School of Computing
FACULTY OF ENGINEERING

UNIVERSITY OF LEEDS

Something W this way comes

Petar Hristov

Submitted in accordance with the requirements for the degree of Mathematics and Computer Science

<Session>

The candidate confirms that the following have been submitted.
<As an example>

Items	Format	Recipient(s) and Date
Beliverable 1, 2, 3	Report	SSO (DD/MM/YY)
Participant consent forms	Signed forms in envelop	SSO (DD/MM/YY)
Deliverable 4	Software codes or URL	Supervisor, Assessor (DD/MM/YY)
Deliverable 5	User manuals	Client, Supervisor (DD/MM/YY)

Type of project: _____

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Student) _____

Summary

I have come here to chew bubble gum and kick ass. And I'm all out of bubble gum.

Acknowledgements

<The page should contain any acknowledgements to those who have assisted with your work. Where you have worked as part of a team, you should, where appropriate, reference to any contribution made by other to the project.>

Note that it is not acceptable to solicit assistance on 'proof reading' which is defined as the "the systematic checking and identification of errors in spelling, punctuation, grammar and sentence construction, formatting and layout in the test";
see http://www.leeds.ac.uk/gat/documents/policy/Proof-reading-policy.pdf.

Contents

Chapter 1

Introduction

1.1 Set Theoretic Notation

Should I do this as an appendix?

1.2 Topology

Topology is the mathematical field that studies continuous change between topological spaces. Any set X can be a topological space as long as we defined a collection of subsets of X called open sets. The open sets represent elements of X which are "near" or "close" to one another. If we have two topological spaces X an Y and wish to study how one can be continuously mapped to the other we instead focus on how the open sets are mapped. The open sets provide certain structure on the sets and we would like to study the functions that preserve that structure and focus on the properties of spaces that are invariant under such functions. Those functions are what we call the continuous functions.

Let us now be more formal now and define these notions precisely.
Definition 1. *Let X be a set and τ be a set of subsets of X. The set τ is a topology on X when the following holds:*

- X and $\emptyset \in \tau$.
- If U and $V \in \tau$ then $U \cap V \in \tau$.
- If $\{U_\lambda\}_{\lambda \in \Lambda}$ is a family of subsets of X, where $U_\lambda \in \tau$ for all $\lambda \in \Lambda$, then $\bigcup_{\lambda \in \Lambda} U_\lambda \in \tau$.

Definition 2. *Let (X, τ) be a topological space. Any subset of $A \subseteq X$ which is in τ is said ot be open.*
Definition 3. *Let (X, τ) be a topological space. Let $x \in X$ be any element of X. We will call x a point in X and we will call any open set A containing x an open neighbourhood of x.*

The pair (X, τ) is called a topological space. In practice one build a topology by first figuring out how they would like their open sets to look like and then takes all unions and finite intersections to obtain the full topology.

Example 1. *The standard topologly on* $\mathbb{R}$.

The standard topology on $\mathbb{R}$ is build from subsets of $\mathbb{R}$ called open balls. The open ball centered at $x \in \mathbb{R}$ with radius $\epsilon \in \mathbb{R}^+$ is a subset $B_\epsilon(x)$ of $\mathbb{R}$ defined as:

$$B_\epsilon(x) = \{y \in \mathbb{R} : |x - y| < \epsilon\}.$$

These are all the points whose distance from x is less than ϵ. The collection of all open balls as x ranges over $\mathbb{R}$ and ϵ ranges over $\mathbb{R}^+$ makes up the building blocks of the topology. The open sets in the topology are all the open balls together with their arbitrary unions and finite intersections.

Example 2. *The standard topologly on* $\mathbb{R}^n$.

We can slightly adjust the previous definition to obtain a topology on $\mathbb{R}^n$. We just have to consider $\vec{x}$ and $\vec{y}$ to be vectors in $\mathbb{R}^n$ and evaluate the distance between them using the standard Eucledian metric. That is if $\vec{x} = (x_1, ..., x_n)$ and $\vec{y} = (y_1, ..., y_n)$ then:

$$B_\epsilon(\vec{x}) = \{\vec{y} \in \mathbb{R}^n : \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} < \epsilon\}$$

is a subset of $\mathbb{R}^n$ with all points of distance less than ϵ from $\vec{x}$. Like previously the topology on $\mathbb{R}^n$ is obtained through arbitrary unions and finite intersections of the set of open balls.

One may notice that the topology we put on a set is by no means unique. If we wish we may even use the topology made up of *all* subsets of a given set. That topology is not preferred because introduces very little structure to our topological space. As we will shortly see it makes the question of continuity rather moot. Topologist prefer topologies that introduce structure on a space that is both intuitive and reflective of the properties they wish to study. The standard metric is useful because Eucledian distance is the natural quantifier of how "near" things are in almost all applied mathematical models. While the information about the actual distance is lost in the generality of the topology, the structure it imposes allows us to talk about important global properties of spaces such as path-connectednes and compactness.

Having a topology on $\mathbb{R}^n$ is all well and good but in this dissertation we shall work with surfaces and triangulations of surfaces in $\mathbb{R}^2$ and $\mathbb{R}^3$. If we had to define a topology on them in a similar fashion we would not get far. Luckily subsets of topological spaces can naturally inherit the topology of their superset.

Definition 4. *Let* (X, τ_X) *be a topological space and* A *be a subset of* X. *The subspace topology of* A *is defined as:*

$$\tau_A = \{U \cap A : U \in \tau_X\}.$$

To obtain all open sets of A take the open sets in X and remove from them all points which are not in A. Going further we will consider any surface embedded in $\mathbb{R}^2$ and $\mathbb{R}^3$ to have the subspace topology of the standard topology unless otherwise stated.

We are finally ready to present the definition of a continuous function.
Definition 5. *A function $f : X \to Y$ is said to be continuous when the preimage of an open set in Y is an open set in X.*

In formal notation if $U \in Y$ is open in Y then $f^{-1}(U)$ is open in X. This definition captures the intuitive understanding we have of continuity - if we "slightly adjust" the output of a function in Y then there should be only a "slight change" in input in X. The "slight change" is formalised by considering all points in a single open set, as we can think of them as being "near". This is the reason why continuous functions are colloquially described as manipulating an object in space without glueing together parts of it or making holes in it. Those disrupt the open sets.

