_e(r'_i)$

Lemma B.0.5 *M'_∞ must not contain a Forbidden Submatrix, and if M'_∞ has a row which has both the first and last columns marked, M^*_∞ must also have a row which has both the first and last columns marked.*

1	0	1
0	1	0

Table B.3: First Three-Quarters of a 2 by 4 matrix of alternatingly marked cells.

Proof Assume M'_∞ contains a Forbidden Submatrix. The two rows involved in the submatrix must be present in M^*_∞ as well, and reordering them still results in a Forbidden Submatrix. Thus M^*_∞ must contain a Forbidden Submatrix as well. However we know from Lemma B.0.3 that M^*_∞ does not contain a Forbidden Submatrix, thus it must be that M'_∞ does not contain a Forbidden Submatrix as well.

The set of rows in M'_∞ is a subset of the rows in M^*_∞, so by definition each row in M'_∞ is identical to some row in M^*_∞. If M'_∞ has a row which has both the first and last columns marked, this row must be identical to some row in M^*_∞. Thus M^*_∞ must have a row which has both the first and last columns marked as well. ∎

Given M'_∞, we now have a well-structured set of rows, but it is still difficult to prove that it has a row which has both the first and last columns marked. Instead, we shall construct another submatrix $\hat{M}_\infty$ using a subset of the rows in M'_∞. As before, let $\hat{L}$ and $\hat{R}$ denote the number of columns and rows, and let $\hat{r}_i$ denote the i-th row in $\hat{M}_\infty$.

This time, instead of exploiting the second constraint, we shall exploit the first constraint and the structure of the Forbidden Submatrices. Let M_{f3} denote the matrix that is the first three-quarters of M_{f1}, as illustrated in Table B.3. We shall construct $\hat{M}_\infty$ as follows:

1. Choose $\hat{r}_1$ as the row in M'_∞ which has **First** $= 1$ and the largest G_e.

2. $\forall_{1<i}$: Choose $\hat{r}_i$ as the row with the largest G_e in M'_∞ while under the following constraints:

   - $G_e(\hat{r}_i) > G_e(\hat{r}_{i-1})$
   - The submatrix formed by columns 1 to $G_e(\hat{r}_i) - 1$ inclusive and by rows $\hat{r}_{i-1}$ and $\hat{r}_i$ contains M_{f3} as a submatrix.

3. Keep adding rows via the previous step until there are no more rows which satisfy the constraints.

Note that because $\textbf{First}(r_1') = 1$, $\hat{M}_\infty$ must be non-empty. Also, it must be the case that $\hat{L} = L$, since none of the rows in M_∞ have been modified, only duplicated and possibly re-ordered. Finally, because of the use of M_{f3}, we shall associate each row $\hat{r}_i$ with a column $\hat{c}_i$, where $\hat{c}_i$ is the column that forms the second column in the M_{f3} submatrix in rows $\hat{r}_i$ and $\hat{r}_{i-1}$. In the case of the first row, let $\hat{c}_1$ denote the first column instead. By this definition, each column $\hat{c}_i$ must be unmarked by row $\hat{r}_{i-1}$ and marked by row $\hat{r}_i$.

Lemma B.0.6 *$\hat{M}_\infty$ must not contain a Forbidden Submatrix, and if $\hat{M}_\infty$ has a row which has both the first and last columns marked, M'_∞ must also have a row which has both the first and last columns marked.*

Proof Assume $\hat{M}_\infty$ contains a Forbidden Submatrix. The two rows involved in the submatrix must be present in M'_∞ as well, and reordering them still results in a Forbidden Submatrix. Thus M'_∞ must contain a Forbidden Submatrix as well. However we know from Lemma B.0.5 that M'_∞ does not contain a Forbidden Submatrix, thus it must be that $\hat{M}_\infty$ does not contain a Forbidden Submatrix as well.

The set of rows in $\hat{M}_\infty$ is a subset of the rows in M'_∞, so by definition each row in $\hat{M}_\infty$ is identical to some row in M'_∞. If $\hat{M}_\infty$ has a row which has both the first and last columns marked, this row must be identical to some row in M'_∞. Thus M'_∞ must have a row which has both the first and last columns marked as well. ∎

Lemma B.0.7 $\forall_{1<i}$: *Every column from $G_e(\hat{r}_i)$ to L inclusive marked in $\hat{r}_i$ must be marked in $\hat{r}_{i-1}$*

Proof By definition, the submatrix formed by columns 1 to $G_e(\hat{r}_i) - 1$ inclusive and by rows $\hat{r}_{i-1}$ and $\hat{r}_i$ contains M_{f3} as a submatrix. Since $\hat{M}_\infty$ does not contain a Forbidden Submatrix by Lemma B.0.6, then there must not exist a column in $\hat{M}_\infty$ which can combine with the M_{f3} submatrix to form M_{f1}.

In a proof by contradiction, assume there exists some column between $G_e(\hat{r}_i)$ and L inclusive which is marked in $\hat{r}_i$ but unmarked in $\hat{r}_{i-1}$. Combining this column with the M_{f3} submatrix would form M_{f1}. Thus it must be that any columns from $G_e(\hat{r}_i)$ to L inclusive marked in $\hat{r}_i$ is marked in $\hat{r}_{i-1}$. ∎

Lemma B.0.8 $\forall_{1 \leq h \leq i}, \forall 1 < j < k$ *: If $\hat{r}_i$ is r'_j, and r'_k is unmarked in column $G_e(\hat{r}_i)$, then $\hat{c}_h$ must be unmarked in r'_k.*

Proof This can be shown via a proof by contradiction. If this lemma is false, there must be some $\hat{c}_h$ that is marked in r'_k. Let $\tilde{h}$ be the smallest integer such that $\hat{c}_{\tilde{h}}$ is marked in r'_k.

By definition, $G_e(r'_k)$ is greater than the G_e of all previous rows. Thus $\tilde{h}$ must be greater than 1, otherwise r'_k would mark the first column and should have been chosen as the first row of $\hat{M}_\infty$. So there must be a row $\hat{r}_{\tilde{h}-1}$ that is unmarked in $\hat{c}_{\tilde{h}}$ by definition. Furthermore, since $\tilde{h}$ was chosen as the smallest integer, it must be that $\hat{c}_{\tilde{h}-1}$ is marked in $\hat{r}_{\tilde{h}-1}$ and unmarked in r'_k.

Consider column $G_e(\hat{r}_i)$. Since it is marked in $\hat{r}_i$, it must also be marked in all previous rows of $\hat{M}_\infty$ by inductive use of Lemma B.0.7.

However, the submatrix formed by the columns $\hat{c}_{\tilde{h}-1}$, $\hat{c}_{\tilde{h}}$, and $G_e(\hat{r}_i)$ and rows $\hat{r}_{\tilde{h}-1}$ and r'_k is the M_{f3} matrix. Then by the construction procedure of $\hat{M}_\infty$, r'_k with its greater G_e value should have been added to $\hat{M}_\infty$ right after row $\hat{r}_{\tilde{h}-1}$ and the rows in between should not have been added. Thus we reach a contradiction, and it must be the case that no $\hat{c}_h$ that is marked in r'_k. ∎

Lemma B.0.9 *The last row of $\hat{M}_\infty$ must be the last row of M'_∞.*

Proof This can be shown via a proof by contradiction. Assume the last row of $\hat{M}_\infty$ is not the last row of M'_∞. In other words, $\hat{r}_{\hat{R}}$ is not $r'_{R'}$, but some other row r'_j in M'_∞. Then the next row in M'_∞ must have $G_s(r'_j) < G_s(r'_{j+1}) < G_e(r'_j) < G_e(r'_{j+1})$. By definition, this means column $G_s(r'_{j+1})$ is marked in r'_{j+1} but unmarked in r'_j, and column $G_e(r'_j)$ is unmarked in r'_{j+1} but marked in r'_j.

By Lemma B.0.8, since r'_{j+1} is unmarked in column $G_e(\hat{r}_{\hat{R}})$, it must also be unmarked in $\hat{c}_{\hat{R}}$.

However, the submatrix formed by the columns $\hat{c}_{\hat{R}}$, $G_s(r'_{j+1})$, and $G_e(\hat{r}_{\hat{R}})$ and the rows $\hat{r}_{\hat{R}}$ and r'_{j+1} must then be the matrix M_{f3}. Furthermore by definition $G_e(r'_j) < G_e(r'_{j+1})$. In other words, r'_j satisfies both constraints, meaning it should have been added to $\hat{M}_\infty$. Thus we reach a contradiction, and it must be the case that the last row of $\hat{M}_\infty$ is the last row of M'_∞. ∎

By definition, the last row of M'_∞ has $\mathbf{Last}(r'_{R'}) = G_e(r'_{R'}) = L$. In other words, it has the last column marked. By Lemma B.0.9, this means the last row of $\hat{M}_\infty$ also has the last column marked. Furthermore, by inductive use of Lemma B.0.7, the last column must be marked in all previous rows of $\hat{M}_\infty$.

