Informatics

Formalizing Graph Trail Properties

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Logic and Computation

eingereicht von

Hanna Elif Lachnitt, B.Sc.
Matrikelnummer 11741619

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Professor Laura Kovács
Mitwirkung: Professor Stefan Szeider
Dr. Martin Suda

Wien, 29. April 2020 [illegible] [illegible]

Hanna Elif Lachnitt   Laura Kovács

Informatics

Formalizing Graph Trail Properties

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Logic and Computation

by

Hanna Elif Lachnitt, B.Sc.

Registration Number 11741619

to the Faculty of Informatics

at the TU Wien

Advisor: Professor Laura Kovács
Assistance: Professor Stefan Szeider
Dr. Martin Suda

Vienna, 29th April, 2020

[illegible] [illegible]

Hanna Elif Lachnitt Laura Kovács

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Hanna Elif Lachnitt, B.Sc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 29. April 2020

[illegible]

Hanna Elif Lachnitt

Danksagung

I would like to express my gratitude to my thesis advisor Professor Laura Kovács, for the continuous support during the work on my thesis and beyond. Furthermore, I want to thank Professor Stefan Szeider for his help with the graph theoretic part of this paper. I am very grateful for the financial support I received through the Helmuth Veith Stipend for the time of my master studies at TU Wien. This made it possible for me to spend two wonderful years at an amazing university. I would like to thank Dr. Anna Prianichnikova, Mihaela Rozman and Professor Georg Weißenbacher for the warm welcome they gave me when I first moved to Vienna.

I want to show my appreciation for the never-ending support of my family, especially for my sisters Charlotte and Marlene. Also I want to thank all my friends that kept encouraging and supporting me.

Acknowledgements

We thank Prof. Byron Cook (AWS) for interesting discussions on reasoning challenges with ordered trails. This work was funded by the ERC Starting Grant 2014 SYMCAR 639270, the ERC Proof of Concept Grant 2018 SYMELS 842066, the Wallenberg Academy Fellowship 2014 TheProSE, the Austrian FWF research project W1255-N23 and P32441, the Vienna Science and Technology Fund ICT19-065 and the Austrian-Hungarian collaborative project 101öu8.

Kurzfassung

Wir prüfen zwei verschiedene Methoden um automatisch über streng monotone fallende oder streng monoton steigende Kantenzüge in kantengewichteten Graphen zu argumentieren. Zunächst präsentieren wir eine Formulierung als prädikatenlogisches Problem und evaluieren die Anwendung automatischer Theorembeweiser. Wir zeigen die Nachteile dieses Ansatzes, durch experimentelle Auswertung der Abhängigkeit der gebrauchten Zeit von der Größe des Graphen. Motiviert von diesen Einschränkungen präsentieren wir einen Beweis für eine untere Grenze der Länge von streng monotonen Kantenzügen, der nicht von der Größe des Graphen abhängt. Wir formalisieren Eigenschaften streng monoton fallender und streng monoton steigender Kantenzüge im Beweisassistenen Isabelle/HOL und verifizieren den Beweis der unteren Grenze der Länge des längsten streng monoton fallenden Kantenzugs. Dafür erweitern wir Isabelles Bibliothek für Graphentheorie mit einem Algorithmus, der die Länge des längsten streng monoton fallenden Kantenzugs von einem Knoten für eine gegebene Verteilung der Kantengewichte berechnet. Des Weiteren beweisen wir in Isabelle, dass in einem ungerichtetem Graphen jeder streng monoton fallende Kantenzug auch ein streng monoton steigender Kantenzug ist. Außerdem präsentieren wir einen konstruktiven Beweis, der eine obere Grenze der minimalen Länge eines streng monotonen Kantenzug in vollständigen Graphen zeigt. Im Zuge dessen geben wir auch einen Algorithmus zur Erzeugung von vollständigen Graphen beliebiger Größe an, die zeigen, dass Graphen mit einer bestimmten maximalen Länge von Kantenzügen existieren.

Abstract

We survey two methods to reason about ordered trails in edge-weighted graphs. Firstly, we provide an encoding in first-order logic in order to apply automated theorem provers to the problem. Conducting experiments on the size of the input instance, we show the disadvantages of this approach. Motivated by this we present a symbolic proof of the lower bound on the length of strictly ordered trails in weighted graphs. Then, we formalize strictly ordered trails in the proof assistant Isabelle/HOL. We express and prove properties of these trails and verify the lower bounds on the length in Isabelle. We do so by extending the graph theory library of Isabelle/HOL with an algorithm computing the length of a longest strictly decreasing graph trail starting from a vertex for a given weight distribution, and prove that in an undirected graph any decreasing trail is also an increasing one. We also present a symbolic proof that shows an upper bound on the minimum length of strictly ordered trails. To this end we present an algorithm to construct graphs that witness the upper bound an implementation of this algorithm.

Contents

CHAPTER 1

Introduction

Many real-world instances of scheduling and planning can be abstracted into graph problems. Therefore, any improvement in formalizing and solving graph problems, in an efficient and automatic manner, is of great importance. This theses aims at increasing our understanding of automated reasoning about strictly ordered trails in edge-weighted graphs. The following problem was brought to our attention by Dr. Byron Cook from Amazon Web Services (AWS).

There are 2020 cities and there is a flight between each pair of cities. The price of a flight is the same in both directions but the prices between two different pairs of cities can never be the same. Is there a route of 2019 cities such that each flight is cheaper than the last one? The same city can be visited several times or a city might not be visited at all.

To illustrate this question consider the following undirected graph K_4, with 4 vertices (cities) where each edge (flight) is annotated with a different integer-valued weight (cost) ranging from $1, \ldots, 6$:

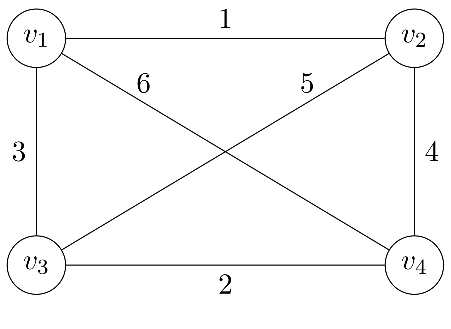


Figure 1.1: Example graph K_4

The property in K_4 that corresponds to a route of cities is a trail. A trail is a sequence of edges in which consecutive edges are incident. Thus, the question we want to address is whether K_4 has a strictly decreasing trail of length 3. The answer is positive, there is a decreasing trail in K_4 starting at vertex v_3, with trail length 3; namely (v_3v_2, v_2v_4, v_4v_3) is such a trail, with each edge in the trail having a higher weight than its consecutive edge in the trail. Similarly, K_4 has decreasing trails of length 3 starting from v_1, v_2, and v_4 respectively.

But is this true for all distributions of weights to the edges? If not, it would mean that we could construct a graph that has 4 vertices and 5 edges, where no vertex is the starting node of a trail of length 3. By enumerating all possibilities it becomes clear that this is not possible. But what about the case where we consider 2020 cities? Enumerating the possibilities becomes quickly practically infeasible. Our goal is to automate the reasoning about such trails.

Thus, as a first step, this thesis explores the possibilities of using an automated reasoning engine on an encoding of the problem that is presented in Section 2.1. The results are presented in Section 2.2. They show that even for relatively small numbers n state-of-the-art theorem provers are not able to prove whether edge-weighted graphs have a strictly decreasing trail. The causes and implications of this phenomenon are further discussed in Section 2.3.

Based on the limitations on automative provers, we aim at formalizing and proving existence of trails of length n, where $n \geq 1$ is a symbolic constant. As such, proving for example that graphs have trails of length 4, for a concrete n, become instances of our approach. For this, we want to consider a more general version of the problem above. Instead of restricting ourselves to decreasing trails of length $n-1$, we consider the problem of finding a longest trail with strictly increasing or decreasing weights in general. Furthermore, we will consider all undirected graphs where any two edges are not labelled with the same weight and not only complete graphs.

Similarly to the theoretical results of [GK73], we show that, given a graph G with n vertices and q edges, there is always a strictly decreasing trail of length at least $2 \cdot \lfloor \frac{q}{n} \rfloor$.This problem of finding a longest trail with strictly increasing or strictly decreasing weights in an edge-weighted graph is an interesting graph theoretic problem [GK73, CCS84, Yus01, DSMP+15], with potential applications to scheduling and cost distribution in traffic planning and routing [CKL19].

Before commencing with automation we want to investigate different ways to make statements about this topic. To this end, we build upon existing works in this area. In particular, the first to raise the question of the minimum length of strictly increasing trails of arbitrary graphs were Chvátal and Komlós [CK70]. Subsequently, Graham and Kletman [GK73] proved that the lower bound of the length of increasing trails is given by $2 \cdot \lfloor \frac{q}{n} \rfloor$. They also gave an upper bound on the minimum length $f(n)$ of increasing trails in complete graphs.

$$f(n) = \begin{cases} n & \text{if } n \in \{3,5\} \\ n-1 & \text{otherwise} \end{cases},$$

Since the ultimate goal of this thesis is to automate these results our proofs diverge from the ones of Graham et al. [GK73], especially since they were made constructive. Therefore, in Chapter 3 all proofs are given and the differences to [GK73] are discussed. To this end we review the basics of graph theory in Section 3.1. Then, Section 3.2 gives a detailed proof of the lower bound of strictly decreasing trails. In Section 3.3 the upper bound in the case of complete graphs is discussed.

In Chapter 4 we present a formalization of strictly-ordered trails in Isabelle/HOL. As our theory builds on the library by Noschinski [Nos13], we summarize the main elements of this theory that are used by us are in Section 4.2. Our addition to this library the theory $Ordered-Trail$ is presented in Section 4.3. This Section contains a formalization of the symbolic proof shown in Section 3.2. Chapter 4 of this thesis was generated from Isabelle/HOL source code using Isabelle's document preparation tool and is therefore fully verified. The source code is available online at `https://github.com/Lachnitt/Formalizing-Graph-Trail-Properties/`.

To conclude, this thesis brings the following contributions.

(1) We formalize strictly increasing trails and provide basic lemmas about their properties in Isabelle/HOL. We also formalize strictly decreasing trails, in addition to the increasing trail setting of [GK73]. We prove the duality between strictly increasing and strictly decreasing trails, that is, any such decreasing trail is an increasing one, and vice versa.

(2) We design an algorithm computing longest ordered paths (Algorithm 3.1), and formally verify its correctness in Isabelle/HOL. We extract our algorithm to a Haskell program code using Isabelle's program extraction tool. Thus, we obtain a fully verified algorithm to compute the length of strictly ordered trails in any given graph and weight distribution. We also provide a C++ program to compute these trails.