So far we have obtained the means of endowing any subset of $\mathbb{R}^n$ with a topology and we have outlined a general recipe for creating continuous function between them - have open sets be the preimage of open sets. We will now introduce our first topological invariant. But first we shall show how we can "move" around in a topological space.
Definition 6. *Let X be a topological space and let $x, y \in X$ be any two points. A path between x and y in X is a continuous function $f : [0, 1] \to X$ such that $f(0) = x$ and $f(1) = y$.*

This is analogous to the definition of a close curve from differential geometry. The main difference being that we have no notion of differentiability or smoothness yet. As an example think of the space X as a surface in $\mathbb{R}^3$ and two distinct points x and y on it. A path between x and y is a curve that starts at x, moves around the surface and ends at y.
Definition 7. *A topological space X is said to be path-connected if there exists a path between any two points $x, y \in X$*

This deceptively simple looking definition actually holds the overarching methodology for reasoning about algebraic invariants of topological spaces. In this case we have employed a two parameter family of utility functions to "measure" a global property of the topological space. The two parameter family is the collection of all paths between all pairs of points.
Proposition 1. *The continuous image of a path-connected space is path-connected.*

Notice that in this definition we have implied surjectivity. Indeed if X, Y are topological spaces such that X is path-connected and Y is not and $f : X \to Y$ is a continuous

function then all we can say is that $im(f)$ is path-connected.

Lastly we must explore how topological spaces are viewed by topologist. After all if two spaces share all of their topological properties should they be considered different? See coffee mug and dognut example []. We shall make this precise with the notions homeomorphism and homotopy equivalence.

Definition 8. *Two topological spaces X and Y are said to be homeomorphic when there exists a bijetion $f : X \to Y$, such that f and f^{-1} are continuous. Furthermore the function f is called a homeomorphism between X and Y.*

Homeomorphism is the appropriate equivalence relation for topological spaces. For example if we have two homeomorphic topological spaces X and Y and we know one of them has a particular topological property, then the other one will have it as well. Conversely if we know that a space is path connected and another one isn't, then they are not homeomorphic.

In the realm of algebraic topology there is another equivalence relation that is more flexible than this and still preserve the algebraic invariants of spaces. Before presenting it we must first introduce homotopy of functions.

Definition 9. *Let X and Y be two topological spaces. Let $f, g : X \to Y$ be two continuous functions. We say that f and g are homotopic when there exists a third continuous function $H : X \times [0, 1] \to Y$ such that:*

- $H(x, 0) = f(x), \forall x \in X$
- $H(x, 1) = g(x), \forall x \in X$

We can think of homotopy as a one parameter family of functions $\{f_t\}_{t\in[0,1]}$ such that $f_0 = f$ and $f_1 = g$. Homotopy defines an equivalence relation on all continuous functions between X and Y. This notion can be extended to topological spaces as follows:

Definition 10. *Two topological spaces X and Y are said to be homotopy equivalent if there exist two continuous functions $f : X \to Y$ and $g : Y \to X$ such that:*

- $f \circ g$ is homotopic to i_X
- $g \circ f$ is homotopic to i_Y,

where i_X and i_Y are the identity functions on X and Y respectively.

Intuitively we can think of homotopy as a continuous deformation of the image of one of the functions to the image of the other. One can readily observe that every homeomorphism is a homotopy equivalence. Let $g = f^{-1}$, then $f \circ g = i_X$ and $g \circ f = i_Y$ as f is a bijection and every function is homotopic to itself. However the converse is not true and a homotopy equivalence is not a homeomorphism.

In a way very similar to abstract algebra continuous function play the role of

homomorphisms and homeomorphisms play the role of isomorphisms. Also note that homotopy equivalence does not necessarily preserve all topological properties of a topological space. They do however preserse those which we care about such as path-connected and algebraic structures defined on the topological spaces. We will prefer to use them instead of homeomorphisms.

I will add more things to this chapter later if I need to

1.3 Building Blocks

THIS CHAPTER IS VERY MUCH UNDER CONSTRUCTION

The most basic building blocks in Computational Algebraic Topology are Abstract Simplical Complexes (ASC). An ASC is a generalisation of a discrete graph and is defined as follows.
Definition 11. *Given a set V, an Abstract Siplicial Complex called Δ on V is a collection of subsets of V such that if $\sigma \in \Delta$ and $\varphi \subseteq \sigma$ then $\varphi \in \Delta$.*

As we are considering these for computational purposes we will only consider ACS of finite sets. Elements of Δ are called simplices. The dimension of a simplex is it's cardinality. We will denote by Δ_n all simplices in Δ of dimension n. Analogous to graph theory simplices of dimension one are called vertices and simplices of dimension two are called edges. To generalise further 2-dimensional simplices are called triangles and 3-dimensional simplices are called polyhedra. Simplices of higher dimension lose their geometric flavour, so will avoid naming them altogether.
Example 3. *Show a simple example of an abstract simplical complex.*
Definition 12. *Let σ be an n-simplex in Δ. A face of σ is any simplex in Δ such that $\varphi \subseteq \sigma$.*

An ASC is closed under taking subsets, so the faces of a simplex are in fact all subsets of the simplex. This is also one of the reasons why high dimensional ASC are avoided in practise. For example an n-dimensional simplex has 2^n faces in the complex.
Definition 13. *Let V be a finite set and Δ an ASC of V. Let $\sigma \in \Delta$. The star of σ is the set of all simplices of which σ is a face. In set theoretic notation:*

$$St(\sigma) = \{\varphi \in \Delta : \sigma \subset \varphi\}.$$

The interplay between continuous mathematics and combinatorics is indeed interesting. For example in this context the star of a vertex plays the role of an open neighbourhood. Unsuprisingly we can define a topology on an ASC called the Alexandroff topology.