By definition, the first row of $\hat{M}_\infty$ has $\mathbf{First} = 1$. In other words, it has the first column marked. Moreover, we just proved that it must also have the last column marked, so it is a row that has both the first and last columns marked. By Lemma B.0.6, Lemma B.0.5 and Lemma B.0.3, M_∞ must have a row which has both the first and last columns marked, thus proving Theorem B.0.2.

We can now present a proof of Theorem B.0.1:

Proof In a proof by contradiction, assume there exists some matrix $\tilde{M}$ that does not have Table B.1 or Table B.2 as submatrices, has more than two columns and no column other than the first or last column where each row of the matrix is in at least one of the following states:

- The row has no marked cells to the right of the column.
- The row has no marked cells to the left of the column.
- The row has a marked cell in the column.
- The row has marked cells in the first and last columns.

In other words, $\tilde{M}$ is some non-empty boolean matrix with more than two columns and the following constraints:

- $\tilde{M}$ must not contain a Forbidden Submatrix.
- Every column other than the first or last one must have at least one row that is not in the first four states. In other words, it has at least one row that has marked cells on both sides but an unmarked cell in the column and this row must also not be one where the first and last columns are both marked.

Let $\tilde{M}'$ be the submatrix of $\tilde{M}$ where all rows with both the first and last columns marked are removed. Being a submatrix, it must be the case that $\tilde{M}'$ also does not contain a Forbidden Submatrix. Now consider each column other than the first or last one. In $\tilde{M}$ it has at least one row that has marked cells on both sides but an unmarked cell in the column and this row must also not be one where the first and last columns are both marked. Since this row does not have both the first and last columns marked, it must also exist in $\tilde{M}'$.

In other words, $\tilde{M}'$ is some non-empty boolean matrix with more than two columns and the following constraints:

- $\tilde{M}'$ must not contain a Forbidden Submatrix.
- Every column other than the first or last one must have at least one row that is not in the first four states. In other words, it has at least one row that has marked cells on both sides but an unmarked cell in the column and this row must also not be one where the first and last columns are both marked.
- No row has both the first and last columns marked since any such rows were removed.

Note that since no row has both the first and last columns marked, it is impossible for a row to be in the fourth state. Thus the constraints can be restated as follows:

- $\tilde{M}'$ must not contain a Forbidden Submatrix.
- Every column other than the first or last one must have at least one row that is not in the first three states. In other words, it has at least one row that has marked cells on both sides but an unmarked cell in the column.
- No row has both the first and last columns marked since any such rows were removed.

We previously defined M_∞ to be an arbitrary non-empty boolean matrix with more than two columns and the following constraints:

- M_∞ must not contain a Forbidden Submatrix.
- Every column other than the first or last one must have at least one row that is not in the first three states. In other words, it has at least one row that has marked cells on both sides but an unmarked cell in the column.

Note that these are identical to the first two constraints of $\tilde{M}'$. By Theorem B.0.2, M_∞ must have some row that has both the first and last columns marked. In other words, any non-empty boolean matrix with more than two columns that is under the first two constraints must have some row that has both the first and last columns marked.

So under the first two constraints, the third constraint must be violated. This means $\tilde{M}'$ cannot exist, and in turn neither can $\tilde{M}$. As we reach a contradiction, it must be the case that any matrix that does not have Table B.1 or Table B.2 as submatrices must either have at most two columns or a Splittable Column. $\blacksquare$

Appendix C

Proofs Regarding Bracketings in the Intersection-Limited Problem Scenarios

This appendix covers the proofs of the observations made in Chapter 6 about timelines with planar bracketing in the Intersection-Limited Problem Scenarios.

C.1 General Observations

As before, for any wraparound bracket b_i, let us say it *interferes* with some other wraparound bracket b_j if there is an endpoint of b_i in between the endpoints of b_j or any other wraparound bracket that interferes with b_j. Note that by this definition, interference is a transitive and symmetric relation. We will also say b'_i *directly interferes* with b'_j only if it has an endpoint in between the endpoints of b'_j. This is a symmetric but not transitive relation. This means any pair of wraparound brackets b_i and b_j interfere if and only if there exists a sequence of wraparound brackets from b_i to b_j such that each wraparound bracket in the sequence directly interferes with its neighbours. The following observations can be made about the structure of these wraparound brackets in B_{rem}:

Observation C.1.1 *For any two wraparound brackets b_i and b_j, if the first endpoint of b_j is after the first endpoint of b_i, the last endpoint of b_j must be after the* *last endpoint of b_i.*

Proof This can be shown via a proof by contradiction. Let the first endpoint of b_j be after the first endpoint of b_i, but the last endpoint of b_j be before the last endpoint of b_i. Then the two brackets must clearly intersect. This result may be easier to see when the timeline is unfolded. Regardless of whether the brackets are clockwise or counterclockwise, their endpoints inevitably form a Forbidden Configuration such as the one shown in Figure 5.3, which means the two brackets must be crossing. This is a contradiction because B_{rem}, being a subset of the planar full bracketing B_{full}, must itself be a planar bracketing. ▮

Observation C.1.2 *For any two wraparound brackets b_i and b_j, if b_i is clockwise while b_j is counter-clockwise, b_i cannot interfere with b_j.*

Proof This can be shown via a proof by contradiction. If b_i is clockwise while b_j is counter-clockwise and b_i interferes with b_j, by definition there must exist at least one pair of directly interfering wraparound brackets b'_i and b'_j which are clockwise and counter-clockwise respectively. As before, let an endpoint be considered negative if it approaches its vertex from below, and positive otherwise.

Since b'_i is clockwise, its first endpoint must be negative while its second endpoint must be positive. In contrast, the first endpoint of b'_j must be positive while the second endpoint must be negative. Consider the transformation from the folded timeline to the unfolded timeline. If the negative second endpoint of b'_j is after the negative first endpoint of b'_i and the positive first endpoint of b'_j is before the positive second endpoint of b'_i, the endpoints would form a Forbidden Configuration such as the one shown in Figure 5.3, which means the two brackets must be crossing. By definition of direct interference, b'_j must have an endpoint between the endpoints of b'_i.

Assume this is the positive first endpoint of b'_j. This means the positive first endpoint of b'_j is before the positive second endpoint of b'_i. This also means the positive first endpoint of b'_j is after the negative first endpoint of b'_i, which means the negative second endpoint of b'_j must also be after the negative first endpoint of b'_i. As we have just shown, these conditions mean the two brackets must be crossing.

Assume instead that it is the negative second endpoint of b'_j that is between the endpoints of b'_i. This means the negative second endpoint of b'_j is after the negative first endpoint of b'_i. This also means that the negative second endpoint of b'_j is before the positive second endpoint of b'_i, which means the positive first endpoint of b'_j must also be before the positive second endpoint of b'_i. Then once again, the two brackets must be crossing.

As we are given a timeline with planar bracketing, such a crossing is a contradiction. Thus no clockwise wraparound bracket may interfere with a counter-clockwise wraparound bracket. ∎

C.2 Total Interference Block Observations

In this section, we consider the bracketings B_1 and B_2 independently, and make some observations about the brackets in each bracketing. For simplicity, the proofs are presented for the B_1 bracketing, but by the symmetry of the splitting procedure described in Section 6.1 the proofs can be generalized to apply for B_2 as well.

Observation C.2.1 *For any pair of wraparound brackets b_i and b_j, if b_i interferes with b_j, b_i must directly interfere with b_j.*

Proof This can be shown via a proof by contradiction. Assume b_i interferes with b_j without directly interfering with b_j. By definition, this means there is some sequence of wraparound brackets from b_i to b_j such that each wraparound bracket in the sequence directly interferes with its neighbours and all wraparound brackets in the sequence including b_i and b_j are in B_1. This means there exists some sequence of three wraparound brackets b'_k, b'_l and b'_m in B_1 such that b'_l directly interferes with b'_k and b'_m, but b'_k and b'_m do not directly interfere.

Now consider B_{rem}. In particular, consider the wraparound brackets after b'_k that directly interfere with b'_k in B_{rem}. If they are all in B_1, by our splitting procedure's usage of Total Interference Blocks, there must be no wraparound brackets in B_{rem} after b'_k that interfere with b'_k unless they directly interfere with b'_k. This contradicts our requirement that b'_m interferes with b'_k without directly interfering with b'_k. Thus there must exist at least one wraparound bracket after b'_k that directly interferes with b'_k in B_{rem} and is assigned to B_2.

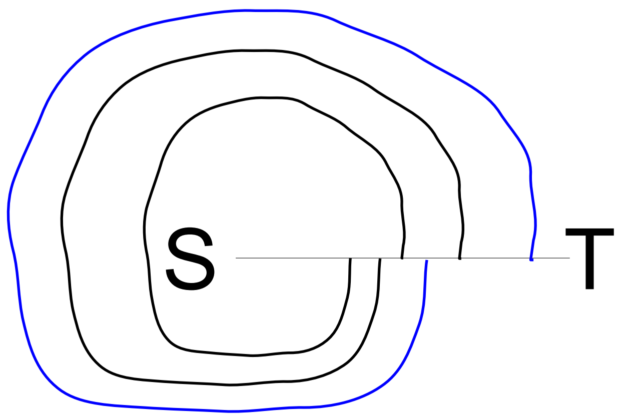

Figure C.1: An example showing the structure required by a wraparound bracket with a negative first endpoint in B_1.