(3) We verify the lower bound on the minimum length of strictly ordered trails of arbitrary graphs, and of complete graphs in particular in Isabelle/HOL. Furthermore, we formulate constructive proofs (Algorithm 3.2 and 3.3) to show the upper bound on the minimum length on strictly ordered trails for complete graphs and argue why such is useful for a planned verification in Isabelle/HOL. We provide a C++ program to generate graphs with n vertices that do not contain trails of length n, thus generating counterexamples to show that except for graphs with 3 and 5 there is always a weight configurations such that there is no strictly ordered trail of length n.

(4) We introduce the digital dataset $Ordered-Trail$ formalizing properties of graph trails. Our dataset consists of ∼2000 lines of Isabelle code. As far as we know this is the first formalization of ordered trails in a proof assistant. Our formalization builds upon the $Graph-Theory$ library by Noschinski [Nos13], that is part of the Archive of Formal Proofs (AFP) and already includes many results on walks and general properties of graphs.

CHAPTER 2

First-Order Theory of Graphs

2.1 Encoding

As a first step to address the ordered trail problem presented in Chapter 1, an encoding of it as a logical formula is desirable. We use a many-sorted first-order logic with equality as employed in the SMT-LIB 2.0 standard [BST+10]. This format is widely used by automated theorem provers [KV13], [DMB08], [BCD+11].

Our encoding is shown in Listing 2.1. We introduce two sorts: V, short for vertex and W, short for weight (lines 1-2). We do not have to restrict the sort of our objects further, for example by making them integers or reals. There is no need to add them together or to compare them to a constant. The only thing that is necessary is to impose an order on the weights such that we can test if a path is decreasing. To this end the uninterpreted function *great* ($>$) is added in line 4.

We impose restrictions on *great*, the relation should be irreflexive, transitive and trichotomous, i.e. a strict total order. These restrictions are added by the following assertions:

property	property as a logical formula	line number in 2.1
irreflexive	$\forall (x:W).\neg(x>x)$	7
transitive	$\forall (x:W)(y:W)(z:W).x>y \wedge y>z \rightarrow x>z$	8
trichotomous	$\forall (x:W)(y:W).x>y \vee y>x \vee x=y$	10

A second function *cost* maps two objects of sort V to one of sort W. The following table shows the meaning of the assertions made on the interplay of *cost* and *great*.

property	property as a logical formula	Intuitive meaning	line nr.
symmetric	$\forall(x:V)(y:V).cost(x,y) = cost(y,x)$	The costs in one direction should be the same as in the backward one	13
distinctive	$\forall(w:V)(x:V)(y:V)(z:V).$ $(w = x \wedge y = z)$ $\vee\ (y = w \wedge x = z)$ $\vee\ cost(x,y) \neq cost(w,z)$	All weights are distinct. Backward directions have to be excluded.	14

The above conditions are universal and do not depend on the specific number of vertices n. The following two conditions however are depended on n.

property	property as a logical formula	Intuitive meaning	line nr.
distinct universe	$\forall v_i, v_j \in \{(v_0:V)\dots(v_n:V)\}.v_i \neq v_j$	There are at least n distinct vertices in the domain.	21
trail of length $n-1$	$\forall(v_0:V)\dots(v_{n-1}:V).$ $\neg(v_0 \neq v_1$ $\wedge\ cost(v_0,v_1) > cost(v_1,v_2) \wedge\ v_1 \neq v_2$ $\wedge\ \dots$ $\wedge\ cost(v_{n-3},v_{n-2}) > cost(v_{n-2},v_{n-1})$ $\wedge\ v_{n-2} \neq v_{n-1})$	There is no trail of length $n-1$ such that the weights are decreasing.	24

It is important to note that the assertion *trail of length n-1* states that there is **no** trail of length $n-1$. This has direct implications on how to interpret the result a theorem prover returns on the encoding. If the prover answers that the set of clauses is *unsatisfiable* there exists a trail, whereas there is no trail if it returns *valid*.

The encoding can also be used to search for trails of length n with only minimal adjustments. In that case the assertion *trail of length n-1* is replaced and exchanged by a modified version that prolongs the trail by another element. Since in the assertion *distinct universe* we only request that there are n different elements in the domain but not that the elements of the trail are different that assertion stays unchanged.

property	property as a logical formula	Intuitive meaning
trail of length n	$\forall(v_0:V)\dots(v_n:V).$ $\neg(v_0 \neq v_1$ $\wedge\ cost(v_0,v_1) > cost(v_1,v_2) \wedge\ v_1 \neq v_2$ $\wedge\ \dots$ $\wedge\ cost(v_{n-2},v_{n-1}) > cost(v_{n-1},v_n)$ $\wedge\ v_{n-1} \neq v_n)$	There is no trail of length n such that the weights are decreasing.

The code in Listing 2.2 illustrates this matter modifying lines 24 to 36 of Listing 2.1. Instead of searching for trails of length 5 in a graph with 6 vertices it now considers trails of length 6. As before, a solver will return unsatisfiable if there is always a trail of length n in any combination of weights and starting vertices and valid if that is not the case. In the following we will use the modified code to find counterexamples to the statement that every graph has a decreasing trail of length n.

```
(declare-sort V 0)
(declare-sort W 0)

(declare-fun great (W W) Bool)
(declare-fun cost (V V) W)

(assert (forall ((x W)) (not (great x x))))
(assert (forall ((x W) (y W) (z W))
  (or (not (great x y)) (not (great y z)) (great x z))))
(assert (forall ((x W) (y W))
  (or (great x y) (great y x) (= x y))))

(assert (forall ((x V) (y V)) (= (cost x y) (cost y x))))
(assert (forall ((w V) (x V) (y V) (z V)) (or
  (and (= w x) (= y z))
  (and (= y w) (= x z))
  (not (= (cost x y) (cost w z))))))

; In the following n = 6

(assert (exists ((d0 V) (d1 V) (d2 V) (d3 V) (d4 V) (d5 V))
  (distinct d0 d1 d2 d3 d4 d5)))

(assert (forall ((c0 V) (c1 V) (c2 V) (c3 V) (c4 V) (c5 V))
  (not (and
    (great (cost c0 c1) (cost c1 c2))
    (distinct c0 c1)
    (distinct c1 c2)
    (great (cost c1 c2) (cost c2 c3))
    (distinct c2 c3)
    (great (cost c2 c3) (cost c3 c4))
    (distinct c3 c4)
    (great (cost c3 c4) (cost c4 c5))
    (distinct c4 c5)
  ))
))

(check-sat)
```

Listing 2.1: Encoding of the trail problem

```
(assert (forall ((c0 V) (c1 V) (c2 V) (c3 V) (c4 V) (c5 V) (c6 V))
  (not (and
    (great (cost c0 c1) (cost c1 c2))
    (distinct c0 c1)
    (distinct c1 c2)
    (great (cost c1 c2) (cost c2 c3))
    (distinct c2 c3)
    (great (cost c2 c3) (cost c3 c4))
    (distinct c3 c4)
    (great (cost c3 c4) (cost c4 c5))
    (distinct c4 c5)
    (great (cost c4 c5) (cost c5 c6))
    (distinct c5 c6)
  ))
))
```

Listing 2.2: Changes Listing 2.1 to search for trails of length n

2.2 Results

Our experiments were conducted using the the SMT-solver Z3 [DMB08] on a standard laptop with 1.7 GHz Dual-Core Intel Core i5 and 8 GB 16000 MHz memory. The time limit was set to one hour. For the time measurement GNU time [Fou18] was used.

2.2.1 Trails of Length n-1

When searching for trails of length $n-1$, i.e. as shown in the encoding in Listing 2.1 the limit is reached when $n = 7$.

$n =$	Result	Time
3	unsat	0.020s
4	unsat	0.063s
5	unsat	4.831s
6	unsat	19m 6.175s

As expected the set of formulas is unsatisfiable for every considered instance of n. The rapid, exponential increase in time is explicable when examining the generated proof. All possible combinations of weights and starting vertices are listed and then tried out.

2.2.2 Trails of Length n

When searching for trails of length n, i.e. as shown in the encoding in Listing 2.1 but with the variation described in Listing 2.2, the limit is reached when n equals 5. It is expected that the limit is lower as in Section 2.2.1 because the prover has to try out one step more for every combination of weights and starting vertices.

$n =$	Result	Time
3	unsat	0.082s
4	sat	0.444s
5	unsat	3m 55.313s

We use the command *(get-model)* to construct a graph that has four vertices and does not contain a trail of length 4. This is a direct counterexample to the statement that every weight-distinct graph with n vertices has a decreasing trail of length n.

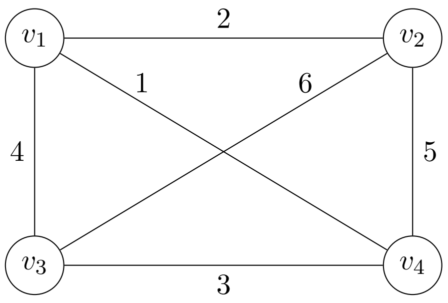


Figure 2.1: K_4 without trails of length 4

2.3 Discussion

Both experimental datasets show that even for small numbers the provers were not able to show the existence of strictly decreasing trails. Looking at the proof they generate it becomes clear that they try out all combinations of weights and starting vertices. This obviously requires a great effort and is not helpful to learn any new ways to approach the problem. Furthermore, up to this point only complete graphs were considered but our goal is to handle all kind of graphs.

We want to emphasize that the limitation described above goes beyond automated provers. In the Isabelle proof assistant, proving that a complete graph with 3 vertices, i.e. K_3, will always contain a strictly decreasing trail of length 3 is quite exhaustive as well, as it requires reasoning about $3! = 6$ possibilities for a distribution of a weight function w and then manually constructing concrete trails:

$$w(v_1,v_2) = 2 \wedge w(v_2,v_3) = 1 \wedge w(v_3,v_1) = 3$$
$$\longrightarrow \mathit{incTrail}\ K_3\ w\ [(v_3,v_2),(v_2,v_1),(v_1,v_3)]$$

Therefore, we need a proof where n is treated as a symbolic value to solve the problem efficiently. Such a proof is presented in Chapter 3. For the formalization we decide to use Isabelle/HOL. It is shortly introduced in Chapter 4.

CHAPTER 3

Symbolic Proof of the Existence of Strictly Ordered Trails

3.1 Graph theory

A *graph* $G = (V, E)$ consists of a set V of *vertices* and a set $E \subseteq V \times V$ of *edges*. A graph is undirected if $(v_1, v_2) \in E$ implies that also $(v_2, v_1) \in E$. A graph is *complete* if every pair of vertices is connected by an edge. A graph is *loopfree* or *simple* if there are no edges $(x, x) \in E$ and *finite* if the number of vertices $|V|$ is finite. Finally, we call a graph $G' = (V', E')$ a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$.