Definition 14. *The collection of all start in an ASC Δ is basis for a topology. The topology is finitely generated by $\{St(\sigma)\}_{\sigma\in\Delta}$.*

Say why this topology is used and why it's interesting and how it is used.

While ASC are a purely combinatorial construct, like graphs they do have a geometric realisation.
Definition 15. *The standard geometric n-simplex is the convex hull of the set of endpoints of the standard basis $[1,0,...,0],[0,1,...,0],...,[0,0,...,1]$ in $\mathbb{R}^{n+1}$ defined as:*

$$\Delta^n = \{(t_0,t_1,...,t_n)\in\mathbb{R}^{n+1} : \sum_{i=0}^{n} t_i = 1 \text{ and } t_i \geq 0, \forall i = 0,1,...,n\}$$

We will define the face of an n-simplex analogously as:
Definition 16. *A face of the standard geometric n-simplex is the convex hull of a subset of endpoints of the standard basis $[1,0,...,0],[0,1,...,0],...,[0,0,...,1]$ in $\mathbb{R}^{n+1}$ defined as:*
Example 4. *Show an example of the basic simplices.*

In order to build a fully functional simplical complex we extend by definitions by the following. The union of all faces of Δ^n is the boundary of Δ^n and it is written as $\partial\Delta^n$. The open simplex $\overset{\circ}{\Delta}{}^n$ is just $\Delta^n\backslash\partial\Delta^n$ as is the interior of Δ^n.

We can now define a simplical complex Δ embedded in $\mathbb{R}^n$ as the finite collection of homeomorphic images of simplices of dimension no more than n. Furthermore if $\sigma\in\Delta$ then all of the faces $\varphi\subset\sigma$ must be in Δ, and $\sigma_1,\sigma_2\ in\Delta$ implies that their intersection $\sigma_1\cap\sigma_2$ is either empty of a face of both.

A simplical complex structure of a topological space X with vertex set V and ASC Δ defined on V is a collection of homeomorphic maps $\{f_\sigma : \Delta^{|\sigma|}\to X\}_{\sigma\in\Delta}$ such that:

- If f_σ is one of the maps and $\varphi\subset\sigma$ then the image of f_φ is a subset of the image of f_σ.
- If f_σ is one of the maps no other map maps to the image of the restriction $f_\sigma|\overset{\circ}{\Delta}{}^{|\sigma|}$.
- If f_{σ_1} and f_{σ_2} are such maps then then the intersection of their images $im(f_{\sigma_1})\cap im(f_{\sigma_2})$ is the image $im(f_\varphi)$ where φ is a face of both.

Theorem 1. *Geometric Realisation Theorem. Every abstract simplicial complex of dimension d has a geometric realisation in $\mathbb{R}^{2d+1}$.*

1.4 Vector Spaces, Quiver Diagrams and Barcode Diagrams

THIS CHAPTER IS VERY MUCH UNDER CONSTRUCTION

Should I define a vector space, bases, etc.?

Should I define a vector space, bases, etc.?

Suppose we have a number of vector spaces $(V_1, V_2, ..., V_n)$

Suppose we have a number of vector spaces $(V_1, V_2, ..., V_n)$ together with linear maps $(f_1, f_2, ..., f_{n-1})$ that that map between consecutive vector spaces like follows :
$f_i : V_i \to V_{i+1}, \forall i = 1, 2, ..., n-1$.

A quiver representation is a directed multigraph where the vertices are sets and directed edges are function between sets. In our case the vertices will be vector spaces and the edges linear maps. The quiver diagram of the configuration we just described looks as follows:

$$V_1 \xrightarrow{f_1} V_2 \xrightarrow{f_2} ... \xrightarrow{f_{n-1}} V_n$$

"This sounds weird, fix it." Not that we can always extend any sequence of vector spaces with the null vector space and the null maps as follows:

$$0 \longrightarrow ... \longrightarrow 0 \longrightarrow V_1 \xrightarrow{f_1} V_2 \xrightarrow{f_2} ... \xrightarrow{f_{n-1}} V_n \longrightarrow 0 \longrightarrow ... \longrightarrow 0$$

A barcode diagram is a digram that shows which shows how the basis elements of the vector spaces evolve as they get mapped through the linear functions once we commit to particular basis elements for each vector space.

Show a barcode diagram.

A Chain Complex is a quiver representation where the image of each maps is a subset of the kernel of the next one.

$$... \longrightarrow V_1 \xrightarrow{d_1} V_2 \xrightarrow{d_2} ... \xrightarrow{d_{n-1}} V_n \longrightarrow ...$$

This example is a chain complex when $im(d_k) \subseteq ker(d_{k+1})$. As the image is a subset of the kernel the we can equivalently write this as the composition $d_{k+1}d_k = 0$. In practical terms once we commit to baseis multiplying consecutive matricies will equal the zero

matrix. An important property of the barcode diagram of chain complexes is that no line can be longer than two units!

Example 5. *A Simple Chain Complex*

Let us now for simplicity and demonstrational purposes assume that each V_i is isomorphic to $\mathbb{R}^n$ for some $n \in \mathbb{Z}$.

An exact sequence is a chain complex where $im(d_k) = ker(d_{k+1})$. Exact sequences are useful because of the nice properties like ...

The homology of a chain complex is defined as a quantifier of how far a chain complex is from being an exact sequnce. It is defined as: $H_k = ker(d_{k+1})/im(d_k)$

Let V be a vector space and W a subspace of V. A coset of W is the set $v + W = \{v + w : w \in W\}$.

A quotient in a vector space is defined in the following way:

$$V/W = \{v + W : v \in V\} = \{\{v + w : w \in W\} : v \in V\}$$

Show a picture of the cosets.