By our splitting procedure, the first such wraparound bracket would have been denoted by b^* at some point, and all wraparound brackets directly interfering with b^* would be assigned to the same bracketing as b^*, which is B_2 in this case. Note that since b^* directly interferes with b'_k, any bracket whose first endpoint is between the first endpoints of b'_k and b^* must directly interfere with b'_k by definition.

Since b'_l directly interferes with b'_k and both b'_l and b'_k are in B_1, b'_l must be one of the wraparound brackets whose first endpoint is before the first endpoint of b^*. Then since b'_l's first endpoint is before the first endpoint of b^*, by Observation C.1.1 its second endpoint must be before the second endpoint of b^*. This means every bracket after b^* that directly interferes with b'_l must be in B_2.

Now consider b'_m. b'_m does not directly interfere with b'_k. It is also after b'_k. Thus it cannot have its first endpoint before the first endpoint of b^*. b'_m directly interferes with b'_l. In other words, it is a bracket after b^* that directly interferes with b'_l. Thus b'_m must be in B_2. This is a contradiction as b'_m is supposed to be a bracket in B_1. Thus any pair of wraparound brackets in B_1 which interfere with each other with respect to B_1 must directly interfere with each other. ∎

Observation C.2.2 *No non-wraparound bracket contains an endpoint from both a clockwise wraparound bracket and a counter-clockwise wraparound bracket.*

Proof Once again, we will prove this via contradiction. Assume there exists some non-wraparound bracket b^+ that contains an endpoint from both a clockwise wraparound bracket b_i and a counter-clockwise wraparound bracket b_j. Note that by our splitting procedure, no negative brackets are assigned to B_1, so b^+ must be a positive bracket.

Since brackets cannot intersect, b^+ cannot contain a positive endpoint of a wraparound bracket since it would intersect that bracket. This means the endpoints contained by b^+ must be negative endpoints. Furthermore, both endpoints of b_j must be before the first endpoint of b_i, otherwise two of the three brackets will inevitably intersect since b_i is clockwise while b_j is counter-clockwise.

Consider the Total Interference Block that contains b_i. Let the first wraparound bracket in this Total Interference Block be b_k. By our splitting procedure, b_k must be in B_2. Since b_i is in B_1, its first endpoint must be after the positive second endpoint of b_k. As before, b^+ cannot contain a positive endpoint of a wraparound bracket since it would intersect that bracket. Thus b^+ must begin after the positive second endpoint of b_k in order to contain the negative endpoint of b_i without containing the positive endpoint of b_k.

This is illustrated by Figure C.1. Here B_1 is represented by the colour blue while B_2 is represented by the colour black. In order to contain a negative endpoint from a blue clockwise wraparound bracket, a positive bracket must begin after the positive endpoint of the black clockwise wraparound bracket in the same Total Interference Block.

However, as previously stated, the negative second endpoint of b_j is before the first endpoint of b_i. In order for b^+ to contain this negative endpoint of b_j as well, this negative endpoint must then be between the first endpoint of b_i and the second endpoint of b_k. By definition, this places b_j inside the same Total Interference Block as b_i and b_k.

This contradicts Observation C.1.2, since the Total Interference Block now contains a clockwise wraparound bracket and a counter-clockwise wraparound bracket. Thus no non-wraparound bracket contains an endpoint from both a clockwise wraparound bracket and a counter-clockwise wraparound bracket. ▮

Observation C.2.3 *If a non-wraparound bracket contains a pair of bracket end-*

points corresponding to a pair of wraparound brackets, the pair of wraparound brackets must directly interfere with each other.

Proof By Observation C.2.2, we know that the pair of wraparound brackets must be either both clockwise, or both counter-clockwise.

Assume they are both clockwise. Let b_i denote the first wraparound bracket and b_j denote the second wraparound bracket. Since b_i does not directly interfere with b_j, by definition b_i must have its second endpoint before b_j has its first endpoint. This means the second endpoint of b_i is between the first endpoints of b_i and b_j.

Since they are both clockwise, their first endpoints must be negative, and the positive bracket must contain both their first endpoints. This also means the second endpoint of b_i is positive, and we know that this positive endpoint is between the first endpoints of b_i and b_j. This means the positive bracket contains the positive endpoint of b_i, which is a contradiction because that inevitably means b_i intersects the positive bracket.

By a similar argument, we will also derive a contradiction if they are both counter-clockwise. Thus the pair of wraparound brackets must directly interfere with each other. ∎

Appendix D

Issues Regarding Interior Sensor Intersections

This appendix discusses the issues regarding interior sensor intersections. In particular, we discuss the constraints required for a problem scenario to guarantee the existence of a $(\infty, \mathtt{i})$-traversal for finite values of $\mathtt{i}$. In Section D.1, we discuss the limits on the range of disk sensor radii required to avoid interior sensor intersections. In other words, the requirements for $\mathtt{i} = 0$. Note that this means the $(\infty, \mathtt{i})$-traversal is a resilience-optimal peripheral traversal. Then in Section D.2, we discuss methods of bounding the minimum number of interior intersections made per sensor by a D_c-resilience-optimal $S-T$ path.

D.1 Limits on the Range of Sensor Disk Sizes for Peripheral Traversals

Figure D.1 illustrates some of the components required in order for all resilience-optimal paths to have an interior sensor intersection. The path is indicated by the red lines, and the middle red line is a sensor intersection of the center yellow sensor that has dual orientation. Gray sensors are used to separate the three red sensor intersections in the yellow sensor, and green sensors (not fully shown) are used to force the path to intersect the yellow sensor.

As can be seen from the figure, if the interior sensor intersection cannot be

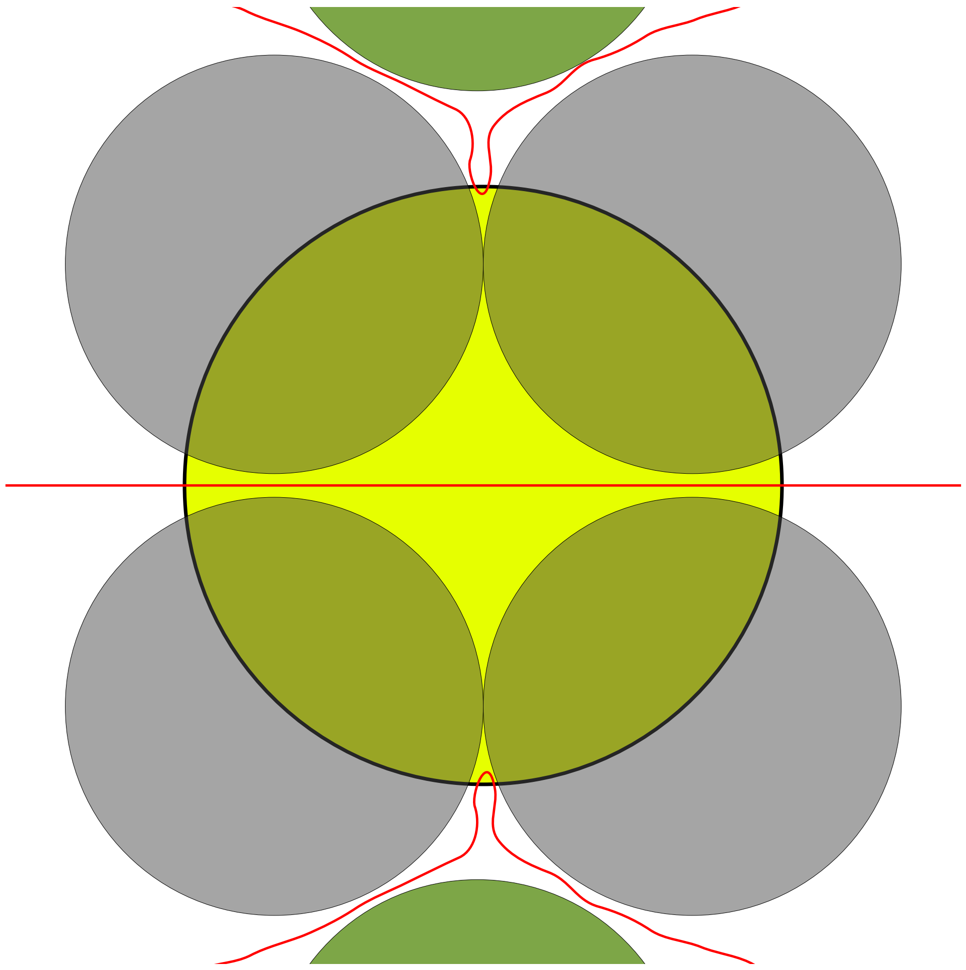

Figure D.1: A schematic illustrating the components necessary in an arrangement to force all resilience-optimal paths to have an interior sensor intersection.

avoided, there must exist some sensors forming an obstruction which prevents the red path from directly going through the yellow sensor in between two distinct red sensor intersections and forming an alternative optimal path that does not have this interior sensor intersection. These obstruction sensors are represented by the gray sensors.