If a graph is equipped with a weight function $w : E \to \mathbb{R}$ that maps edges to real numbers, it is called an *edge-weighted graph*. In the following, whenever a graph is mentioned it is implicitly assumed that this graph comes equipped with an weight function. A vertex labelling is a function $L : V \mapsto \mathbb{N}$.

A *trail of length* k in a graph $G = (V, E)$ is a sequence $(e_1, \ldots, e_k)$, $e_i \in E$, of distinct edges such that two subsequent edges share a common vertex. A *strictly decreasing trail* in an edge-weighted graph $G = (V, E)$ with weight function w is a trail such that $w(e_i) > w(e_{i+1})$. Likewise, a *strictly increasing trail* is a trail such that $w(e_i) < w(e_{i+1})$.

We will denote the length of a longest strictly increasing trail with $P_i(w, G)$. Likewise we will denote the length of a longest strictly decreasing trail with $P_d(w, G)$. In any undirected graph, it holds that $P_i(w, G) = P_d(w, G)$, a result that we will formally verify in Section 4.3.1.

Let $f_i(n) = \min P_i(w, K_n)$ denote the minimum length of an strictly increasing trail that must exist in the complete graph with n vertices. Likewise, $f_d(n) = \min P_d(w, K_n)$ in the case that we consider strictly decreasing trails.

3.2 Lower Bound on the Length of Strictly Decreasing Trails

The proof introduced in the following is based on similar ideas as in [GK73]. However, we diverge from [GK73] in several aspects. Firstly, we consider strictly decreasing instead of strictly increasing trails, reducing the complexity of the automated proof (see Section 4.3). Furthermore, we add tighter bounds than necessary to give a fully constructive proof in terms of an algorithm for computing the length of these trails (see Section 3.2.2).

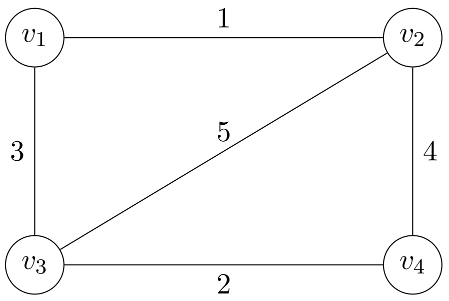
3.2.1 Proof

We start by introducing the notion of a weighted subgraph and then we built on that by specifying a family of labelling functions:

Definition 1 (Weighted Subgraph)**.** *Let $G = (V, E)$ be a graph with weight function $w : E \to \{1, \dots, q\}$ where $|E| = q$. For each $i \in \{0, ..., q\}$ define a weighted subgraph $G^i = (V, E^i)$ such that $e \in E^i$ iff $w(e) \in \{1, ..., i\}$. That is, G^i contains only edges labelled with weights $\leq i$.*

Definition 2 (Labelling Function)**.** *For each $G^i = (V, E^i)$ we define $L^i : E^i \to \{1, \dots, \frac{n(n-1)}{2}\}$ be a labelling function such that $L^i(v)$ is the length of a longest strictly decreasing trail starting at vertex v using only edges in E^i.*

Example 1. *In Figure 3.1 the example graph from Figure 1.1 is revisited to illustrate these definitions.*



Decreasing trails from v_3 are:

(v_3v_4), (v_3v_1, v_1v_2),

(v_3v_2, v_2v_1),

(v_3v_2, v_2v_4, v_4v_3)

Therefore, $L^5(v_3) = 3$

Decreasing trails from v_1 are:

(v_1v_2), (v_1v_3, v_3v_4)

Therefore, $L^5(v_1) = 2$

Figure 3.1: Graph G^5 with labelling function L^5

We need to prove the following property.

Lemma 1. *Let $i < q$ and G^i a labelled subgraph of G. Then, adding the edge labelled with $i + 1$ to the graph G_i increases the sum of the lengths of strictly decreasing trails at least by 2, i.e., $\sum_{v \in V} L^{i+1}(v) \geq \sum_{v \in V} L^i(v) + 2$.*

Proof. Let e be the edge labelled with $i+1$ and denote its endpoints with u_1 and u_2. It holds that $E^i \cup \{e\} = E^{i+1}$, therefore the graph G^{i+1} is G^i with the additional edge e. As $w(e') < w(e)$, for all $e' \in E^i$ we have $L^{i+1}(v) = L^i(v)$ for all $v \in V$ with $u_1 \neq v, u_2 \neq v$. It also holds that $L^{i+1}(u_1) = \max(L^i(u_2)+1, L^i(u_1))$ because either that longest trail from u_1 can be prolonged with edge e ($i+1$ will be greater than the weight of the first edge in this trail by construction of L^{i+1}) or there is already a longer trail starting from u_1 not using e. We derive $L^{i+1}(u_2) = \max(L^i(u_1)+1, L^i(u_2))$ based on a similar reasoning. See Figure 3.2 for a visualisation.

Note that $L^{i+1}(v) = L^i(v)$ for $v \in V \backslash \{u_1, u_2\}$, because no edge incident to these vertices was added and a trail starting from them cannot be prolonged since the new edge has bigger weight than any edge in such a trail.

If $L(u_1) = L(u_2)$, then $L^{i+1}(u_1) = L^i(u_1)+1$ and $L^{i+1}(u_2) = L^i(u_2)+1$ and thus the sum increases exactly by 2. If $L(u_1) > L(u_2)$ then $L^{i+1}(u_2) = L^i(u_1)+1 \geq L^i(u_2)+2$, otherwise $L^{i+1}(u_1) = L^i(u_2)+1 \geq L^i(u_1)+2$. Thus,

$$\begin{aligned}
&\sum_{v\in V} L^{i+1}(v) \\
&= \sum_{v\in(V-\{u_1,u_2\})} L^{i+1}(v) + L^{i+1}(u_1) + L^{i+1}(u_2) \\
&\geq \sum_{v\in(V-\{u_1,u_2\})} L^{i+1}(v) + L^{i}(u_1) + L^{i}(u_2) + 2 \\
&= \sum_{v\in V} L^{i}(v) + 2.
\end{aligned}$$

□

Lemma 2. $\sum_{v\in V} L^q(v) \geq 2q$.

Proof. By induction, using the property $\sum_{v\in V} L^{i+1}(v) \geq \sum_{v\in V} L^i(v) + 2$ from Lemma 1. For the induction base note that $\sum_{v\in V} L^0(v) = 0$ because G^0 does not contain any edges and thus no vertex has a strictly decreasing trail of length greater than 0. □

We next prove the lower bound on the length of longest strictly decreasing trails.

Theorem 1. *Let $G = (V, E)$ be an undirected edge-weighted graph such that $|V| = n$ and $|E| = q$. Let $w : E \to \{1, \dots, q\}$ be a weight function assuming different weights are mapped to to different edges. Then, $P_d(w, G) \geq 2 \cdot \lfloor \frac{q}{n} \rfloor$ i.e., there exists a strictly decreasing trail of length $2 \cdot \lfloor \frac{q}{n} \rfloor$.*

Proof. Assume that no vertex is a starting point of a trail of length at least $2 \cdot \lfloor \frac{q}{n} \rfloor$, that is $L^q(v) < 2 \cdot \lfloor \frac{q}{n} \rfloor$, for all $v \in V$. Then, $\sum_{v\in V} L^q(v) < 2 \cdot \lfloor \frac{q}{n} \rfloor n \leq 2 \cdot q$. But this is a

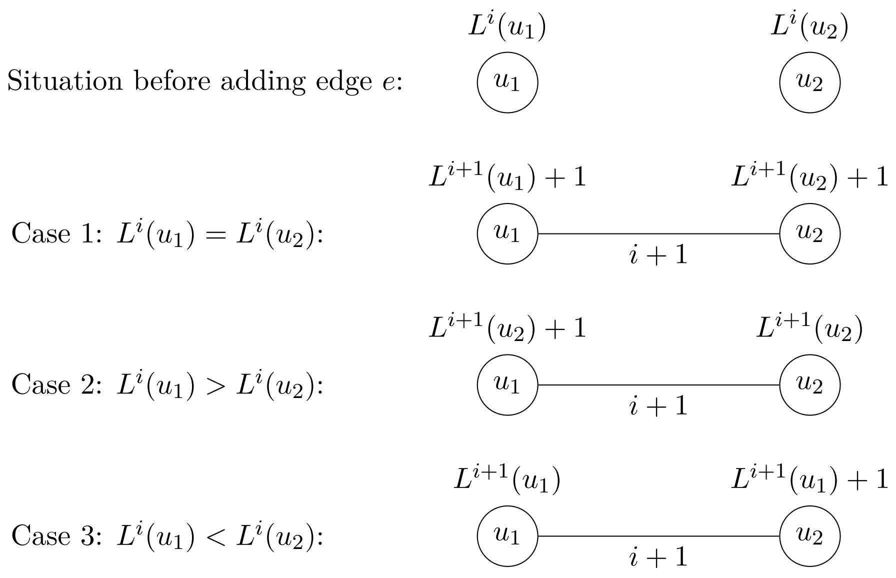


Figure 3.2: Case distinction when adding edge e in Lemma 1

contradiction to Lemma 2 that postulates that the sum of the length of all longest strictly decreasing trails $\sum_{v \in V} L^q(v)$ is greater than $2 \cdot q$. Hence, there has to be at least one vertex with a strictly decreasing trail that is longer than $2 \cdot \lfloor \frac{q}{n} \rfloor$ in G^q. This trail contains a subtrail of length $2 \cdot \lfloor \frac{q}{n} \rfloor$. Since $E^q = E$ it follows that $G^q = G$, which concludes the proof. $\square$

Based on Theorem 1, we get the following results.

Corollary 1. *In an undirected graph G it also holds that $P_i(w, G) \geq 2 \cdot \lfloor \frac{q}{n} \rfloor$ since when reversing a strictly decreasing trail one obtains a strictly increasing one. In this case, define $L^i(v)$ as the length of a longest strictly increasing trail ending at v in G^i.*

Corollary 2. *Let G be as in Theorem 1 and additionally assume that G is complete and undirected. Then, there exists a trail of length at least $n-1$, i.e. $f_d(n) = f_i(n) \geq n-1$.*

3.2.2 Algorithm for Computing Strictly Decreasing Trails

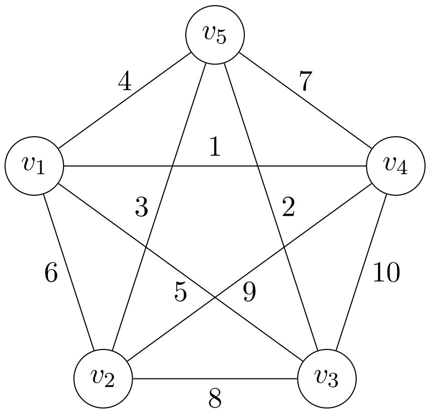
Note that the proof of Lemma 1 is constructive, yielding Algorithm 3.1 for computing longest strictly decreasing trails. Function $findEndpoints$ searches for an edge in a graph G by its weight i and returns both endpoints. Function $findMax$ returns the maximum value of the array L.