Luckily in $\mathbb{R}^n$ we have the following theorem: $\mathbb{R}^n/\mathbb{R}^m \simeq \mathbb{R}^{n-m}$ where we have slightly abused notation as $\mathbb{R}^m$ can not be a subspace of $\mathbb{R}^n$, but we consider it isomorhpic to one for $m \leq n$.

$$\mathbb{R}^3 \longrightarrow \mathbb{R}^2 \longrightarrow \mathbb{R}^4$$

1.5 Homology

1.6 Reeb Graph & Contour Tree

1.7 Contour Trees

Lemma 1. *In a tree with no vertices of degree two at least half of the vertices are leaves.*

Proof. Let $T = (V, E)$ be a tree with no vertices of degree two and let $L \subseteq V$ be the set of all leaves. As all leaves have degree one we have that $L = \{u \in V : d(u) = 1\}$. Furthermore for any tree we know that $|E| = |V| - 1$. Let us now use the handshake lemma:

$$\sum_{u \in V} d(u) = 2|E| = 2(|V| - 1) = 2|V| - 2.$$

We will not separe the sum on the leftmost hand side of the equation in two parts. One for the vertices vertices in L and one for the vertices in $V \backslash L$.

$$\sum_{u \in L} d(u) + \sum_{u \in V \backslash L} d(u) = 2|V| - 2.$$

All the vertices in L are leaves. By definition the degree of a leaf is one. Therefore $\sum_{u \in L} d(u) = |L|$. This leads us to the following:

$$|L| + \sum_{u \in V \backslash L} d(u) = 2|V| - 2$$

$$|L| = 2|V| - 2 - \sum_{u \in V \backslash L} d(u).$$

There are no vertices in T of degree two and all vertices of degree one are in L. This means that all vertices in $V \backslash T$ have degree at least three. We can conclude that:

$$\sum_{u \in V \backslash L} d(u) \geq \delta(T - L).|V \backslash L| = 3(|V| - |L|)$$

Combining this with the previous equation we obtain that:

$$|L| \leq 2|V| - 2 - 3(|V| - |L|)$$

$$|L| \leq 2|V| - 2 - 3|V| + 3|L|$$

$$-2|L| \leq -|V| - 2$$

$$|L| \geq \frac{|V|}{2} + 1$$

Which is exactly what we set out to proove.

□

Lemma 2. *There are at least k vertices for every vertex of degree k in a tree.*

Proof. Let T be a tree and $u \in V(T)$ be a vertex in it. As any tree can be rooted, let us root T at u and call the new directed tree T_u. Let $U = \{u_1, u_2, ..., u_k\}$ be the neighbours of u. For each $u_i \in U$ if u_i is not a leaf let u_i be one of it's children. Repeat this process

until every u_i is a leaf. This is possible because T is finite. All of the u_i are distinct, for otherwise there would be a cycle in T.

□

Chapter 2

Something "W" This Way Comes!

2.1 Height graphs

A height graph is a graph $G = (V, E)$ together with a real valued function h defined on the vertices of G. In the case of heights graphs that are Contour Trees we will impose the Morse Theory restriction that all vertices have unique heights. In other words $h(u) \neq h(v)$ for all $u, v \in V(G)$ where $u \neq v$. The function h naturally induces a total ordering on the vertices. From now on we will assume the vertices of G are given in ascending order. That is to say, $V(G) = \{v_1, v_2, ..., v_n\}$ where $h(v_1) < h(v_2) < ... < h(v_n)$. This lets us work with the indices of the vertices without having to compare their heights directly. In this notation $h(v_i) < h(v_j)$ when $i < j$.

Introducing the height function allows us to talk about ascending and descending paths. A path in the graph is a sequence of vertices $(u_1, u_2, ..., u_k)$ where $u_i \in V(G)$ for $i \in \{1, 2, ..., k\}$ and $u_i u_{i+1} \in E(G)$ for $i \in \{1, 2, ..., k-1\}$. Furthermore a path in a height graph is ascending whenever $h(u_1) < h(u_2) < ... < h(u_k)$. Conversely if we traverse the path in the opposite direction it would be descending. We will simply call these path monotone whenever we wish to avoid committing to a specific direction of travel.

What we are interested in are the paths in the height tree which form a zigzag pattern. As shown in fig[] they can be decomposed into monotone paths of alternating direction that share exactly one vertex. More formally, if P is a path in a height tree we can always decompose it into vertexwise maximal monotone subpaths $(P_1, P_2, ..., P_k)$ such that $P_i \subseteq P$, $|P_i \cap P_{i+1}| = 1$ and $P_i \cup P_{i+1}$ is not a monotone path for $i \in \{1, 2, ..., k-1\}$ and $k \geq 1$.

One way to characterise paths in a height tree is by the number of subpaths in their monotone path decomposition. The maximum path with respect to this property is precisely the lower bound on the parallel algorithm introduced in []. As a special case we must note that paths that can be decomposed into less than four monotone paths do not pose an algorithmic problem. To simplify this characterisation note that the number of subpaths in the monotone decomposition is exactly the number of vertices in which we change direction as we traverse the path. We shall name those special vertices kinks.

A kink in a path is a vertex whose two neighbours are either both higher or both lower. Given the path $(u_1, u_2, ..., u_k)$ an inside vertex $u_i \neq u_1, u_k$ is a kink when

$h(u_i) \notin \big(min(h(u_{i-1}),\ h(u_{i+1}),\ max(h(u_{i-1}),\ h(u_{i+1})\big)$. To avoid cumbersome notation in this context we shall adopt a slight abuse of notation and in the future write similar statements as $h(u_i) \notin$ or $\in \big(h(u_{i-1}),\ h(u_{i+1})\big)$ where it will be understood that the lower bound of the interval is the smaller of the two and the upper bound the larger.

We can use the number of kinks in a path to define a metric on it. Intuitively this is similar to how the length of a path measures the number of edges between it's vertices. We will make an analogous definition of the w-length of a path as the number of inside vertices which are kinks. Let us denote a path from u to v as $u \rightsquigarrow v$ and with $d(u \rightsquigarrow v)$ measure the length of the longest path between u and v and with $w(u \rightsquigarrow v)$ the path with the largest number of kinks (or the longest w-path). One immediate observation we can make is that $w(u \rightsquigarrow v) < d(u \rightsquigarrow v)$ for any two vertices in any graph. This follows from that fact that the longest path between u and v also has the largest number of vertices. A path with k vertices has length $k-1$ and $k-2$ internal vertices which may or may not be kinks.