Furthermore, there must be other sensors that force the path into the yellow sensor, otherwise there exists an alternative optimal path that does not make the sensor intersection, such as in this case where its clear that the red path could just skip the top and bottom sensor intersections. In other words, the green sensors have to overlap the yellow sensor in order to force the red path to intersect it, but at the same time the green sensors must not overlap too many of the gray sensors or the path would not even be able to reach the yellow sensor without crossing either the gray or green sensors.

In Figure D.1, since the disks are approximately equal-sized, there is no way to move the green sensors inwards to force the red sensor intersections. As such there exists alternative resilience-optimal paths without interior sensor intersections.

However, it should be clear that if the range of sensor disk sizes is sufficiently large, there exists arrangements that force every optimal path to have an interior sensor intersection, such as the arrangements presented in Figure D.2 and Figure D.3.

The arrangement in Figure D.2 is composed of disk sensors with radii in the range $[0.72, 1]$. The circle in the center of the figure represents a sensor of radius 1, and the smaller circles represent disk sensors of radii 0.72. The sensors in this arrangement have been positioned such that all resilience-optimal paths must make an interior sensor intersection of the center sensor, with one such path illustrated in red. Note that the regions S and T are infinitesimally small regions that are bounded by the center sensor and two smaller sensors, which is what forces the peripheral sensor intersections to the top and bottom of the center sensor.

The arrangement in Figure D.3 is similar, but composed of disk sensors with radii in the range $[0.6, 1]$. In this case, S and T are well-separated in the sense that the distance between S and T is at least $2\sqrt{3}$, but the sensors have once again been positioned to force every resilience-optimal path to make an interior sensor intersection of the center sensor.

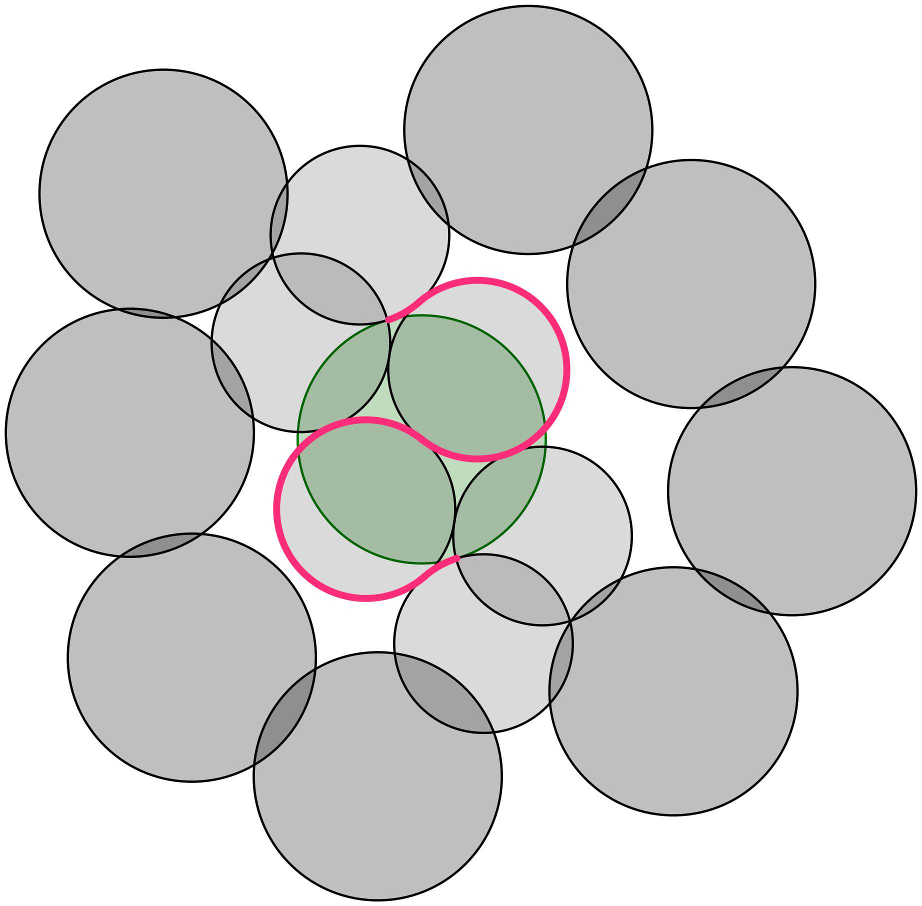

Figure D.2: An extremal example arrangement where the range of disk radii is just large enough to force all resilience-optimal paths to have an interior sensor intersection.

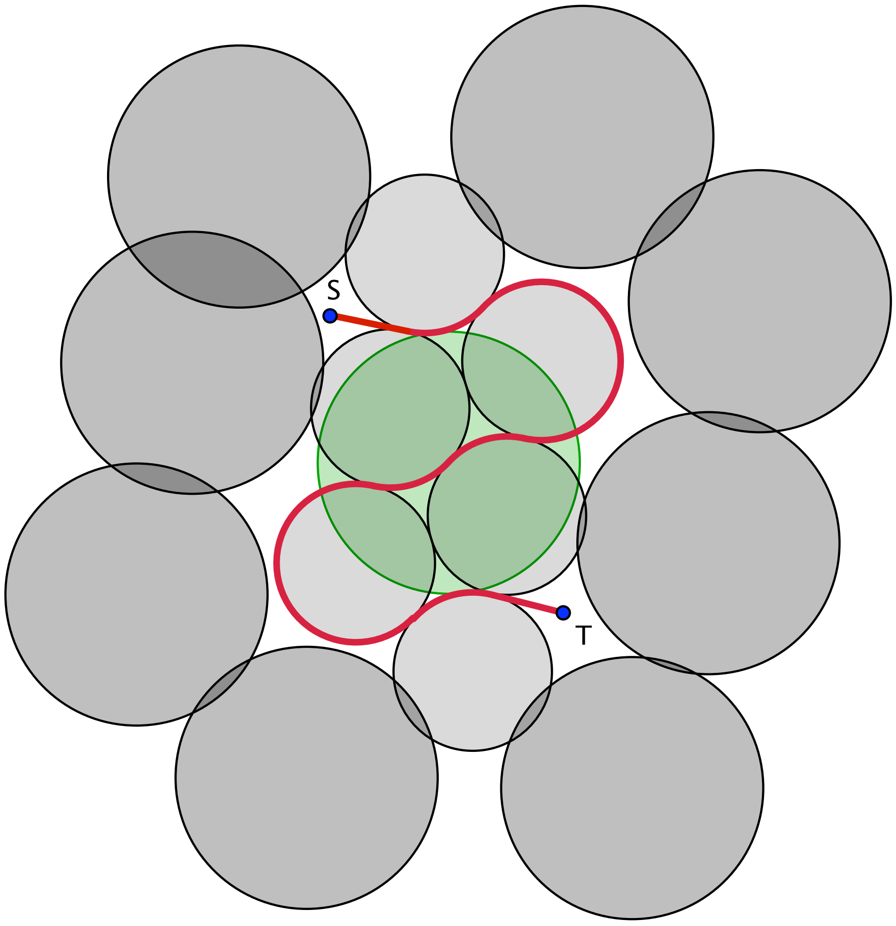

Figure D.3: An extremal example arrangement where the range of disk radii is just large enough to force all resilience-optimal paths to have an interior sensor intersection under the constraint that the distance between S and T is at least $2\sqrt{3}$.

We believe that the arrangements presented in Figure D.2 and Figure D.3 are actually extremal examples, in the sense that decreasing the range of the sensor radii by any amount would render it impossible to force all resilience-optimal paths to make an interior sensor intersection of the center sensor. In other words, we make the following conjecture:

Conjecture D.1.1 *For all arrangements of disk sensors with radii in the range* $[1-\xi, 1]$*, if* $\xi < 0.28$*, there exists a resilience-optimal peripheral traversal. Furthermore, if the distance between S and T is at least* $2\sqrt{3}$*, the upper bound on* ξ *can be increased to* 0.4 *while maintaining the existence of a resilience-optimal peripheral traversal.*

D.2 Bounds on the Minimum Number of Interior Sensor Intersections for Optimal Paths

Consider the sensor splitting arguments presented in Section 5.5, where a D_c-resilience-optimal acyclic $S-T$ path is used to split every sensor along every sensor intersection into subsensors which overlap exactly at the sensor intersections. As previously mentioned, every pair of sensor intersections to the same sensor requires an obstruction formed by other sensors in order to prevent the existence of alternative D_c-resilience-optimal acyclic $S-T$ paths which make fewer sensor intersections. This property holds true for the multiply-intersected subsensors as well, which means each multiply-intersected subsensor must take up some amount of area for its associated obstruction sensors.