Example 2 shows more in detail how the algorithm works.

Example 2. *Consider the complete graph K_6 with an edge-labelling as shown in Figure 3.3. The table shows one run of the algorithm, each entry in the table corresponds to one iteration of the for loop in line 5 in Algorithm 3.3.*

Algorithm 3.1: Find Longest Strictly Decreasing Trail

```
Input: Graph $G = (V, E)$
Output: Length of longest decreasing trail in G
Function findLongestTrail($V, E$):
    for $v \in V$ do
        $L(v) := 0$;
    end
    for $i = 1; i < |E|; i++$ do
        $(u, v) = findEndpoints(i)$;
        $temp = L(u)$;
        $L(u) = \max(L(v) + 1, L(u))$ ;
        $L(v) = \max(temp + 1, L(v))$ ;
    end
    return findMax(L,V);
End Function
```



(a) Example Graph K_6

	$L(v_1)$	$L(v_2)$	$L(v_3)$	$L(v_4)$	$L(v_5)$
0	0	0	0	0	0
1	1	0	0	1	0
2	1	0	1	1	1
3	1	2	1	1	1
4	2	2	1	1	2
5	2	2	3	1	2
6	3	3	3	1	2
7	3	3	3	3	2
8	3	4	4	3	2
9	3	4	3	5	2
10	3	5	6	5	2

(b) Steps of findLongestTrail(K_6)

Figure 3.3: Example of executing Algorithm 3.1

An implementation of this algorithm in C++ can be found online at `https://github.com/Lachnitt/Formalizing-Graph-Trail-Properties/`.

3.3 Upper Bound on the Length of Strictly Decreasing Trails in Complete Graphs

In Section 3.2 we showed that any graph contains a strictly ordered trail of length at least $2 \cdot \lfloor \frac{q}{n} \rfloor$, this means we found a lower bound on the length. In this section an upper bound on the minimum length of strictly ordered trails is proven. Trivially, we can construct weight assignments for complete graphs with n vertices such that trails of length $\binom{n}{2}$ exist. However, this bound is not the tightest bound that can be proven.

For complete graphs Corollary 2 states that a trail of length at least $n-1$ has to exist. But is there always a weight assignment such that all vertices in a graph are starting point for a strictly ordered trail of at most $n-1$? Indeed the minimum length of an strictly decreasing or increasing trail that must exist in the complete graph with n vertices is:

$$f_d(n) = f_i(n) = \begin{cases} n & \text{if } n \in \{3,5\} \\ n-1 & \text{otherwise} \end{cases},$$

that is for complete graphs with $n = 3$ or $n = 5$ vertices there always has to be a trail of length at least n whereas for any other number n of vertices there only has to be a trail of length $n-1$.

The proof that is given in Section 3.2 is constructive. In the formalization of strictly ordered trails in Isabelle/HOL in Section 4.3.3 this property is crucial and the proof is split up in two parts. A first one that shows the correctness of the algorithm and a second one proving that a certain output of the algorithm entails existence of a strictly decreasing trail.

This general idea of formalizing an algorithm in Isabelle and then showing its correctness which also has the benefit of being able to generate a fully verified program could also be useful when the upper bound on the length of strictly decreasing trails should be verified.

Thus, instead of purely reviewing the proofs given in 3.2, this sections rather focusses on showing a modified version of these proofs such that they become constructive. While in the even case the divergence to [GK73] is greater than in the odd case that follows [GK73] more closely, both rely on the same ideas as in [GK73]. The differences will be discussed at the end of each subsection more in detail.

An implementation of the algorithms in C++ can be found can be found online at `https://github.com/Lachnitt/Formalizing-Graph-Trail-Properties/`.

3.3.1 Preliminaries

Definition 3. *Let m denote the number of edges in a complete graph, $m = \frac{n*(n-1)}{2}$.*

Lemma 3. *Let $G = (V, E)$ be a complete edge-weighted graph without a trail of length n. Then, for each vertex $v \in V$ there is a trail of length $n-1$ starting from v.*

Proof. From Corollary 2 have that $\sum_{v \in V} L^m(v) \geq n*(n-1)$. Then, conclude that since there are n vertices in total and no vertex $v \in V$ such that $L^m(v) \geq n$ exists that each vertex $v \in V$ has to satisfy $L^m(v) \geq n-1$. Because there is no trail of length n it holds that $L^m(v) < n$ for all $v \in V$. Thus, there is a trail of length $n-1$ starting from each vertex in the graph. □

Lemma 4. *Let $G = (V, E)$ be a complete edge-weighted graph without a trail of length n. Then, $\sum_{v \in V} L^{i+1}(v) = \sum_{v \in V} L^i(v) + 2$, for all $v \in V$ and $i < m$.*

Proof. From Lemma 1 we know that the sum increases by at least two. Assume towards a contradiction that it increases by more than two in any step. It follows that $\sum_{v \in V} L^m(v) > n * (n - 1)$. But that is a contradiction to Lemma 3. □

Corollary 3. *Let $G = (V, E)$ be a complete edge-weighted graph without a trail of length n. For each step $i \geq 1$ in Algorithm 3.1 there are only two possibilities: Let v and w are endpoints of the edge labelled with i. Either $L^i(v)$ and $L^i(w)$ are both increased by 1 or one of them is not changed while the other is increased by 2.*

Definition 4. *A step i in Algorithm 3.1 that considers an edge (u, v) will be called:*

- *1-1 step if $L^{i+1}(u) = L^i(u) + 1$ and $L^{i+1}(v) = L^i(v) + 1$*
- *2-0 step if $L^{i+1}(u) = L^i(u) + 2$ and $L^{i+1}(v) = L^i(v) + 0$*
- *0-2 step if $L^{i+1}(u) = L^i(u) + 0$ and $L^{i+1}(v) = L^i(v) + 2$*

Theorem 2. *Let $(e_1, ..., e_m), e_i \in E$ be a sequence of distinct edges. Let G be the result of adding the edges in the order of their index in the sequence to an empty graph. Adding the edges uses only 1-1, 2-0 and 0-2 steps and for every vertex $v \in V$ the number of edges added in 2-0 and 0-2 steps is the same if and only if G does not contain strictly ordered trails of length n.*

Proof. ($\Rightarrow$) After adding the edges each vertex v has $n - 1$ outgoing edges. Assume k of them are *1-1* step edge, l are *2-0* step edges. Then, there are also l *0-2* step edges. Thus, $L(v) = 1 * k + 2 * l + 0 * l = n - 1$ for every $v \in V$. Therefore, the resulting graph has no trails of length n.

($\Leftarrow$) Assume the graph does not contains a trail of length n. From Corollary 3 have that only *1-1*, *2-0* and *0-2* steps are used. Assume towards a contradiction that there is at least one vertex $v \in V$ the number of edges added in *2-0* and *0-2* steps is not the same.

Case 1: If there are $k, k > 0$ *2-0* steps more than *0-2* steps, let l denote the number of *0-2* steps and m the number of 1-1 steps. Note that $(k + l) + l + m = n - 1$. Then, $L(v) = 2 * (k + l) + 0 * l + 1 * m = 2k + 2l + m = k + n - 1 > n - 1$ which is a contradiction to the assumption that the graph does not contain a trail of length n.

Case 2: If there are $k, k > 0$ *0-2* steps more than *2-0* steps, let l denote the number of *2-0* steps and m the number of *1-1* steps. Note that $(k + l) + l + m = n - 1$. Then, $L(v) = 0 * (k + l) + 2 * l + 1 * m = 2l + m = n - 1 - k < n - 1$. Have from Lemma 3 that $L(v) = n - 1$. This is a contradiction. □

3.3.2 Graphs with an Even Number of Vertices

In the case that the number n of vertices is even, a graph without strictly ordered trails of length n can be constructed using Algorithm 3.2. From Theorem 2 we know that we have to come up with an order to add edges to an empty graph that only uses *1-1*, *2-0* and *0-2* steps and whenever we perform an *2-0* step on an edge incident to vertex v then we also have to perform a corresponding *0-2* step on another edge incident to that vertex.

Thus, we designed the algorithm that only uses *1-1* steps. The basic idea is that in the beginning of each round all vertices have the same label. Then, pairs of vertices are put together that have not been put together before. An edge is added between them and their labels are increased by one. This is of course only possible because the number of edges is even. Otherwise, we could not only rely on *1-1* steps since one vertex remains without a partner each round.

Algorithm 3.2: Construct an Even Graph without a Trail of Length n

```
    Input: Set of vertices $V$, $|V|$ even
    Output: Complete graph $G = ((V, E), w)$, weight function $w : E \to \{1, .., \frac{n*(n-1)}{2}\}$
Function ConstructGraphEven($V$):
    weight = 1;
    for $v \in V$ do
        L(v) = 0;
    end
    for $i = 0; i < n - 1; i++$ do
        //Finds $n/2$ edges that are not incident
        $(e_1, ..., e_{n/2})$ = findEdges(G);
        for $j = 1; j \leq n/2; j++$ do
            $L(endpoint1(e_j))++$;
            $L(endpoint2(e_j))++$;
            addEdgeWeight(G,$e_i$,j);
        end
    end
    return G;
End Function
```

Example 3. *An execution of the algorithm on the set of vertices* $V = \{v_1, v_2, v_3, v_4\}$*, i.e.* $n = 4$ *is shown in Figure 3.4.*

The longest trails from v_1 *are:* (v_1, v_4, v_2, v_1)

The longest trails from v_2 *are:* (v_2, v_3, v_1, v_2)

The longest trails from v_3 *are:* (v_3, v_2, v_4, v_3)

The longest trails from v_4 *are:* (v_4, v_1, v_3, v_4)

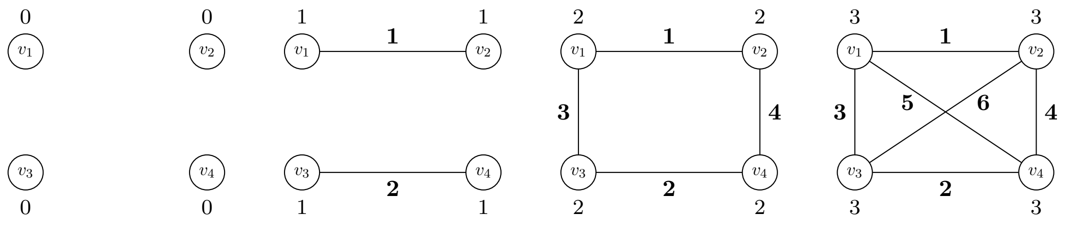


Figure 3.4: Example of executing Algorithm 3.2

Thus, there is no trail of length n in the graph. Notice that this is not the only possible configuration, e.g. 5 and 6 might be switched.