2.2 Height trees

We will not restrict our attention to height trees. Those are unsurprisingly height graph which are trees. The first key property of trees will make use of is that there is a unique path between any two vertices. This allows us to slightly simplify some of our notation. Instead of $d(u \rightsquigarrow v)$ and $w(u \rightsquigarrow v)$ we will write $d(u, v)$ and $w(u, v)$ respectively.

We are now fully prepared to unveil that which we seek - the longest w-path in a tree (the one with the most kinks). As there is a unique path between any two vertices this can be posed as an optimisation problem as follows:

$$\max_{u,v \in V(T)} \{w(u \rightsquigarrow v)\} = \max_{u,v \in V(T)} \{w(u, v)\}$$

The search space is quadratic in the number of vertices and this can be computed by running a modified version of Breadth First Search (BFS) from every vertex in the tree. This modified BFS computes the w-distances from a starting vertex to all others. The presudocode for this modification is presented in []. The running time for this is $O(n^2)$ and is far from satisfactory given that the actual algorithm for construction the contour tree runs in time $O(n\alpha(n))$. This is because in practical terms a $O(n^2)$ algorithm is completely unusable on datasets which a near linear time algorithm can process.

And indeed we can do better. As the reader may have noticed the definitions we have made so far are analogous to the task of computing the longest path between any two vertices of a tree. This is completely intentional as we will demonstrate how algorithms

for computing the longest path in a tree can be modified to produce the longest w-path instead. Finding the longest path of a graph in the general case is an *NP-hard*. Fortunately the Contour Tree is a tree. The longest path in a tree is known in the literature as it's diameter and has a polynomial time algorithm. The two most popular linear time algorithms found in the literature I will denote as Double Breadth First Search (2xBFS) and Dynamic Programming (DP). We will now take a look at how these algorithms work and hint at how they can be adapted in the next chapter.

2.2.1 Double Breadth First Search

This algorithm works in two phases. First it picks any vertex in the tree, say s, and finds the one farthest from it using Breadth First Search (BFS). Let us call that vertex u. In the second phase it runs a second BFS from u and again records the farthest one from it. Let us call that v. The output of the algorithm is the pair of vertices (u, v) and the distance between them, $d(u, v)$. That distance is the diameter of the tree.

This algorithm has linear time complexity as it consists of two consecutive linear graph searches. It's correctness if a direct consequence of the following Lemma.
Lemma 3. *Let s be any vertex in a tree. Then the most distant vertex from s is an endpoint of a graph diameter.*

The proof of this Lemma can be found at []. In the next section we shall demonstrate how this proof can be modified to produce a near optimal algorithm linear time algorithm for finding a path whose w-length is bounded from bellow by the w-diameter of a tree.

2.2.2 DP

The second approach is based on the Dynamic Programming paradigm. It is most often applied to optimisation problems that exhibit recursive substructures of the same type as the original problem. The key ingredients in developing a dynamic-programming algorithm are [Into to Algorithms]:

1. Characterise the structure of the optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of the optimal solution.

Naturally, trees exhibit optimal substructure through their subtrees. For our intents and purposes we shall define a subtree as a connected subgraph of a tree. We will only consider rooted trees in the context of this algorithm and we must define the analogously. In a rooted tree let v and u be two vertices such that v is the parent and u

is the child. We shall define the subtree rooted at u as the maximal (vertex-wise) subgraph of T that contains u but does not contain v. We will denote it as T_u. Clearly the rooted subtree at u is smaller than T as it does not contain at least one of the vertices of the T namely - v. This definition will allows us to recursively consider all subtrees of a rooted tree $\{T_u\}_{u \in V(G)}$. Also note that if all vertices in T_u except u inherit their parent from T then T_u is also a rooted subtree - it's root being u.

To continue we will need to define two functions on the vertices T. Let $h(u)$ be the height of the subtree rooted at u. The height is defined as the longest path in T_u from u to one of the leaves of T_u. We will also define $D(u)$ longest path in T_u. The function we will maximize is $D(s)$ where s is the root of T. With these two function we are now ready to recursively define the value of the optimal solution:

$$D(v) = max\left\{ \max_{u \in N(v)} \Big(D(u) \Big), \max_{u,w \in N(v)} \Big(h(u) + h(w) + 2 \Big) \right\}.$$

The base case for this recursive formula is at the leaves of T. If u is a leaf of T then $V(T_u) = \{u\}$. This allows us to set $h(u) = 0$ and $D(u) = 0$ and consider all leaves as base cases. We are guaranteed to reach the base case as each subtree is strictly smaller and we bottom out at the leaves.

This algorithm can be implemented in linear time through Depth First Search (DFS) by using two auxilary arrays that hold the values for $h(u)$ and $D(u)$ for every $u \in V(T)$. We shall omit the proof of correctness and running time of this algorithm in favour of presenting ones for the modified version directly.

2.3 W Diameter Detector

2.3.1 Linear Time Algorithm - W-BFS

We shall first explore how we can modify the Double Breadth First Search algorithm to computer the w-diameter of a height tree. The new algorithm will follow exactly the same steps, except it will run a modified version of BFS that computes w-distance [see algorithm next page]. What the algorithm does is it runs a BFS from any vertex in the graph and records the leaf that is farthest in terms of w-length. This furthest leaf is guaranteed to be either the endpoint of a path in the whose w-length least that of the actual w-diameter of the tree minus two.

Before proving the correctness of the algorithm we must first establish two useful properties that relate the w-length of a path to it's subpaths.

Definition 17. *Subpath Property*

Algorithm 1 Computing the W Diameter of a Height Tree.

```
1: function W_BFS(T, root)
2:     root.d = 0
3:     root.π = root
4:     furthest = root
5:     Q = ∅
6:     Enqueue(Q, root)
7:     while Q ≠ ∅ do
8:         u = Dequeue(Q)
9:         if u.d ≥ furthest.d then
10:            furthest = u
11:        for all v ∈ T.Adj[u] do
12:            if v.π == ∅ then
13:                v.π = u
14:                if h(u) ∉ (h(v), h(u.π)) then
15:                    v.d = u.d + 1
16:                else
17:                    v.d = u.d
18:                Enqueue(Q, v)
19:    Return furthest
20: function CALCULATE_W_DIAMETER(T)
21:    s = <any vertex>
22:    u = W_BFS(T, s)
23:    v = W_BFS(T, u)
24:    return v.d
```

Let $a \rightsquigarrow b$ be a path and $c \rightsquigarrow d$ it's subpath. Then $w(a,b) \leq w(c,d)$.