Consequently, since each sensor has a limited amount of area, there must be some positive function of ξ that is independent of n which produces an upper bound on the number of multiply-intersected subsensors that can be derived from each sensor. Then observe that by construction, the number of multiply-intersected subsensors derived from a sensor must be exactly one more than the number of interior sensor intersections for that sensor. In other words, there exists a positive function $I(\xi)$ that is independent of n and upper bounds the minimum number of interior sensor intersections made per sensor by a D_c-resilience-optimal $S-T$ path. In particular, we present the following conjecture:

Conjecture D.2.1 *The minimum number of interior sensor intersections made per sensor by a resilience-optimal $S-T$ path is at most $\frac{\pi}{(\pi-2)(1-\xi)^2}-1$. In other words, $I(\xi) < \frac{\pi}{(\pi-2)(1-\xi)^2}-1$.*

This conjecture stems from considering the area of smallest possible multiply-intersected subsensor. Intuitively, this subsensor should be derived from a sensor with the minimum allowable radius, and have obstruction sensors which also have the minimum allowable radius. Furthermore, the number of sensor intersections should be exactly 2 to minimize the amount of area needed for obstructions.

Then we believe that the subsensor shown in Figure D.4 has the minimum area due to the analysis of unit disks presented in [2]. Here the 3 sensors represented by circles each have radius $1-\xi$, and the area with the thick black outline represents the subsensor derived from the yellow sensor which we believe has the minimum area. The two sensor intersections to this subsensor are shown as the red and blue lines, and the two gray sensors are tangent at a point just between the pair of sensor intersections, allowing them to serve as an obstruction for the pair of sensor intersections. By simple geometry, the area of this subsensor is at least $4(\frac{\pi(1-\xi)^2}{4}-\frac{(1-\xi)^2}{2}) = (\pi-2)(1-\xi)^2$.

Next, note that by definition, the largest sensor has radius 1, so it has area π. Then since we just showed that the area of each multiply-intersected subsensor is at least $(\pi-2)(1-\xi)^2$, the number of subsensors derived from each sensor must be at most $\frac{\pi}{(\pi-2)(1-\xi)^2}$. Thus the number of interior sensor intersections is at most $\frac{\pi}{(\pi-2)(1-\xi)^2}-1$.

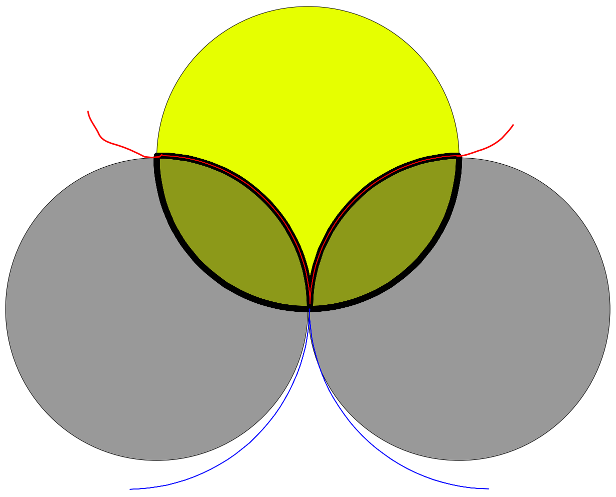

Figure D.4: A multiply-intersected subsensor of minimum area.

Appendix E

Proofs Regarding High Order Voronoi Diagrams

This appendix covers the proofs of the properties about High Order Voronoi diagrams with additive weights used in Section 7.5. As before, let **weightD**$(\check{p}, s_i)$ and **EuclidD**$(\check{p}, s_i)$ denote the weighted and Euclidean distances respectively from some point $\check{p}$ to some site s_i. Note that by definition the weighted distance from a site to itself is simply its weight, so **weightD**(s_i, s_i) also denotes the weight of a site s_i. We will also let **EuclidD**$(\check{p}_1, \check{p}_2)$ denote the Euclidean distance from some point $\check{p}_1$ to some other point $\check{p}_2$. Lastly, for any site s_i, we'll say that it appears in some Voronoi diagram if and only if there exists some cell in that Voronoi diagram that lists s_i as one of its $c-1$ closest sites.

E.1 Proofs Regarding General High Order Voronoi Diagrams with Additive Weights

Let $v_{\tilde{c}}$ be some order-$\tilde{c}$ Voronoi diagram with additive weights where $\tilde{c}$ is some positive integer, and let $s_{\tilde{i}}$ and $s_{\tilde{j}}$ be some sites that are used in constructing $v_{\tilde{c}}$.

Lemma E.1.1 *If there exists a cell in $v_{\tilde{c}}$ that has $s_{\tilde{i}}$ listed as one of its closest sites, site $s_{\tilde{i}}$ must be inside some cell that lists it as one of the closest sites.*

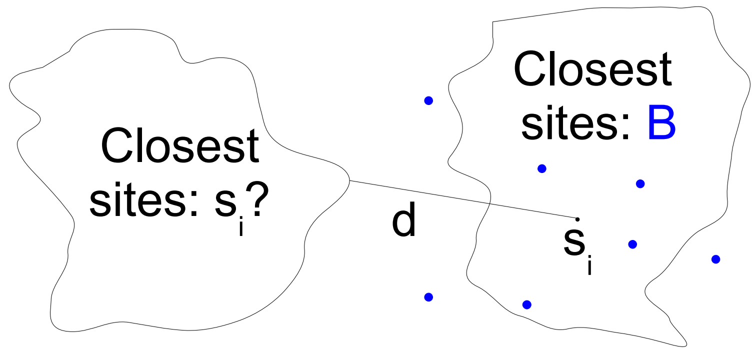

Figure E.1: An example Voronoi diagram illustrating why $s_{\tilde{i}}$ must be inside some cell that lists it as one of the closest sites if it has a cell.

Proof This can be shown via a proof by contradiction. If this is false, there must exist a cell that has $s_{\tilde{i}}$ listed as one of its closest sites, but the cell that $s_{\tilde{i}}$ is inside must not list $s_{\tilde{i}}$ as one of the closest sites.

Consider the cell that contains $s_{\tilde{i}}$. Since $v_{\tilde{c}}$ is a Voronoi diagram with additive weights, even if the cell contains $s_{\tilde{i}}$ it is possible to not have $s_{\tilde{i}}$ as the closest site. Let B be the set of closest sites for this cell. In other words, the weighted distance from $s_{\tilde{i}}$ to itself is $\mathbf{weightD}(s_{\tilde{i}}, s_{\tilde{i}})$, and there is a set B of sites such that $\forall_{s_b \in B}$: $\mathbf{weightD}(s_b, s_{\tilde{i}}) < \mathbf{weightD}(s_{\tilde{i}}, s_{\tilde{i}})$.

Consider any point $\check{p}$ in the cell that lists $s_{\tilde{i}}$ as one of its closest sites. Let $\mathbf{EuclidD}(\check{p}, s_{\tilde{i}}) = d$. This is illustrated in Figure E.1, where the blue dots represent the sites in B. Then by definition, $\mathbf{weightD}(\check{p}, s_{\tilde{i}}) = d + \mathbf{weightD}(s_{\tilde{i}}, s_{\tilde{i}})$.

However, $\forall_{s_b \in B} : \mathbf{weightD}(\check{p}, s_b) < d + \mathbf{weightD}(s_b, s_{\tilde{i}})$, since the weighted distance from s_b to $\check{p}$ must be less than the weighted distance of the path from s_b to $s_{\tilde{i}}$ to $\check{p}$. Thus $\forall_{s_b \in B}$: $\mathbf{weightD}(\check{p}, s_b) < \mathbf{weightD}(\check{p}, s_{\tilde{i}})$. Note that this applies for all points in the cell that lists $s_{\tilde{i}}$ since we let $\check{p}$ be any point in it. Thus the cell that lists $s_{\tilde{i}}$ must be closer to the sites in B than to $s_{\tilde{i}}$.

Since each cell can only list $|B|$ sites as its closest sites, the cell that lists $s_{\tilde{i}}$ actually cannot possibly list $s_{\tilde{i}}$ as one of its closest sites when all sites in B are closer than $s_{\tilde{i}}$. Thus it cannot exist and we have a contradiction. ∎

Lemma E.1.2 *If there exists a cell in $v_{\tilde{c}}$ that does not contain the point associated*

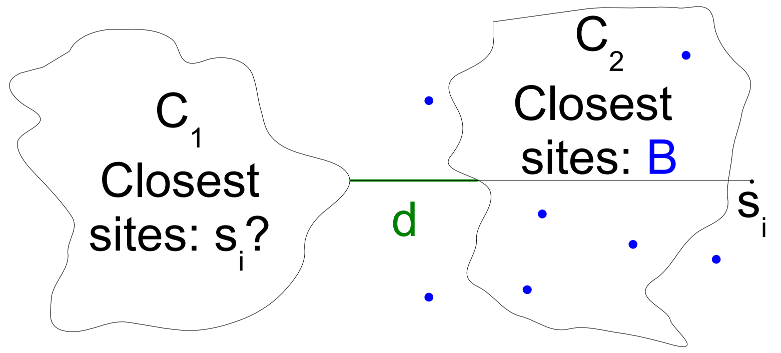

Figure E.2: An example Voronoi diagram illustrating why walking towards $s_{\tilde{i}}$ must never enter a cell without $s_{\tilde{i}}$ listed as one of its closest sites.

with $s_{\tilde{i}}$ but has $s_{\tilde{i}}$ listed as one of its closest sites, all cells intersected by a line segment from this cell to $s_{\tilde{i}}$ must list $s_{\tilde{i}}$ as one of their closest sites.