Remark 1. *It is not enough to add the edges in any order. Figure 3.5 gives an example where that would lead to a dead end. In the third graph displayed in the figure no further 1-1 step is possible. Therefore, at the begin of each round Algorithm 3.2 first searches for all n/2 edges to be connected in that round at the beginning of that round, to make sure to avoid to run into such a deadlock.*

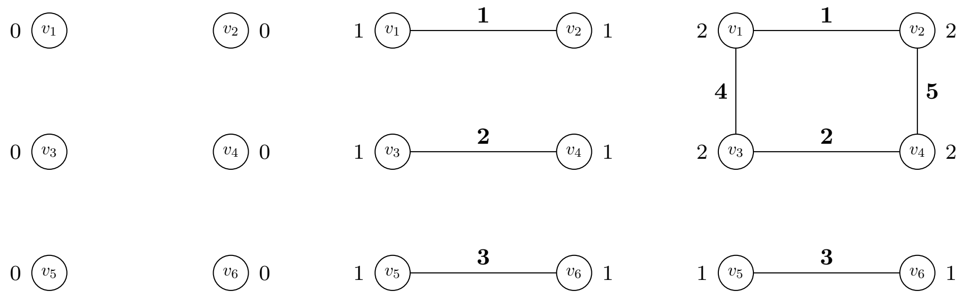


Figure 3.5: Adding edges in the wrong order

Remark 2. *Note that for an implementation it might be beneficial to consider cycles of length n and then split them up in 2 sets of n/2 edges. But in this case n/2 edges might remain in the last step, if n = 2*m and* $2 \nmid m$*.*

Lemma 5. *For every even number n, there is a complete graph* $G = (V, E)$ *with n vertices that does not contain a trail of length n*

Proof. In each round i of Algorithm 3.2 (see line 6) we add n/2 edges to the graph. Edges that are already in the graph are not considered, thus the sequence of edges we generate is distinct. Therefore, after n-1 round the graph is complete. It is always possible to find n/2 edges that are not incident because all vertices always have the same number of incident edges, that is i. In one round the labels of all vertices in the graph are considered as when n/2 non incident edges are chosen n different vertices are endpoints. Therefore, since at the beginning of each round all vertices have the same label, $L(v) = i$ for each

$v \in V$ after a round $L(v) = i + 1$ for each $v \in V$. Thus, only *1-1* steps were made. Then, with Theorem 2 it follows that the constructed graph does not contain a trail of length n.

□

This proof differs from the one in [GK73] in several important points. In [GK73] the authors first show that length of the longest strictly ordered trail is subadditive. Then, they state that every graph with an even number of vertices can be divided into $n - 1$ different matchings (these do correspond to the rounds in Algorithm 3.2). We do not explicitly use the subadditivity here but rather use a constructive variant which then can be verified using a simple induction. This is beneficial with regard to the automation of this proof, which we plan to do for further work (see Chapter 5).

3.3.3 Graphs with an Odd Number of Vertices

There are two special cases, $n = 3$ and $n = 5$ for which every complete graph has a trail of length at least n. For both cases we could just list all possibilities, as it was essentially done by the automated provers in Section 2.1. Another way is to argue about the steps that can be done to add the first five edges to the graph [GK73].

For all odd n such that $n \notin \{3, 5\}$ Algorithm 3.3 constructs witnesses, i.e. graphs with n vertices that do not contain a strictly ordered trail of length n. Intuitively, the algorithm decomposes any graph into two smaller graphs that share a common vertex and thus have an odd number of vertices as well. The base cases are 7, 9 and 11. For these cases special decompositions are hard-coded. They consists of a cycle with an even number of vertices and satellite vertices, an example for one decomposition of a K_7 is given in Figure 3.6. These special decompositions were found by [GK73]. In our Algorithm 3.3 the function *add* adds edges to an edge list and marks them with 0,1,2. If the edge is a satellite, i.e. is only connected to the rest of the decomposition with one endpoint it is marked with 0. If it is part of the cycle it is labelled either 1 or 2 such that the circle is alternately labelled with 1s and 2s.

When two decompositons are merged together, only new satellite vertices are added to a decomposition. Thus, in lines 42, 44, 47, and 49 all edges are marked with 0. To remember which vertex the satellite nodes should be connected to the sets A, B, α and β are used. In the end, the recursive function $findDecomps$ returns $(n - 1)/2$ subgraphs of K_n. The merge process is shown in Example 5 for the graph $n = 7$. The labelling of the edges takes place in *ConstructGraphOdd* using the marks that were done earlier. See Example 4 for an labelling of a decomposition according to its marking in the case of $n = 7$.

Example 4. *Figure 3.6 shows a decomposition of a graph with 7 vertices. The cycle with an even number of vertices is* $(v_1, v_7, v_2, v_5, v_1)$*, the set of satellite vertices* $\{v_3, v_4, v_6\}$*. The edges to the satellite vertices are marked with 0, in the cycle the edges are alternating marked with 1 and 2 (see 3.6a). A weight assignment that could be produced from these marks is shown in 3.6b.*

Algorithm 3.3: Construct an Odd Graph without a Trail of Length n

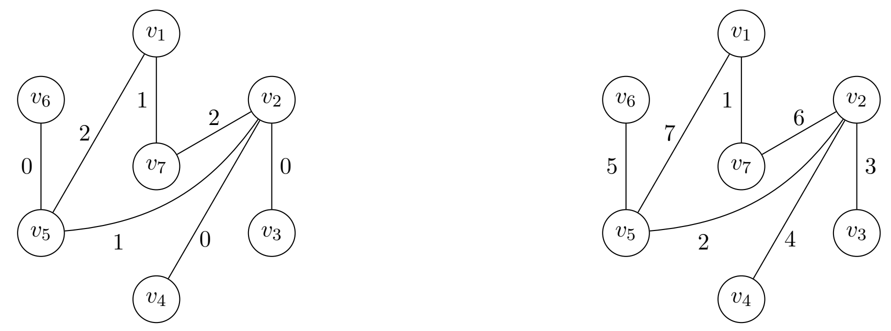
```
Input: Set of vertices $V$, $n = |V|$ odd
Output: Complete graph $G = ((V, E), w)$, weight function $w : E \rightarrow \{1, .., \frac{n*(n-1)}{2}\}$
Function ConstructGraphOdd($V$):
    weight = 1;
    E_List = findDecomps($V, \{\}$);
    G = (V,{});
    for $E$ : $E\_List$ do
        for ($e_1, e_2, mark$) : $E$ do
            if $mark == 1$ then
                G.add($e_1, e_2$,weight);
                weight++;
            end
        end
        for ($e_1, e_2, mark$) : $E$ do
            if $mark == 0$ then
                G.add($e_1, e_2$,weight);
                weight++;
            end
        end
        for ($e_1, e_2, mark$) : $E$ do
            if $mark == 2$ then
                G.add($e_1, e_2$,weight);
                weight++;
            end
        end
    end
    return G;
End Function
```

Example 5. *Figure 3.7 shows an example of merging two special decompositions as in function $findDecomps$ of Algorithm 3.3. It becomes very clear that all new edges that are added lead to satellite edges and do not affect the cycle.*

The proof that Algorithm 3.3 indeed produces a valid weight distribution relies on similar ideas as the proof in the even case presented in Lemma 5. However, the interplay of the different sets has to be considered and is quite involved. A detailed proof is thus omitted since it would go beyond the scope of this thesis However, interested readers might want to consult [GK73] since Algorithm 3.3 is a straight-forward adaption of the inductive proof there.

Algorithm 3.3: Construct an Odd Graph without a Trail of Length n

```
Function findDecomps(V):
    n = |V|;
    if n==7 || n == 9 || n == 11 then
        return hard-coded configuration;
    Find $m, m'$ such that $n = 2*m + 2*m' + 1$;
    $(V_1, V_2)$ = disjointSplit(butlast($V$), $2*m$, $2*m'$);
    $(A_1, B_1, E_1, \alpha_1, \beta_1)$ = findDecomps($V_1$++last($V$));
    $(A_2, B_2, E_2, \alpha_2, \beta_2)$ = findDecomps($V_2$++last($V$));
    $A = A_1 \cup A_2$;
    $B = B_1 \cup B_2$;
    $\alpha = \alpha_1 @ \alpha_2$;
    $\beta = \beta_1 @ \beta_2$;
    $E = E_1 @ E_2$;
    for $int\ i = 0; i < m; i++$ do
        for $v \in A_2$ do
            E[i].add($\alpha_1[i], v, 0$);
        for $v \in B_2$ do
            E[i].add($v, \beta_1[i], 0$);
    for $int\ i = m; i < m + k; i++$ do
        for $v \in B_1$ do
            E[i].add($\alpha_2[i - m], v, 0$);
        for $v \in A_1$ do
            E[i].add($v, \beta_2[i - m], 0$);
    return (A,B,E,$\alpha$,$\beta$);
End Function
```

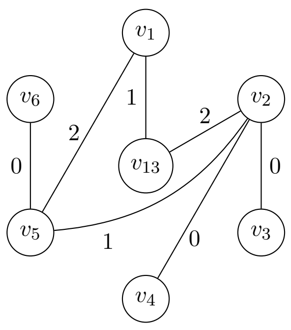


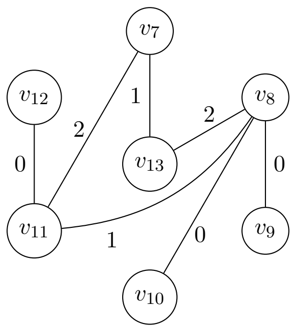
(a) A decomposition of K_7 with marked edges as returned by *findDecomps*

(b) Labelling edges with weights according to their marks as done in *ConstructGraphOdd*

Figure 3.6: Labelling step in Algorithm 3.3

$G_1[0]$:

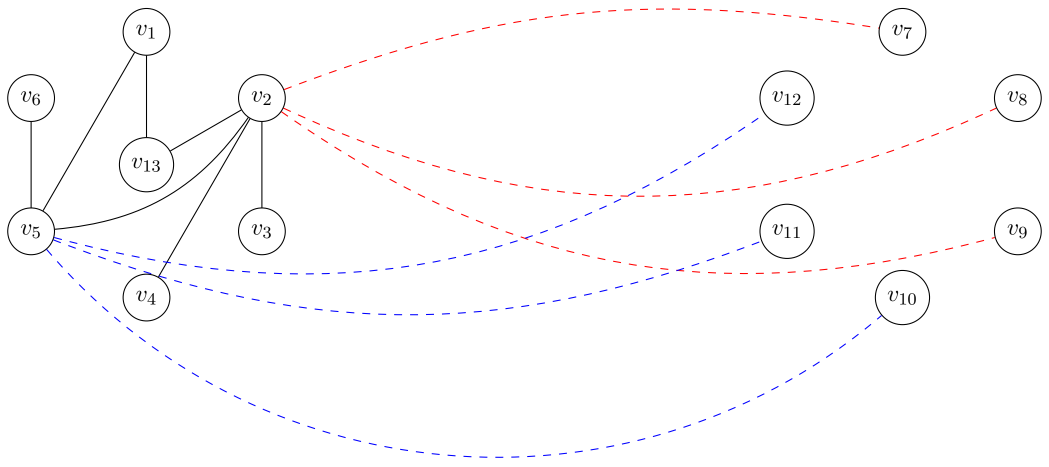




$G_2[0]$:

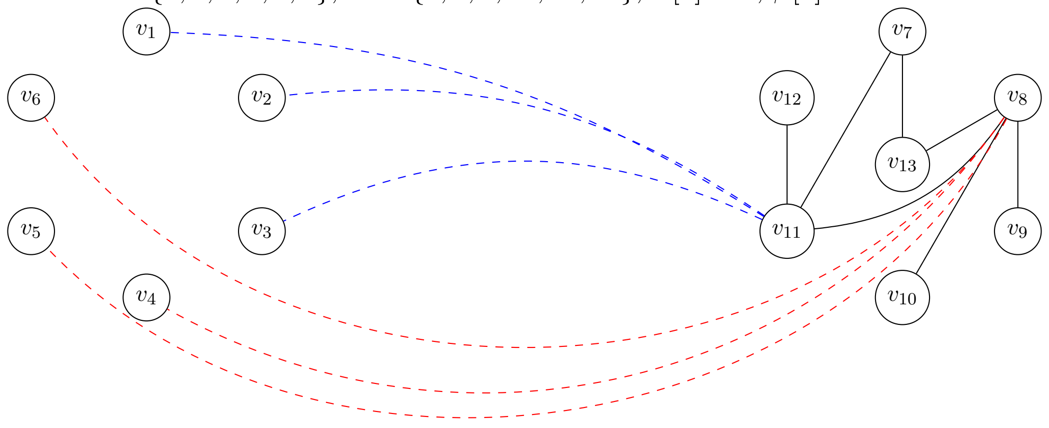
$A_1 = \{1,2,3\}$, $B_1 = \{4,5,6\}$, $\alpha_1[0] = 2$, $\beta_1[0] = 5$

$A_2 = \{7,8,9\}$, $B_2 = \{10,11,12\}$, $\alpha_2[0] = 8$, $\beta_2[0] = 11$



$G[0]$:

$A = \{1,2,3,7,8,9\}$, $B = \{4,5,6,10,11,12\}$, $\alpha[0] = 2$, $\beta[0] = 5$



$G[3]$:

$A = \{1,2,3,7,8,9\}$, $B = \{4,5,6,10,11,12\}$, $\alpha[3] = 9$, $\beta[3] = 11$

Figure 3.7: Merging two decompositions in Algorithm 3.3

CHAPTER 4

Embedding in Isabelle/HOL

4.1 Isabelle/HOL

Isabelle is a generic proof assistant originally developed by Lawrence Paulson in 1986 and is written in Standard ML [Pau93]. Isabelle/HOL is a specialization of Isabelle for Higher-Order Logic (HOL). Its main proof method is resolution based. In contrast to automated theorem provers as for example Vampire [KV13], Isabelle is interactive, i.e. requires human help to find proofs. However, over the Sledgehammer [BBP13] interface, Automatic Theorem Prover (ATP) and Satisfiability Modulo Theories (SMT) solver including Vampire [KV13], CVC4 [BCD$^+$11] and E [Sch02] can be used.

In the official tutorial on Isabelle/HOL [Nip13] the following "equation" is used to explain some of Isabelle's main features:

"HOL = Functional Programming + Logic"

How does this relate to our graph problem? We will use functional programming to formalize the algorithm 3.2.2 and then formalize Theorem 1 using a logical formula. Isabelle's expressiveness allows us to translate all lemmas from Section 3.2 into program code. Throughout the entire document the Intelligible semi-automated reasoning (ISAR) language is used, a more human-readable alternative to the so-called apply scripts. This allows us to also translate the single proof steps from Section 3.2 into ISAR.

4.2 Graph Theory in the Archive of Formal Proofs

To increase the reusability of our work, we build upon the *Graph_Theory* library by Noschinski [Nos13]. Graphs are represented as records consisting of vertices and edges that can be accessed using the selectors *pverts* and *parcs*. We recall the definition of the type *pair_pre_digraph*:

```
record 'a pair_pre_digraph = pverts :: "'a set" parcs :: "'a rel"
```

Now restrictions upon the two sets and new features can be introduced using locales. Locales are Isabelle's way to deal with parameterized theories [Bal10]. Consider for example *pair_wf_digraph*. The endpoints of an edge can be accessed using the functions *fst* and *snd*. Therefore, conditions *arc_fst_in_verts* and *arc_snd_in_verts* assert that both endpoints of an edge are vertices. Using so-called sublocales a variety of other graphs are defined.

```
locale pair_wf_digraph = pair_pre_digraph +
  assumes arc_fst_in_verts: "⋀e. e ∈ parcs G ⟹ fst e ∈ pverts G"
  assumes arc_snd_in_verts: "⋀e. e ∈ parcs G ⟹ snd e ∈ pverts G"
```

An object of type *'b awalk* is defined in *Graph_Theory.Arc_Walk* as a list of edges. Additionally, the definition *awalk* imposes that both endpoints of a walk are vertices of the graph, all elements of the walk are edges and two subsequent edges share a common vertex.

```
type_synonym 'b awalk = "'b list"
```

```
definition awalk :: "'a ⇒ 'b awalk ⇒ 'a ⇒ bool"
"awalk u p v ≡ u ∈ verts G ∧ set p ⊆ arcs G ∧ cas u p v"
```

We also reuse the type synonym *weight_fun* introduced in *Weighted_Graph*.

```
type_synonym 'b weight_fun = "'b ⇒ real"
```

Finally, there is a useful definition capturing the notion of a complete graph, namely *complete_digraph*.

```
definition complete_digraph :: "nat ⇒ ('a,'b) pre_digraph ⇒ bool" ("K_")
where "complete_digraph n G ≡ graph G ∧ card (verts G) = n ∧ arcs_ends
G = {(u,v). (u,v) ∈ verts G × verts G ∧ u ≠ v}"
```

4.3 Formalization of Trail Properties in Isabelle/HOL

4.3.1 Increasing and Decreasing Trails in Weighted Graphs

In our work we extend the graph theory framework from Section 4.2 with new features enabling reasoning about trails. To this end, a trail is defined as a list of edges. We will only consider strictly increasing trails on graphs without parallel edges. For this we require the graph to be of type *pair_pre_digraph*, as introduced in Section 4.2.

Two different definitions are given in our formalization. Function *incTrail* can be used without specifying the first and last vertex of the trail whereas *incTrail2* uses more of *Graph_Theory's* predefined features. Moreover, making use of monotonicity *incTrail* only requires to check if one edge's weight is smaller than its successors' while *incTrail2* checks if the weight is smaller than the one of all subsequent edges in the sequence, i.e. if the list is sorted. The *equivalence between the two notions* is shown in the following.

```
fun incTrail :: "'a pair_pre_digraph ⇒ ('a ×'a) weight_fun ⇒ ('a ×'a)
list ⇒ bool" where
"incTrail g w [] = True" |
"incTrail g w [e₁] = (e₁ ∈ parcs g)" |
"incTrail g w (e₁#e₂#es) =
                (if w e₁ < w e₂ ∧ e₁ ∈ parcs g ∧ snd e₁ = fst e₂
                 then incTrail g w (e₂#es) else False)"

definition (in pair_pre_digraph) incTrail2 where
"incTrail2 w es u v ≡ sorted_wrt (λ e₁ e₂. w e₁ < w e₂) es ∧
                        (es = [] ∨ awalk u es v)"

fun decTrail :: "'a pair_pre_digraph ⇒ ('a ×'a) weight_fun ⇒ ('a ×'a)
list ⇒ bool" where
"decTrail g w [] = True" |
"decTrail g w [e₁] = (e₁ ∈ parcs g)" |
"decTrail g w (e₁#e₂#es) =
                (if w e₁ > w e₂ ∧ e₁ ∈ parcs g ∧ snd e₁ = fst e₂
                 then decTrail g w (e₂#es) else False)"

definition (in pair_pre_digraph) decTrail2 where
"decTrail2 w es u v ≡ sorted_wrt (λ e₁ e₂. w e₁ > w e₂) es ∧
                        (es = [] ∨ awalk u es v)"
```

Defining trails as lists in Isabelle has many advantages including using predefined list operators, e.g., drop. Thus, we can show one result that will be constantly needed in the following, that is, that *any subtrail of an ordered trail is an ordered trail itself.*

```
lemma incTrail_subtrail:
  assumes "incTrail g w es"
  shows "incTrail g w (drop k es)"
```

```
lemma decTrail_subtrail:
  assumes "decTrail g w es"
  shows "decTrail g w (drop k es)"
```

In Isabelle we then show the equivalence between the two definitions *decTrail* and *decTrail2* of strictly decreasing trails. Similarly, we also show the equivalence between the definition *incTrail* and *incTrail2* of strictly increasing trails.

```
lemma (in pair_wf_digraph) decTrail_is_dec_walk:
  shows "decTrail G w es ⟷
          decTrail2 w es (fst (hd es)) (snd (last es))"

lemma (in pair_wf_digraph) incTrail_is_inc_walk:
  shows "incTrail G w es ⟷
          incTrail2 w es (fst (hd es)) (snd (last es))"
```

Any strictly decreasing trail $(e_1, \ldots, e_n)$ can also be seen as a strictly increasing trail $(e_n, ..., e_1)$ if the graph considered is undirected. To this end, we make use of the locale *pair_sym_digraph* that captures the idea of symmetric arcs. However, it is also necessary to assume that the weight function assigns the same weight to edge (v_i, v_j) as to (v_j, v_i). This assumption is therefore added to *decTrail_eq_rev_incTrail* and *incTrail_eq_rev_decTrail*.

```
lemma (in pair_sym_digraph) decTrail_eq_rev_incTrail:
  assumes "∀ v₁ v₂. w (v₁,v₂) = w(v₂,v₁)"
  shows "decTrail G w es ⟷
          incTrail G w (rev (map (λ(v₁,v₂). (v₂,v₁)) es))"

lemma (in pair_sym_digraph) incTrail_eq_rev_decTrail:
  assumes "∀ v₁ v₂. w (v₁,v₂) = w(v₂,v₁)"
  shows "incTrail G w es ⟷
          decTrail G w (rev (map (λ(v₁,v₂). (v₂,v₁)) es))"
```

4.3.2 Weighted Graphs

We add the locale *weighted_pair_graph* on top of the locale *pair_graph* introduced in *Graph_Theory*. A *pair_graph* is a finite, loop free and symmetric graph. We do not restrict the types of vertices and edges but impose the condition that they have to be a linear order.