This property follows from the fact that all kinks of the path from c to d are also kinks of the path from a to b.

Definition 18. *Path Decomposition Property Property*

Let $a \rightsquigarrow b$ be the path $(a, u_1, u_2, ..., u_k, b)$ and u_i be an inside vertex for any $i \in \{1, 2, ..., k\}$. Then:

$$w(a,b) = w(a,u_i) + w(u_i,b) + w_{a \rightsquigarrow b}(u)$$

where:

$$w_{a \rightsquigarrow b}(u_i) = \begin{cases} 0 : \text{if } h(u_i) \in \big(h(u_{i-1}),\ h(u_{i+1})\big) \ // \ u_i \text{ is not a kink} \\ 1 : \text{otherwise } // \ u_i \text{ is a kink.} \end{cases}$$

Indeed u_i can be a kink in the path from a to b, but it cannot be a kink in the paths from a to u_i and from u_i to b because it is an endpoint of both. All other kinks are counter by either $w(a,u_i)$ or $w(u_i,b)$. To account for this in future proof we must consider additional cases for both possible values of $w_{a \rightsquigarrow b}(u_i)$ when using this property.

Lemma 4. *The Algorithm produces the endpoints of a path who is at most 2 kinks shy of being the kinkiest path in the tree.*

Proof. After running the BFS twice we obtain two vertices u and v such that:

$$w(s,u) \geq w(s,t), \forall t \in V(T) \tag{2.1}$$

$$w(u,v) \geq w(u,t), \forall t \in V(T) \tag{2.2}$$

Furthermore let a and b be two leaves that are the endpoints of a path that is a W diameter. For any such pair we know that:

$$w(a,b) \geq w(c,d), \forall c,d \in V(T) \tag{2.3}$$

By this equation we have that $w(a,b) \geq w(u,v)$. Our goal in this proof will be to give a formal lower bound on $w(u,v)$ terms of $w(a,b)$. To this end let t be the first vertex in the path between a and b that the first BFS starting at s discovers. From this description it is clear that t cannot be a or b unless s is equal to a or b.

The proof can then be split into several cases.

Case 1. When the path from a to b does not share any vertices with the path from s to u.

Case 1.1. When the path from u to t goes through s.

In this case $s \rightsquigarrow u$ is a subpath of $t \rightsquigarrow u$, which in turn means that $w(t,u) \geq w(s,u)$. By equation 2.2 we also have that $w(s,u) \geq w(s,a)$. We can therefore conclude that $w(t,u) \geq w(a,t)$ as $s \rightsquigarrow a$ is a subpath of $t \rightsquigarrow a$.

Now via path decomposition of $a \rightsquigarrow b$ and $u \rightsquigarrow b$ at t have that:

$$w(a,b) = w(b,t) + w(t,a) + x$$
$$w(u,b) = w(b,t) + w(t,u) + y.$$

Where $x, y \in \{0,1\}$ depending on whether there is a kink at t for the path from a to b and from u to b respectively. As $w(t,u) \geq w(a,t)$ we can show that:

$$w(u,b) \geq w(b,t) + w(t,a) + y$$
$$w(u,b) \geq w(b,t) + w(t,a) + x - x + y$$
$$w(u,b) \geq w(a,b) - x + y$$
$$w(u,b) \geq w(a,b) + (y - x)$$

But as $w(u,v) \geq w(u,b)$ (by equation 2.2) we obtain that:

$$w(u,v) \geq w(a,b) + (y - x)$$

Considering all possible values that x and y can take, we can see that the minimum value for the right hand side of the equation is at $y = 0$ and $x = 1$. The final conclusion we may draw is that $w(u,v) \geq w(a,b) - 1$.

Case 1.2. When the path from u to t does not go through s.

If the path from u to t does not go through s then the paths $s \rightsquigarrow t$ and $s \rightsquigarrow u$ have a common subpath. Let s' be the last vertex in that subpath. We will be able to reduce this case to the previous one by using s' in the place of s. We must only account for a situation where s' is a kink for one of the paths and not the other. We know that $w(t,u) \geq w(s',u)$ (as a subpath) and through path decomposition we obtain that:

$$w(s,a) = w(s,s') + w(s',a) + x$$
$$w(s,u) = w(s,s') + w(s',u) + y$$

We know that $w(s,u) \geq w(s,a)$ and therefore:

$$w(s,s') + w(s',u) + y \geq w(s,s') + w(s',a) + x$$
$$w(s',u) + y \geq w(s',a) + x$$
$$w(s',u) \geq w(s',a) + (x-y)$$

Since $w(t,u) \geq w(s',u)$ we can further conclude that:

$$w(t,u) \geq w(s',a) + (x-y)$$

From the fact that $t \rightsquigarrow a$ is a subpath of $s' \rightsquigarrow a$ it follows that $w(s',a) \geq w(t,a)$. This lets us obtain that:

$$w(t,u) \geq w(t,a) + (x-y)$$

Now we are ready to proceed in a similar fashion as the previous case. We will decompose the paths from b to a and from b to u at the vertex t as follows:

$$w(b,a) = w(b,t) + w(t,a) + z$$
$$w(b,u) = w(b,t) + w(t,u) + w$$
$$w(b,u) \geq w(b,t) + w(t,a) + (x-y) + w$$
$$w(b,u) \geq w(b,t) + w(t,a) + z - z + (x-y) + w$$
$$w(b,u) \geq w(a,b) - z + (x-y) + w$$
$$w(b,u) \geq w(a,b) + (x-y) + (w-z)$$

The minimum value for the right hand side of this equation is at $x, w = 0$ and $y, z = 1$. Now as $w(u,v) \geq w(u,b)$ we finally obtain that:

$$w(u,v) \geq w(a,b) - 2$$

Case 2. When the path from a to b shares at least one vertex with the path from s to u.