Proof This can be shown via a proof by contradiction. If this is false, there exists a cell C_1 that does not contain $s_{\tilde{i}}$ but has $s_{\tilde{i}}$ listed as one of its closest sites and a straight line segment L between some point $\check{p}_1 \in C_1$ and $s_{\tilde{i}}$ intersects some point $\check{p}_2$ in some cell C_2 that does not list $s_{\tilde{i}}$ as one of its closest sites.

Let B be the set of closest sites for cell C_2. Then because $\check{p}_2$ is inside C_2, it must be that $\forall_{s_b \in B} : \mathbf{weightD}(\check{p}_2, s_b) < \mathbf{weightD}(\check{p}_2, s_{\tilde{i}})$. Let $\mathbf{EuclidD}(\check{p}_1, \check{p}_2) = d$. This is illustrated by Figure E.2. Then by definition, $\mathbf{weightD}(\check{p}_1, s_{\tilde{i}}) = d + \mathbf{weightD}(\check{p}_2, s_{\tilde{i}})$.

Also, $\forall_{s_b \in B} : \mathbf{weightD}(\check{p}_1, s_b) \leq d + \mathbf{weightD}(\check{p}_2, s_b)$, since the weighted distance from s_b to $\check{p}_1$ must be less than the weighted distance of the path from s_b to $\check{p}_2$ to $\check{p}_1$. However since $\forall_{s_b \in B} : \mathbf{weightD}(\check{p}_2, s_b) < \mathbf{weightD}(\check{p}_2, s_{\tilde{i}})$ and $\mathbf{weightD}(\check{p}_1, s_{\tilde{i}}) = d + \mathbf{weightD}(\check{p}_2, s_{\tilde{i}})$, we have that $\forall_{s_b \in B} : \mathbf{weightD}(\check{p}_1, s_b) < \mathbf{weightD}(\check{p}_1, s_{\tilde{i}})$.

In other words, each site $s_b \in B$ must be closer to C_1 than $s_{\tilde{i}}$ in terms of the weighted distance. Since C_1 can only list $|B|$ sites as its closest sites, it cannot possibly list $s_{\tilde{i}}$ as one of its closest sites when all sites in B are closer than $s_{\tilde{i}}$. Thus it cannot exist and we have a contradiction. ∎

Lemma E.1.3 *All cells in $\mathcal{V}_{\tilde{c}}$ that have $s_{\tilde{i}}$ listed as one of their closest sites must be connected with respect to $s_{\tilde{i}}$, in the sense that there exists a path between any pair of cells that never enters a cell that does not have $s_{\tilde{i}}$ listed as one of the closest sites.*

Proof This can be shown via a proof by contradiction. If this is false, there must exist two cells C_1 and C_2 which have $s_{\tilde{i}}$ listed as one of their closest sites that are not connected, in the sense that there exists no path between them that never enters a cell that does not have $s_{\tilde{i}}$ listed as one of the closest sites.

By Lemma E.1.2, there is a path from C_1 to $s_{\tilde{i}}$ that never enters a cell that does not have $s_{\tilde{i}}$ listed as one of the closest sites. Similarly, by Lemma E.1.2, there is a path from C_2 to $s_{\tilde{i}}$ that never enters a cell that does not have $s_{\tilde{i}}$ listed as one of the closest sites. Then the combination of these two paths must be a path that connects C_1 and C_2, and this path never enters a cell that does not have $s_{\tilde{i}}$ listed as one of the closest sites. Thus we reach a contradiction. $\blacksquare$

Lemma E.1.4 *If $s_{\tilde{i}}$ and $s_{\tilde{j}}$ both appear in $\mathcal{V}_{\tilde{c}}$, there can only exist a point $\tilde{p}$ in $\mathcal{V}_{\tilde{c}}$ which uses $s_{\tilde{i}}$ and $s_{\tilde{j}}$ as part of its $\tilde{c}+1$ closest sites if there exists at least one edge in $\mathcal{V}_{\tilde{c}}$ which uses $s_{\tilde{i}}$ and $s_{\tilde{j}}$ as part of its set of $\tilde{c}+1$ closest sites.*

Proof Note that this proof may require that the points are non-degenerate, but this can be handled by the Simulation of Simplicity Technique by Edelsbrunner and Mucke [6].

This can be shown via a proof by contradiction. If this lemma is false, there must exist some point $\tilde{p}$ in $\mathcal{V}_{\tilde{c}}$ which uses some pair of sites $s_{\tilde{i}}$ and $s_{\tilde{j}}$ as part of its $\tilde{c}+1$ closest sites, but no edge in $\mathcal{V}_{\tilde{c}}$ which uses $s_{\tilde{i}}$ and $s_{\tilde{j}}$ as part of its set of $\tilde{c}+1$ closest sites. Note that this means no edge in $\mathcal{V}_{\tilde{c}}$ can have the same set of closest sites as $\tilde{p}$, so $\tilde{p}$ cannot be on an edge in $\mathcal{V}_{\tilde{c}}$.

We know that $\mathcal{V}_{\tilde{c}}$ is an order-$\tilde{c}$ Voronoi diagram. This means the cells in $\mathcal{V}_{\tilde{c}}$ have $\tilde{c}$ closest sites. In the absence of degeneracy, cells which share an edge must differ by exactly one site in their set of closest sites. In other words, the two cells must share $\tilde{c}-1$ closest sites but each have one more closest site that is not in the set of closest sites for the other. This means that the edges, which by definition have exactly one cell on each side, must have $\tilde{c}-1+2=\tilde{c}+1$ closest sites.

Let $v_{\tilde{c}+1}$ denote an order-$\tilde{c}+1$ Voronoi diagram constructed on the same set of sites and weights. The cells in $v_{\tilde{c}+1}$ have $\tilde{c}+1$ closest sites by definition. Since $\tilde{p}$ exists, there must exist some cell in $v_{\tilde{c}+1}$ which contains $\tilde{p}$ and thus has the same $\tilde{c}+1$ closest sites as $\tilde{p}$. Let $\mathbf{Cell}(v_{\tilde{c}+1})$ denote this cell. By definition, all points inside $\mathbf{Cell}(v_{\tilde{c}+1})$ must have the same $\tilde{c}+1$ closest sites. Then because no edge in $v_{\tilde{c}}$ must use $s_{\tilde{i}}$ and $s_{\tilde{j}}$ as part of its set of closest sites, there cannot be any edge in $v_{\tilde{c}}$ that overlaps $\mathbf{Cell}(v_{\tilde{c}+1})$.

In other words, if there is a edge in $v_{\tilde{c}}$ that overlaps $\mathbf{Cell}(v_{\tilde{c}+1})$, there exists a point in $\mathbf{Cell}(v_{\tilde{c}+1})$ that is also on this edge and thus the edge must have the same set of closest sites as p^*. Thus $\mathbf{Cell}(v_{\tilde{c}+1})$ must be completely and strictly inside some cell in $v_{\tilde{c}}$. Let $\mathbf{Cell}(v_{\tilde{c}})$ denote this cell. $\mathbf{Cell}(v_{\tilde{c}})$ cannot have both $s_{\tilde{i}}$ and $s_{\tilde{j}}$ as part of its closest $\tilde{c}$ sites, otherwise the edges of $\mathbf{Cell}(v_{\tilde{c}})$ would have them as part of its closest sites which is a contradiction.

Assume $\mathbf{Cell}(v_{\tilde{c}})$ does not have $s_{\tilde{i}}$ as one of its $\tilde{c}$ closest sites.

Now consider the edges of $\mathbf{Cell}(v_{\tilde{c}+1})$. Since $\mathbf{Cell}(v_{\tilde{c}+1})$ is inside $\mathbf{Cell}(v_{\tilde{c}})$, all these edges must be inside $\mathbf{Cell}(v_{\tilde{c}})$ as well. Furthermore, the cells on the other side of these edges must have portions inside $\mathbf{Cell}(v_{\tilde{c}})$ as well. These cells must then also have the $\tilde{c}$ sites of $\mathbf{Cell}(v_{\tilde{c}})$ as part of their set of $\tilde{c}+1$ closest sites, which means that all the cells adjacent to $\mathbf{Cell}(v_{\tilde{c}+1})$ share the same $\tilde{c}$ sites of $\mathbf{Cell}(v_{\tilde{c}})$ but all of them do not have $s_{\tilde{i}}$ as one of their closest sites. In other words, leaving $\mathbf{Cell}(v_{\tilde{c}+1})$ in any direction causes some other site to replace $s_{\tilde{i}}$.