Furthermore, all weights have to be integers between 0 and $\lfloor \frac{q}{2} \rfloor$ where 0 is used as a special value to indicate that there is no edge at that position. Since the range of the weight function is in the reals, the set of natural numbers *{1,..,card (parcs G) div 2}* has to be casted into a set of reals. This is realized by taking the image of the function *real* that casts a natural number to a real.

```
locale weighted_pair_graph =
pair_graph "(G:: ('a::linorder) pair_pre_digraph)" for G +
  fixes w :: "('a×'a) weight_fun"
  assumes dom: "e ∈ parcs G ⟶ w e ∈ real ` {1..card (parcs G) div 2}"
      and vert_ge: "card (pverts G) ≥ 1"
```

We introduce some useful abbreviations, according to the ones in Section 3.1

```
abbreviation (in weighted_pair_graph) "q ≡ card (parcs G)"
abbreviation (in weighted_pair_graph) "n ≡ card (pverts G)"
abbreviation (in weighted_pair_graph) "W ≡ {1..q div 2}"
```

Note an important difference between Section 3.2 and our formalization. Although a *weighted_pair_graph* is symmetric, the edge set contains both "directions" of an edge, i.e., (v_1, v_2) and (v_2, v_1) are both in *parcs G*. Thus, the maximum number of edges (in the case that the graph is complete) is $n \cdot (n-1)$ and not $\frac{n\cdot(n-1)}{2}$. Another consequence is that the number q of edges is always even.

```
lemma (in weighted_pair_graph) max_arcs:
  shows "card (parcs G) ≤ n*(n-1)"

lemma (in weighted_pair_graph) even_arcs:
  shows "even q"
```

The sublocale *distinct_weighted_pair_graph* refines *weighted_pair_graph*. The condition *zero* fixes the meaning of 0. The weight function is defined on the set of all vertices but since self loops are not allowed; we use 0 as a special value to indicate the unavailability of the edge. The second condition *distinct* enforces that no two edges can have the same weight. There are some exceptions however captured in the statement $(v_1 = u_2 \land v_2 = u_1) \lor (v_1 = u_1 \land v_2 = u_2)$. Firstly, (v_1, v_2) should have the same weight as (v_2, v_1). Secondly, $w(v_1, v_2)$ has the same value as $w(v_1, v_2)$. Note that both edges being self loops resulting in them both having weight 0 is prohibited by condition *zero*. Our decision to separate these two conditions from the ones in *weighted_pair_graph* instead of making one locale of its own is two-fold: On the one hand, there are scenarios where distinctiveness is not wished for. On the other hand, 0 might not be available as a special value.

```
locale distinct_weighted_pair_graph = weighted_pair_graph +
  assumes zero: "∀ v1 v2. (v1,v2) ∉ parcs G ⟷ w (v1,v2) = 0"
      and distinct: "∀ (v1,v2) ∈ parcs G. ∀ (u1,u2) ∈ parcs G.
                     ((v1 = u2 ∧ v2 = u1) ∨ (v1 = u1 ∧ v2 = u2)) ⟷
                     w (v1,v2) = w (u1,u2)"
```

One important step in our formalization is to show that the weight function is surjective. However, having two elements of the domain (edges) being mapped to the same element of the codomain (weight) makes the proof complicated. We therefore first prove that the weight function is surjective on a restricted set of edges. Here we use the fact that there is a linear order on vertices by only considering edges were the first endpoint is bigger than the second.

Then, the surjectivity of w is relatively simple to show. Note that we could also have assumed surjectivity in `distinct_weighted_pair_graph` and shown that distinctiveness follows from it. However, distinctiveness is the more natural assumption that is more likely to appear in any application of ordered trails.

```
lemma (in distinct_weighted_pair_graph) restricted_weight_fun_surjective:
  shows "(∀k ∈ W. ∃(v₁,v₂) ∈ (parcs G). w (v₁,v₂) = k)"

lemma (in distinct_weighted_pair_graph) weight_fun_surjective:
  shows "(∀k ∈ W. ∃(v₁,v₂) ∈ (parcs G). w (v₁,v₂) = k)"
```

4.3.3 Computing a Longest Ordered Trail

We next formally verify Algorithm 3.2.2 and compute longest ordered trails. To this end, we introduce the function `findEdge` to find an edge in a list of edges by its weight.

```
fun findEdge :: "('a×'a) weight_fun ⇒ ('a×'a) list ⇒ real ⇒ ('a×'a)"
where
"findEdge f [] k = undefined" |
"findEdge f (e#es) k = (if f e = k then e else findEdge f es k)"
```

Function `findEdge` will correctly return the edge whose weight is k. We do not care in which order the endpoints are found, i.e. whether (v_1, v_2) or (v_2, v_1) is returned.

```
lemma (in distinct_weighted_pair_graph) findEdge_success:
  assumes "k ∈ W" and "w (v₁,v₂) = k" and "(parcs G) ≠ {}"
  shows "(findEdge w (set_to_list (parcs G)) k) = (v₁,v₂)
       ∨ (findEdge w (set_to_list (parcs G)) k) = (v₂,v₁)"
```

We translate the notion of a labelling function $L^i(v)$ (see Definition 2 of Section 3.2) into Isabelle. Function `getL G w`, in short for get label, returns the length of the longest strictly decreasing path starting at vertex v. In contrast to Definition 2 subgraphs are treated here implicitly. Intuitively, this can be seen as adding edges to an empty graph in order of their weight.

```
fun getL :: "('a::linorder) pair_pre_digraph ⇒ ('a×'a) weight_fun ⇒ nat
⇒ 'a ⇒ nat" where
"getL g w 0 v = 0" |
"getL g w (Suc i) v =
     (let (v₁,v₂) = (findEdge w (set_to_list (arcs g)) (Suc i)) in
     (if v = v₁ then max ((getL g w i v₂)+1) (getL g w i v) else
     (if v = v₂ then max ((getL g w i v₁)+1) (getL g w i v) else
      getL g w i v)))"
```

To add all edges to the graph, set $i = |E|$. Recall that `card (parcs g)` $= 2 * |E|$, as every edge appears twice. Then, iterate over all vertices and give back the maximum length which is found by using `getL G w`. Since `getL G w` can also be used to get a longest strictly increasing trail ending at vertex v the algorithm is not restricted to strictly decreasing trails.

```
definition getLongestTrail :: "('a::linorder) pair_pre_digraph ⇒ ('a×'a)
weight_fun ⇒ nat" where
"getLongestTrail g w =
  Max (set [(getL g w (card (parcs g) div 2) v)
  v .  <- sorted_list_of_set (pverts g)])"
```

Exporting the algorithm into Haskell code results in a fully verified program to find a longest strictly decreasing or strictly increasing trail.

```
export_code getLongestTrail in Haskell module_name LongestTrail
```

Using an induction proof and extensive case distinction, the correctness of Algorithm 3.2.2 is then shown in our formalization, by proving the following theorem:

```
theorem (in distinct_weighted_pair_graph)  correctness:
  assumes "∃ v ∈ (pverts G). getL G w (q div 2) v = k"
  shows "∃ xs. decTrail G w xs ∧ length xs = k"
```

4.3.4 Minimum Length of Ordered Trails

The algorithm introduced in Section 4.3.3 is useful on its own. Additionally, it can also be used to verify the lower bound on the minimum length of a strictly decreasing trail $P_d(w,G) \geq 2 \cdot \lfloor \frac{q}{n} \rfloor$.

To this end, Lemma 1 from Section 3.2 is translated into Isabelle where it is called `minimal_increase_one_step`. The proof is similar to its counterpart, also using a case distinction. Lemma 2 is subsequently proved, here named `minimal_increase_total`.

```
lemma (in distinct_weighted_pair_graph) minimal_increase_one_step:
  assumes "k + 1 ∈ W"
  shows "(∑ v ∈ pverts G. getL G w (k+1) v)
        ≥ (∑ v ∈ pverts G. getL G w k v) + 2"
```

```
lemma (in distinct_weighted_pair_graph) minimal_increase_total:
  shows "(∑ v ∈ pverts G. getL G w (q div 2) v) ≥ q"
```

From `minimal_increase_total` we have that that the sum of all labels after q div 2 steps is greater than q. Now assume that all labels are smaller than q div n. Because we have n vertices, this leads to a contradiction, which proves `algo_result_min`.

```
lemma (in distinct_weighted_pair_graph) algo_result_min:
  shows "(∃ v ∈ pverts G. getL G w (q div 2) v ≥ q div n)"
```

Finally, using lemma `algo_result_min` together with the `correctness` theorem of section 4.3.3, we prove the lower bound of $2 \cdot \lfloor \frac{q}{n} \rfloor$ over the length of a longest strictly decreasing trail. This general approach could also be used to extend our formalization and prove existence of other trails. For example, assume that some restrictions on the graph give raise to the existence of a trail of length $m \geq 2 \cdot \lfloor \frac{q}{n} \rfloor$. Then, it is only necessary to show that our algorithm can find this trail.

```
theorem (in distinct_weighted_pair_graph) dec_trail_exists:
  shows "∃ es. decTrail G w es ∧ length es = q div n"

theorem (in distinct_weighted_pair_graph) inc_trail_exists:
  shows "∃ es. incTrail G w es ∧ length es = q div n"
```

Corollary 2 is translated into *dec_trail_exists_complete*. The proof first argues that the number of edges is $n \cdot (n-1)$ by restricting its domain as done already in Section 4.3.2.

```
lemma (in distinct_weighted_pair_graph) dec_trail_exists_complete:
  assumes "complete_digraph n G"
  shows "(∃ es. decTrail G w es ∧ length es = n-1)"
```