We can do a path decomposition as follows:

$$w(s,u) = w(s,t) + w(t,u) + x$$
$$w(s,a) = w(s,t) + w(t,a) + y$$

As $w(s,u) \geq w(s,a)$ (by equation 2.2)we obtain that:

$$w(t,u) \geq w(t,a) + (y - x)$$

This again leads us to:

$$w(b,a) = w(b,t) + w(t,a) + z$$
$$w(b,u) = w(b,t) + w(t,u) + w$$
$$w(b,u) \geq w(b,t) + w(t,a) + (x - y) + w$$
$$w(b,u) \geq (w(b,t) + w(t,a) + z) - z + (x - y) + w$$
$$w(b,u) \geq w(a,b) - z(x - y) + w$$
$$w(b,u) \geq w(a,b) + (x - y) + (w - z)$$

Where similarly to the previous case we obtain that:

$$w(u,v) \geq w(a,b) - 2$$

Based on these cases we can conclude that for any input tree the algorithm will produce a path that is at most two less than the actual maximum path.

□

Lemma 5. *The time complexity of the algorithm is $O(|V|)$.*

Proof. The modified BFS function has the same time complexity of BFS as all we have added is an "*if, then, else*" statement. The time complexity of BFS is $O(|V| + |E|)$, but in a tree $|E| = |V| - 1$, so the overall complexity is $O(2|V| - 1) = O(|V|)$. Running the modified BFS function twice is still linear, thus the overall complexity of the algorithm is linear as well. □

Lemma 6. *The space complexity of the algorithm is $O(|V|)$.*

Proof. Completely analogous to the standard BFS algorithm, this algorithm uses the same amount of memory in the standard memory model. □

2.3.2 Attempts at resolving the accuracy of 2xBFS

For the intents of purpose of this dissertation the accuracy of this algorithm is sufficient. In large enough data sets this estimate provides enough insight to correlate the observed iterations needed to collapse the split and join trees and the resulting w-diameter of the tree. This is demonstrated in the following section. Regardless of such practical considerations it is still of inherent theoretical interest to understand how we may be able to obtain an accurate linear time algorithm for computing the w-diameter.

One key observation we can make is that on the second run of the BFS we get a w-path that is necessarily longer or equal to one found in the first BFS search. A natural question to ask is whether running the BFS a third, fourth or for that matter nth time would result in the actual w-diameter. On every successive iteration we get a w-path that is longer or equal to the previous one, because w-length is a symmetric path property ($w(a, b) = w(b, a)$). By doing this we can hope that we will eventually obtain a w-path closer to the w-diameter. However there is no guarantee that this will happen. In fact in some cases it is possible that each successive BFS will return the same path over and over again. Obverse how in @TODO fig[] all iterations of BFS go from the vertex u to the vertex v and then from v to u and so on.

Another this we can do is to run the algorithm multiple times from different starting points and keep the maximum value found. This approach unreliable as well as show in @TODO fig[]. That artificial example shows that there can simply be too few starting points which would produce the w-diameter.

What we do know is that starting at any vertex s let the vertices $U = \{u_1, u_2, ..., u_n\}$ be the furthest away and $W = \{w_1, w_2, ..., w_n\}$ be the ones second furthest away. By the proof of the algorithm we know that not necessarily all vertices in those sets would produce a w-diameter. Thus lets us define $R \subseteq U \cup W$ the set of vertices which are an endpoint of a w-diameter. As we have shown we can construct an example where $|R| = 2$ and $|U \cup W|$ is arbitrarily large. Can we then find some property of the vertices in R and pick them out in the first phase? This I will leave open for the future generations to ponder. I hope in doing so all people of the world will unite unite and end all wars and prejudices in order to work towards this common good!

2.3.3 Dynamic Programming Approach

*Idea redefine $N(u) = N(u)/u.\pi$ so you can simplify notation.

While it is encouraging that we have obtained an algorithm that bounds the w-diameter it is also quite unsatisfactory that we were not able to directly obtain it. To remedy this we will resort to modifying the second tree diameter algorithm that we outlined previously. We will use the same optimisation strategy i.e. dynamic programming. We need only make two key changes. Instead of the function $h(u)$ that computes the height of a subtree with root u we will use the function $w(u)$ that stores the longest w-path that starts at the root of the subtree. We will remane the function that stores the value of the optimal solution for subroblems from $D(u)$ to $W(u)$ accordingly. To summarise $W(u)$ returns the length of the largest w-path in the subtree T_u and $w(u)$ the length of the largest w-path in T_u that starts at u.

This may seem like a simple substitution at first glance, but the devil is in the details. As in the previous modification all additional difficulties stem from the difference in combining path lengths and path w-lengths. In the tree diameter scenario path combination is straightforward. For a tree with root s we will first find two distinct children $u, v \in N(s)$ of s such that $h(u)$ and $h(v)$ is maximum amongst all children. Next we will combine them to obtain the longest path that goes through s. This path combination yield the sum $h(u) + h(v) + 2$, where we account for the two additional edges $us, sv \in E(T_s)$. This reasoning of course extends to all subtrees in T.

In the latter case of w-path combinations we must be vigilant of which vertices become kinks in the path combinations. Let us observe a similar scenario where s is the root the tree and $u, v \in V(T_s)$ are two of the children with maximal values for $h(u)$ and $h(v)$. We would ideally like to combine $w(u)$ and $w(v)$ like so: $w(u) + w(v) + w_{u,v}(s)$. This however is not correct! There is a hidden assumption in the sum that the only vertex that can become a kink in this path combination is s. Contrary to this, in fact u and v can also become kinks. Observe that $w(u)$ and $w(v)$ are the w-length of two paths - one starting at u and ending in a leaf of T_u and one starting at v and ending in a leaf of T_v. In the new path both u and v become inside vertices and depending on whether they become kinks or not the sum may further increase by two. To account for this we must also look at the children of u and v.