By Lemma 7.5.4, going from $\mathbf{Cell}(v_{\tilde{c}+1})$ towards $s_{\tilde{i}}$ should never enter a cell that does not have $s_{\tilde{i}}$ listed as one of their closest sites. Since all neighbouring cells do not list $s_{\tilde{i}}$, the only possibility is that the position of $s_{\tilde{i}}$ is inside $\mathbf{Cell}(v_{\tilde{c}+1})$.

Then we have that $s_{\tilde{i}}$ is inside $\mathbf{Cell}(v_{\tilde{c}+1})$, and $\mathbf{Cell}(v_{\tilde{c}+1})$ is inside $\mathbf{Cell}(v_{\tilde{c}})$, so $s_{\tilde{i}}$ is inside $\mathbf{Cell}(v_{\tilde{c}})$ even though $\mathbf{Cell}(v_{\tilde{c}})$ does not have $s_{\tilde{i}}$ as one of its $\tilde{c}$ closest sites. Then by Lemma 7.5.3, it must be that no cell in $v_{\tilde{c}}$ has $s_{\tilde{i}}$ as its closest site, which is a contradiction because one of the preconditions was that $s_{\tilde{i}}$ appears in $v_{\tilde{c}}$. A similar argument for $s_{\tilde{j}}$ also derives a contradiction. ∎

E.2 Proofs Regarding Relationships between the Arrangement and the Voronoi Diagram

In this section, we provide the proofs of the relationships between the arrangement of sensors and the Voronoi diagram constructed in Section 7.5.

Lemma E.2.1 *Any cell in $\mathcal{V}_{c-1}(\mathtt{A})$ that contains points at distances greater than $d_{boundary}$ from $\mathtt{A}$ must not have any sensor in $\mathtt{A}$ listed as one of its $c-1$ closest sites.*

Proof By Proposition 7.5.8, any points at distances greater than $d_{boundary}$ from $\mathtt{A}$ must have the boundary sensors as their closest sites. Thus cells containing these points cannot have any sensor in $\mathtt{A}$ listed as one of their closest sites. ∎

Lemma E.2.2 *Any cell in $\mathcal{V}_{c-1}(\mathtt{A})$ that has a sensor in $\mathtt{A}$ listed as one of its $c-1$ closest sites must be bounded.*

Proof If a cell in $\mathcal{V}_{c-1}(\mathtt{A})$ is not bounded, it must contain points at distances greater than $d_{boundary}$ from $\mathtt{A}$. Then by Lemma E.2.1, this cell cannot have any sensor in $\mathtt{A}$ listed as one of its $c-1$ closest sites. ∎

Lemma E.2.3 *A point $\check{p}$ lies inside a disc sensor s_i if and only if in $\mathcal{V}_{c-1}(\mathtt{A})$, $\textbf{weightD}(\check{p}, s_i) \leq 1$.*

Proof For any sensor s_i, r_i denotes the radius of s_i. Then by definition, a point $\check{p}$ lies inside sensor s_i if and only if its Euclidean distance to the center of sensor s_i is less than or equal to r_i. In other words, $\textbf{EuclidD}(\check{p}, s_i) \leq r_i$.

As previously stated, when $\mathcal{V}_{c-1}(\mathtt{A})$ is constructed, a site is derived from sensor s_i using the center of sensor s_i as its position and it is assigned a weight of $1-r_i$. By definition, the weighted distance is the Euclidean distance plus the weight of the site $1-r_i$, or in other words $\textbf{weightD}(\check{p}, s_i) = \textbf{EuclidD}(\check{p}, s_i) + 1 - r_i$.

Substituting this into our previous equation we get $\textbf{weightD}(\check{p}, s_i) \leq r_i + 1 - r_i = 1$. ∎

Lemma E.2.4 *In $\mathtt{A}$, if any sensor s_i is fully inside some other sensor s_j, then for any point $\check{p}$, $\textbf{weightD}(\check{p}, s_j) < \textbf{weightD}(\check{p}, s_i)$.*

Proof Let L be the straight line segment that connects the centers of s_i and s_j. There must be a point $\check{p}_i$ on the boundary of s_i that is on L such that the center of s_i is between the center of s_j and $\check{p}_i$.

Since s_j fully covers s_i, $\check{p}_i$ must be inside s_j as well. Then by Lemma E.2.3, $\mathbf{weightD}(\check{p}_i, s_j) < \mathbf{weightD}(\check{p}_i, s_i)$. However, because of their layout on L, it then be the case that every point on L is closer to s_j than s_i, including the center of s_i itself. In other words, $\mathbf{weightD}(s_i, s_j) < \mathbf{weightD}(s_i, s_i)$.

Then for any point $\check{p}$ in the arrangement, $\mathbf{weightD}(\check{p}, s_j) < \mathbf{weightD}(s_i, s_j) + \mathbf{EuclidD}(\check{p}, s_i)$, since its weighted distance from the center of j cannot be greater than the weighted distance from the center of j to the center of i and then to the point. Then since $\mathbf{weightD}(s_i, s_j) + \mathbf{EuclidD}(\check{p}, s_i) < \mathbf{weightD}(s_i, s_i) + \mathbf{EuclidD}(\check{p}, s_i) = \mathbf{weightD}(\check{p}, s_i)$, $\mathbf{weightD}(\check{p}, s_j) < \mathbf{weightD}(\check{p}, s_i)$. Thus there exists no points that are closer to s_i than s_j. ∎

Lemma E.2.5 *For any two sensors s_i and s_j in $\mathtt{A}$, if the region associated with s_i is not fully inside the region associated with s_j, **weightD**$(s_i, s_i) <$ **weightD**(s_i, s_j).*

Proof If s_i is not fully inside s_j, by Lemma E.2.3 there must exist a point $\check{p}$ in s_i but not in s_j that has $\mathbf{weightD}(\check{p}, s_i) \leq 1 < \mathbf{weightD}(\check{p}, s_j)$.

By definition, $\mathbf{weightD}(s_i, s_i) = \mathbf{weightD}(\check{p}, s_i) - \mathbf{EuclidD}(\check{p}, s_i)$.

Furthermore, even in the worst-case scenario when s_j is in the same direction as s_i from $\check{p}$, $\mathbf{weightD}(s_i, s_j) \geq \mathbf{weightD}(\check{p}, s_j) - \mathbf{EuclidD}(\check{p}, s_i) > \mathbf{weightD}(\check{p}, s_i) - \mathbf{EuclidD}(\check{p}, s_i) = \mathbf{weightD}(s_i, s_i)$. ∎

Lemma E.2.6 *In $\mathcal{V}_{c-1}(\mathtt{A})$, if a sensor s_i is not listed as one of the closest sites for any cell, it must be fully inside $c-1$ other sensors.*

Proof This can be shown via a proof by contradiction. Assume s_i is not fully inside $c-1$ other sensors even though it is not listed as one of the closest sites for any cell in $\mathcal{V}_{c-1}(\mathtt{A})$.

Consider the center of s_i. By Lemma E.2.5, if it is not fully inside some sensor s_j, $\mathbf{weightD}(s_i, s_i) < \mathbf{weightD}(s_i, s_j)$. Thus unless it is fully inside $c-1$ other sensors, the center of s_i will have s_i as one of its $c-1$ closest sites. In that case, the cell in $\mathcal{V}_{c-1}(\mathtt{A})$ that contains the center of s_i must list s_i as one of the closest sites. Thus we reach a contradiction. ∎

Lemma E.2.7 *Two vertices s_i and s_j are connected by an edge in $G_c(\mathtt{A})$ if and only if there exists a point $\check{p}$ and some set ψ of at most c sensors including s_i and s_j such that for all s_l in ψ, **weightD**$(\check{p}, s_l) \leq 1$ and the weighted distance is greater than 1 for all other sensors.*

Proof By definition, two vertices s_i and s_j are connected by an edge in $G_c(\mathtt{A})$ if and only if there exists a point $\check{p}'$ that is covered by some set ψ' of at most c distinct sensors that includes s_i and s_j.

By Lemma E.2.3, for all s'_l in ψ', **weightD**$(\check{p}', s'_l) \leq 1$ and the weighted distance is greater than 1 for all other sensors, fitting the criteria above.

In a similar argument, assume there exists a point $\check{p}$ and some set ψ of at most c sensors including s_i and s_j such that for all s_l in ψ, **weightD**$(\check{p}, s_l) \leq 1$ and the weighted distance is greater than 1 for all other sensors.

In the construction of $\nu_{c-1}(\mathtt{A})$, the weight of any sensor was defined as one minus its radius. By definition, the weighted distance is the weight of the sensor plus the Euclidean distance. Restating the above formula in Euclidean distances, we derive that $\check{p}$ must have Euclidean distance to the center of a sensor less than or equal to the radius of the sensor if and only if it is one of the sensors in ψ.