4.3.5 Example Graph K_4

We return to the example graph from Figure 1.1 and show that our results from Sections 4.3.1-4.3.4 can be used to prove existence of trails of length k, in particular $k = 3$ in K_4 using Isabelle. Defining the graph and the weight function separately, we use natural numbers as vertices.

```
abbreviation ExampleGraph:: "nat pair_pre_digraph" where
"ExampleGraph ≡ (|
  pverts =
    {1,2,3,(4::nat)},
  parcs =
    {(v1,v2). v1 ∈ {1,2,3,(4::nat)} ∧ v2 ∈ {1,2,3,(4::nat)} ∧ v1 ≠ v2}
|)"

abbreviation ExampleGraphWeightFunction :: "(nat×nat) weight_fun" where
"ExampleGraphWeightFunction ≡ (λ(v1,v2).
   if (v1 = 1 ∧ v2 = 2) ∨ (v1 = 2 ∧ v2 = 1) then 1 else
  (if (v1 = 1 ∧ v2 = 3) ∨ (v1 = 3 ∧ v2 = 1) then 3 else
  (if (v1 = 1 ∧ v2 = 4) ∨ (v1 = 4 ∧ v2 = 1) then 6 else
  (if (v1 = 2 ∧ v2 = 3) ∨ (v1 = 3 ∧ v2 = 2) then 5 else
  (if (v1 = 2 ∧ v2 = 4) ∨ (v1 = 4 ∧ v2 = 2) then 4 else
  (if (v1 = 3 ∧ v2 = 4) ∨ (v1 = 4 ∧ v2 = 3) then 2 else 0))))))"
```

We show that the graph K_4 of Figure 1.1 satisfies the conditions that were imposed in *distinct_weighted_pair_graph* and its parent locale, including for example no self loops and distinctiveness. Of course there is still some effort required for this. However, it is not necessary to manually construct trails or list all possible weight distributions. Additionally, instead of $q!$ statements there are at most $\frac{3q}{2}$ statements needed.

```
interpretation example:
  distinct_weighted_pair_graph ExampleGraph ExampleGraphWeightFunction
```

Now it is an easy task to prove that there is a trail of length 3. We only add the fact that `ExampleGraph` is a `distinct_weighted_pair_graph` and lemma `dec_trail_exists`.

```
lemma ExampleGraph_decTrail:
  shows "∃ xs. decTrail ExampleGraph ExampleGraphWeightFunction xs
         ∧ length xs = 3"
```

CHAPTER 5

Related Work and Discussion

5.1 Related Work

In 1970 Chvátal and Komlós [CK70] showed a generalisation of Gallai's theorem connecting graph colourings and strictly ordered paths. Thus, they raised the question of the minimum length of strictly increasing paths in complete graphs. Subsequently, Graham and Kletman [GK73] proved that the lower bound of the length of increasing trails is given by $2 \cdot \lfloor \frac{q}{n} \rfloor$. They also gave an upper bound on the minimum length $f(n)$ of increasing trails in complete graphs.

$$f(n) = \begin{cases} n & \text{if } n \in \{3,5\} \\ n-1 & \text{otherwise} \end{cases},$$

Furthermore, they show that in a complete graph with n vertices there has to be an increasing path of length at least $\frac{1}{2}(\sqrt{4n-3}-1)$ and at most $\frac{3n}{4}$. The upper bound was afterwards improved by Calderbank, Chung and Sturtevant [CCS84]. They showed that the maximum length of a path with increasing edge weights is lesser equal $(\frac{1}{2}+o(1))n$. Then, Milans [Mil15] proved that the lower bound can be increased to $(\frac{1}{20}-o(1))(\frac{n}{\log\ n})^{\frac{2}{3}}$. Further progress was made recently by Bucić et al. [BKP$^+$18] who improved the lower bound to $n^{1-o(1)}$.

There has also been effort to show such bounds on graphs with an average degree of d. As in the case of complete graphs these bounds have been improved incrementally. The most recent improvement was made in [BKP$^+$18] where the authors proved that if $d \geq 2$ the lower bound on strictly ordered trails of a graph with average degree d is $\frac{d}{2^{O(\sqrt{\log d \log\log n})}}$. In addition to this, other classes of graphs have been considered, e.g., trees and planar graphs [RSY01], on random edge-ordering [Yus01] or on hypercubes [DSMP$^+$15].

In terms of formalizing strictly ordered trails in edge-weighted graphs, the $Graph-Theory$ library by Noschinski [Nos13], that is part of the Archive of Formal Proofs (AFP) provides many results on graphs. This includes the formalization of weight functions and as well as many different kinds of graphs. Furthermore, it contains definitions of walks, trails and paths as well as a formalization of shortest paths. However, the authors do not formalize strictly ordered trails, nor formalize the special weighted graphs we use (locale *distinct-weighted-pair-graph*). Therefore, our formalization extends [Nos13] with definitions on strictly decreasing and increasing trails and provides many basic lemmas on them. We also add the proof of the minimum length of strictly ordered trails. The main challenges were the reasoning on the surjectivity of the weight function (not present in [Nos13] either) as well the correctness proof of the algorithm.

5.2 Discussion and Further Work

In this thesis we formalized strictly decreasing and increasing trails. We proved an lower bound on the length of strictly ordered trails which exemplary shows how to approach ordered trail problems.

For further work we plan to formalize the upper bound on the minimum length of strictly ordered trails in complete graphs as shown in Section 3.3. Because we already transformed the proof by [GK73] who used decomposition of graphs into cycles or 1-factors into constructive proofs we already have a basis to work on. In particular, note that in Section 2.1 we demonstrated that the two special cases $n = 3$ and $n = 5$ can be shown using an an automated reasoning engine. Similar to Section 4.3.3 we plan to obtain fully verified programs that realise Algorithms 3.2 and 3.3. We believe that formalizing this result would be a valuable extension to the theory `Ordered_Trail`.

One of our goals was to learn lessons from the Isabelle/HOL encoding that can be used to improve the proof finding of the automated provers used in Section 2. In these respect our formalization showed the importance of showing surjectivity of the weight function as well as that it is better to reason about decreasing instead of increasing trails since the algorithm considers edges in ascending order and new edges can better be added to the end of a sequence than to the beginning.

Our formalization can be easily extended and could therefore serve as a basis for further work in this field. The definitions *incTrail* and *decTrail* and the respective properties that are proven in Section 4.3.1 are the key to many other variants of trail properties. Because our theory builds on [Nos13] all notions and proofs introduced there can be combined with our newly added contributions.

One possible direction for further investigation are monotone paths. As mentioned above there is no tight bound but narrowing down that such a path has to have length at least $n^{1-o(1)}$ and at most $(\frac{1}{2} + o(1))n$ would be already useful.

CHAPTER 6
Conclusion

In this work we formalized strictly increasing and strictly decreasing trails in the proof assistant Isabelle/HOL. Furthermore, we showed correctness of an algorithm to find such trails. We provided a fully verified program to compute monotone trails. We used this algorithm to prove the result that every graph with n vertices and q edges has a strictly decreasing trail of length at least $2 \cdot \lfloor \frac{q}{n} \rfloor$.

For further work we plan to show that this is a tight bound for every n except for $n = 3$ and 5. We presented the theoretical foundations for such an undertaking and provide an algorithm to generate graphs with n vertices that do not contain strictly ordered trails of length n. Therefore, we expect that we will be able to obtain a fully verified program in this case, as well. We provide a program to construct such graphs, that can be used as a counterexample generator.

Our results are built on the already existing Isabelle *Graph_theory* from the Archive of Formal Proofs. Thus, our results can be used by any theory using graphs that are specified as in this library. Therefore, our theory is highly reusable and might be the basis for further work in this field.

List of Figures

List of Algorithms

Acronyms

AFP	Archive of Formal Proofs. 4, 36
ATP	Automatic Theorem Prover. 25
AWS	Amazon Web Services. 1
HOL	Higher-Order Logic. 25
ISAR	Intelligible semi-automated reasoning. 25
SMT	Satisfiability Modulo Theories. 25

Bibliography

[Bal10] Clemens Ballarin. Tutorial to locales and locale interpretation. In *Contribuciones científicas en honor de Mirian Andrés Gómez*, pages 123–140. Universidad de La Rioja, 2010.

[BBP13] Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C Paulson. Extending sledgehammer with smt solvers. *Journal of automated reasoning*, 51(1):109–128, 2013.

[BCD$^+$11] Clark Barrett, Christopher L Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. Cvc4. In *International Conference on Computer Aided Verification*, pages 171–177. Springer, 2011.

[BKP$^+$18] Matija Bucic, Matthew Kwan, Alexey Pokrovskiy, Benny Sudakov, Tuan Tran, and Adam Zsolt Wagner. Nearly-linear monotone paths in edge-ordered graphs. *arXiv preprint arXiv:1809.01468*, 2018.

[BST$^+$10] Clark Barrett, Aaron Stump, Cesare Tinelli, et al. The smt-lib standard: Version 2.0. In *Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, England)*, volume 13, page 14, 2010.

[CCS84] A Robert Calderbank, Fan RK Chung, and Dean G Sturtevant. Increasing sequences with nonzero block sums and increasing paths in edge-ordered graphs. *Discrete mathematics*, 50:15–28, 1984.

[CK70] V Chavtal and J Komlos. Some combinatorial theorems on monocity. In *Notices of the American Mathematical Society*, volume 17, page 943. Amer Mathematical Soc 201 Charles St, Providence, RI 02940-2213, 1970.

[CKL19] Byron Cook, Laura Kovács, and Hanna Lachnitt. Personal Communications on Automated Reasoning at AWS, 2019.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[DSMP$^+$15] Jessica De Silva, Theodore Molla, Florian Pfender, Troy Retter, and Michael Tait. Increasing paths in edge-ordered graphs: the hypercube and random graphs. *arXiv preprint arXiv:1502.03146*, 2015.

[Fou18] Free Software Foundation. Gnu time. `https://www.gnu.org/software/time/`, 2018. Accessed: 2020-04-09.

[GK73] RL Graham and DJ Kleitman. Increasing paths in edge ordered graphs. *Periodica Mathematica Hungarica*, 3(1-2):141–148, 1973.

[KV13] Laura Kovács and Andrei Voronkov. First-order theorem proving and vampire. In *Proc. of CAV*, pages 1–35, 2013.

[Mil15] Kevin G. Milans. Monotone paths in dense edge-ordered graphs, 2015.

[Nip13] Tobias Nipkow. Programming and proving in isabelle/hol, 2013.

[Nos13] Lars Noschinski. Graph theory. *Archive of Formal Proofs*, April 2013. `http://isa-afp.org/entries/Graph_Theory.html`, Formal proof development.

[Pau93] Lawrence C. Paulson. Natural deduction as higher-order resolution. *CoRR*, cs.LO/9301104, 1993.

[RSY01] Yehuda Roditty, Barack Shoham, and Raphael Yuster. Monotone paths in edge-ordered sparse graphs. *Discrete Mathematics*, 226(1-3):411–417, 2001.

[Sch02] Stephan Schulz. E–a brainiac theorem prover. *Ai Communications*, 15(2, 3):111–126, 2002.

[Yus01] Raphael Yuster. Large monotone paths in graphs with bounded degree. *Graphs and Combinatorics*, 17(3):579–587, 2001.