Let us focus on the children of u and without loss of generality later apply the same reasoning to the children of v. We need a way of detecting the potential of u to become a kink or it's potential w-height. Let $\{u_1, u_2, ..., u_k\}$ be all children of u. We must pick out the with the maximum w-height u' such that u will form a kink with u' and s. If one such exists then the potential w-height $p(u)$ of u is $h(u) + 1$ and $h(u)$ otherwise. In summary $p(u) = \max_{i \in \{1,2,...,k\}} \big(h(u_i) + w_{u_i,s}(u)\big)$.

After computing the potential w-height of all children of s we are not ready to compute the maximum w-path passing through s. This is exactly the path combination with two of the children of s - $u, v \in T_u$ such that $p(u) + p(v) + w_{u,v}(s)$ is maximum. Armed with

the means of combining subproblems we are finally ready to present the recursive formula for the optimal solution of the problem.

$$W(s) = max\left\{ \max_{u \in N(s)} \Big(W(u) \Big), \max_{u,v \in N(s)} \left(\max_{u' \in N(u)} \Big(w(u') + w_{u',s}(u) \Big) + \max_{v' \in N(v)} \Big(w(v') + w_{v',s}(v) \Big) + w_{u,v}(s) \right) \right\}$$

Which one looks better?

$$W(s) = max \left\{ \begin{array}{l} \max\limits_{u \in N(s)} \Big(W(u) \Big), \\ \max\limits_{u,v \in N(s)} \left(\max\limits_{u' \in N(u)} \Big(w(u') + w_{u',s}(u) \Big) + \max\limits_{v' \in N(v)} \Big(w(v') + w_{v',s}(v) \Big) + w_{u,v}(s) \right) \end{array} \right\}$$

Algorithm 2 Computing the W Diameter of a Height Tree.

```
1:  function W_DFS(T, s)
2:      if |T.Adj[s]| == 1 AND s.π ≠ s then
3:          s.W = 0
4:          s.w = 0
5:          return
6:      for all u ∈ T.Adj[s] do
7:          if u.π == ∅ then
8:              u.π = s
9:              W_DFS(T, u)
10:
11:     Array p
12:     for all u ∈ T.Adj[s]/s.π do
13:         p[u] = 0
14:         for all v ∈ T.Adj[u]/u do
15:             p[u] = max(p[u], v.h + w_{v,s}(u))
16:     maxCombine = 0
17:     for all u ∈ T.Adj[s]/s.π do
18:         for all v ∈ T.Adj[s]/s.π do
19:             maxCombine = max(maxCombine, p[u] + p[v] + w_{u,v}(s))
20:
21:     maxSubsolution = 0
22:     for all u ∈ T.Adj[s] do
23:         maxSubsolution = max(maxSubsolution, u.W)
24:
25:     s.W = max(s.maxCombine, s.maxSubsolution)
26: function Calculate_W_Diameter(T)
27:     s = <any vertex>
28:     s.π = s
29:     W_DFS(T, s)
30:     return s.W
```

Lemma 7. *The Algorithm produces the w-diameter of a height tree.*

Proof. TBA □

The complexity of the proposed solution is:

$$O\left(|V| + |E| + \sum_{u \in V} \sum_{v \in N(u)} d(v) + \sum_{u \in V} d(u)^2\right)$$

Where $\sum_{u \in V} \sum_{v \in N(u)} d(v)$ is the loop over all children of children and $\sum_{u \in V} d(u)^2$ is the double loop over all children in the final path combination.

Firstly we can show that:

$$O\left(\sum_{u \in V} \sum_{v \in N(u)} d(v)\right) = O(|V|)$$

This is because as we are in tree, every vertex will be visited exactly once as a child of a child. If it were visited twice then there would be two distinct paths to that vertex which would mean a cycle.

The other argument is more difficult to bound. One thing that is clear is that

$$\sum_{u \in V} d(u)^2 \geq \sum_{u \in V} d(u) = 2|E|$$

This is true because the degree of a vertex is a positive integer and for any $x \in Z^+, x^2 \geq x$. This lower bound shows that it may be possible to obtain linear time complexity. I will demonstrate how we can bound it from above.

A triangle is the complete graph on three vertices. As trees have no cycles they cannot have induced triangles. Therefore for any edge in a tree $uv \in E(T)$ we have that $d(u) + d(v) \leq |V|$. Indeed, if we do not have an induced triangle there are no vertices that $d(u)$ and $d(v)$ count twice. Summing over all edges we get that:

$$\sum_{uv \in E(T)} d(u) + d(v) \leq |E|.|V|$$

The key to solving this is to notice is that if we expand the summation every term $d(u)$ will be present exactly $d(u)$ times (one for each of it's edges). This allows us to obtain that:

$$2|E| \leq \sum_{u \in V(T)} d(u)^2 \leq |E|.|V|$$

Overall for the two sums we have shown that:

$$O\Big(\sum_{u \in V} \sum_{v \in N(u)} d(v)\Big) = O(|V|), \quad O\Big(\sum_{u \in V(T)} d(u)^2\Big) = O(|V|.|E|).$$

Therefore the time complexity of the dynamic programming solution is:

$$O\big(|V| + |E| + |V| + |V|.|E|\big) = O\big(|V|.|E|\big).$$

The running time is quadratic. Theoretically this is no better than a brute force exhaustive search. Despite this we have reasons to believe that it has the potential for better practical performance. The main reason that leads us to this conclusion is that the quadratic behaviour comes from the double loop on the children of all vertices. We know from the **lemma in previous chapter** that in any tree for any vertex of degree d there are at least d distinct leaves. Therefore for any vertex of high degree there will be as many vertices which are base cases for the recursion and will take constant processing time. This behaviour is/is not demonstrated in the next chapter where implementations of both w-diameter algorithms are compared empirically.

[1]

References

[1] D. Parikh, N. Ahmed, and S. Stearns. An adaptive lattice algorithm for recursive filters. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 28(1):110–111, 1980.

Appendices

Appendix A

External Material

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Appendix B

Ethical Issues Addressed