Then, $\check{p}$ must be a point covered by a set ψ of at most c sensors, which includes s_i and s_j. This means that s_i and s_j must be connected by an edge in $G_c(\mathtt{A})$. ∎

Lemma E.2.8 *Two vertices s_i and s_j can be connected by an edge in $G_c(\mathtt{A})$ only if there exists a point $\check{p}'$ which has s_i and s_j as two of its c closest sites in terms of weighted distance.*

Proof By Lemma E.2.7, we have that s_i and s_j are connected by an edge in $G_c(\mathtt{A})$ only if there exists a point $\check{p}$ and some set ψ of at most c sensors including s_i and s_j such that for all s_l in ψ, **weightD**$(\check{p}, s_l) \leq 1$ and the weighted distance is greater than 1 for all other sensors. Then by definition, this point $\check{p}$ must have these at most c sensors as its closest sites in terms of weighted distance.

Let ψ' be a set of c sensors containing ψ. If $\psi < c$, use the other sensors closest to $\check{p}$ as members in ψ'. Then by this definition $\check{p}$ must have ψ' as the set of c closest sites. This means, $\check{p}$ can only exist if there exists some point $\check{p}'$ which uses s_i and s_j as part of its c closest sites, since $\check{p}$ is a candidate point for $\check{p}'$.

Thus we have that two vertices are connected by an edge in $G_c(\mathtt{A})$ only if there exists a point $\check{p}'$ which uses s_i and s_j as part of its c closest sites in terms of weighted distance. ∎

Lemma E.2.9 *Two vertices s_i and s_j that appear in $\mathcal{V}_{c-1}(\mathtt{A})$ can be connected by an edge in $G_c(\mathtt{A})$ only if there exists at least one edge in $\mathcal{V}_{c-1}(\mathtt{A})$ which uses their two associated sensors as part of the set of closest sites.*

Proof By Lemma E.2.8, we have that two vertices s_i and s_j that appear in $\mathcal{V}_{c-1}(\mathtt{A})$ can be connected by an edge in $G_c(\mathtt{A})$ only if there exists a point $\check{p}$ in $\mathcal{V}_{c-1}(\mathtt{A})$ which uses s_i and s_j as part of its c closest sites. By Lemma 7.5.6, $\check{p}$ exists only if there exists at least one edge in $\mathcal{V}_{c-1}(\mathtt{A})$ which uses s_i and s_j as part of the set of closest sites. ∎

Appendix F

A Further Extension to Multi-Path Algorithms: The Decomposition Function

This section discusses a further extension to Multi-Path Algorithms that is not used in the other sections of this thesis. This extension stems from the observation that X-Path constructs can be broken up into subconstructs, and the W^* measure of the X-Path construct can then be used to bound the W^* measure for the subconstructs. Thus rather than the merging rules we have been exploiting using the $\mathtt{M}$-functions, this extension presents $\mathtt{D}$-functions which exploit decomposition rules.

Let $\tilde{\mathtt{V}}_1$ and $\tilde{\mathtt{V}}_2$ denote two $2X$-tuples of vertices in V, such that every $\tilde{\mathtt{V}}_1$ construct contains a $\tilde{\mathtt{V}}_2$ construct as a subconstruct, in the sense that the paths of a $\tilde{\mathtt{V}}_2$ construct are subpaths of the $\tilde{\mathtt{V}}_1$ construct and respect the sequential ordering of the tuples. Note that this is only possible when vertices are shared by the tuples in certain patterns, and we denote this relation by saying $\tilde{\mathtt{V}}_1 \ominus * = \tilde{\mathtt{V}}_2$. In other words, every $\tilde{\mathtt{V}}_1$ construct can be decomposed into a $\tilde{\mathtt{V}}_2$ construct plus some uninteresting set of edges. Then the $\mathtt{D}$-function is defined as follows:

$$\mathtt{D}(d_1, \tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2) = \begin{cases} d_1 + \sum_{\mathtt{c} \in \mathtt{S}_{\hat{\mathtt{A}}}} (\mathtt{H}(\mathtt{c}, \tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2)) & \text{if } \tilde{\mathtt{V}}_1 \ominus * = \tilde{\mathtt{V}}_2 \\ \infty & \text{otherwise} \end{cases}$$

Observation F.0.10 *Given a $\tilde{V}_1$ construct $\hat{P}_1$ with at most d_1 hidden colours, if $\hat{P}_1$ can be decomposed into a $\tilde{V}_2$ construct $\hat{P}_2$ plus some uninteresting set of edges, $\hat{P}_2$ has at most $d_1 + \sum_{c \in S_{\hat{A}}}(H(c, \tilde{V}_1, \tilde{V}_1, \tilde{V}_2))$ hidden colours.*

Proof Since $\hat{P}_1$ can be decomposed into $\hat{P}_2$ plus some uninteresting set of edges, the colours used by $\hat{P}_2$ must be a subset of the colours used by $\hat{P}_1$. Thus $\hat{P}_2$ can only use a colour $\mathtt{c}$ if it is one of the d_1 hidden colours of $\hat{P}_1$, or $\mathtt{E}(\mathtt{c}, \tilde{\mathtt{V}}_1) = 1$. However, if $\mathtt{E}(\mathtt{c}, \tilde{\mathtt{V}}_2) = 1$, $\mathtt{c}$ would be an exposed colour of $\hat{P}_2$ and then by definition it cannot be a hidden colour of $\hat{P}_2$. As a result, by the definition of the $\mathtt{H}$ function in Section 3.3, $\hat{P}_2$ must have at most $d_1 + \sum_{\mathtt{c} \in \mathtt{S}_{\hat{\mathtt{A}}}}(\mathtt{H}(\mathtt{c}, \tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_1, \tilde{\mathtt{V}}_2))$ hidden colours. ∎

This $\mathtt{D}$-function is treated the same way as the merge function in every phase $y > 0$: the algorithm calls it multiple times to generate values guaranteed to be at least $W^*(\tilde{\mathtt{V}})$ for some $\tilde{\mathtt{V}}$, and assigns each array entry $d_y[\tilde{\mathtt{V}}]$ only the minimum value guaranteed to be at least $W^*(\tilde{\mathtt{V}})$.

This extension can actually be extremely useful in some situations because of the following theorem:

Theorem F.0.11 *Let B denote the set of colours used by a Minimum-Colour $s' - t'$ path P. If there exists a vertex v^* in V that is assigned every colour in B and no other colours, even a Two-Path Algorithm will output the exact value of $W'(s', t')$, regardless of how many edges in P are assigned colours in B.*

Proof Construct an 2-Path valid derivation tree as follows:

1. Create two leaf nodes corresponding to a 0-length path at v^* and a 0-length path at s' respectively.

2. Create a parent node for these two leaf nodes. This parent node is assigned a vertex tuple such that it corresponds to a Two-Path construct that has a 0-length path at v^* as its first path and a 0-length path at the start of P as its second path.

3. From this node, create a sequence of parent nodes that iteratively merges this Two-Path construct with the edges on P from s' to t'.

The root node of this valid derivation tree is thus associated with a Two-Path construct that has a 0-length path at v^* as its first path and P as its second path. As all colours used by P are assigned to v^*, all used colours must be exposed colours, so there are no hidden colours with respect to this construct. In other words, the W^* measure for the corresponding vertex tuple is equal to 0. Furthermore, because of the construction procedure above, the weight of the root node will actually be 0 as well since each merge step will never find a colour `c` for which `H` equals 1.

Then by applying the `D`-function, the algorithm will immediately get $|B|$ as an upper bound on $W'(s',t')$, and the theorem immediately follows. ▮

In other words, Theorem F.0.11 means we can exactly determine the Minimum-Colour $s'-t'$ path in cases where there exists some vertex which is assigned the exact set of colours used by a Minimum-Colour $s'-t'$ path.

Furthermore, we note that even if this desirable vertex doesn't exist in `A`, users can add extra vertices to "'seed"' the algorithm with close guesses to improve the approximation if they have some intuitive idea about the set of colours used by a Minimum-Colour $s'-t'$ path.

For instance, if a user believes that a Minimum-Colour $s'-t'$ path uses a set of colours B, an extra vertex v^* can be added to the graph v^* is assigned every colour in B and there are no edges connected to v^*. In effect, v^* having no edges means the algorithm is forced to use decomposition rules on vertex tuples associated with v^*, and the algorithm is prevented from using a path from s' to t' that somehow includes v^*, which is ideal since v^* does not represent a real vertex in the original graph. By Theorem F.0.11, v^* would then allow the algorithm to exactly determine the Minimum-Colour $s'-t'$ path if the user's beliefs were correct.

Alternatively, even if no Minimum-Colour $s'-t'$ path really uses only colours in B, if almost all colours used are in B it stands to reason that the algorithm should derive a close approximation of the Minimum-Colour $s'-t'$ path. However, note that this does not mean one can just seed the algorithm with every guess, since there are clearly an exponential number of subsets of the colours in the graph.

Unfortunately, while this shows that the above modification can make the algorithm perform extremely well in some situations, in the worst-case scenarios there would never exist such a useful vertex v^* and the users would not have any close guesses as to what set of colours will be used by a Minimum-Colour $s' - t'$ path. Thus this decomposition rule does not seem useful for proving any upper bounds on the approximation ratios of Multi-Path Algorithms.