Technical Report AC-TR-20-012

September 2020

Formalizing Graph Trail Properties in Isabelle/HOL

Laura Kovács, Hanna Lachnitt, and Stefan Szeider

This is the authors' copy of a paper that will appear in the proceedings of CICM 2020, the 13th International Conference on Intelligent Computer Mathematics.
DOI 10.1007/978-3-030-53518-6_12
www.ac.tuwien.ac.at/tr

Formalizing Graph Trail Properties in Isabelle/HOL

Laura Kovács, Hanna Lachnitt, and Stefan Szeider

TU Wien, Vienna, Austria
{laura.kovacs,hanna.lachnitt,stefan.szeider}@tuwien.ac.at

Abstract. We describe a dataset expressing and proving properties of graph trails, using Isabelle/HOL. We formalize the reasoning about strictly increasing and decreasing trails, using weights over edges, and prove lower bounds over the length of trails in weighted graphs. 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 any decreasing trail is also an increasing one.

Keywords: weighted graph · increasing/decreasing trails · Isabelle/HOL · verified theory formalization

1 Introduction

The problem of finding a longest trail with strictly increasing or strictly decreasing weights in an edge-weighted graph is an interesting graph theoretic problem [8,3,14,7], with potential applications to scheduling and cost distribution in traffic planning and routing [5]. In this paper, we formalize and automate the reasoning about strictly increasing and strictly decreasing trail properties by developing an extendable flexible library in the proof assistant Isabelle/HOL [11].

As a motivating example consider the following (undirected) graph K_4, where each edge is annotated with a different integer-valued weight ranging from $1, \ldots, 6$:

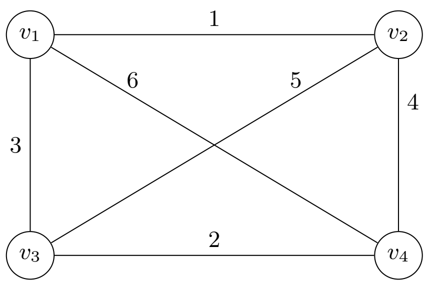


Fig. 1. Example graph K_4

When considering K_4, the question we address in this paper is whether K_4 has a strictly decreasing trail of length $k \geq 1$. A trail is a sequence of distinct edges $(e_1, \ldots, e_k)$, $e_i \in E$ such that there exists a corresponding sequence of vertices $(v_0, ..., v_k)$ where $e_i = v_{i-1}v_i$. A strictly-ordered trail is a trail where the edge weights of $(e_1, \ldots, e_k)$ are either strictly increasing or strictly decreasing. Our work provides a formally verified algorithm computing such strictly-ordered trails. Note that 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. A natural question to ask, which we address in this paper, is whether it is possible to construct a graph such that the constructed graph has 4 vertices and 5 edges, and no vertex is the starting node of a trail of length 3? We answer this question negatively, in an even more general setting, not restricted to 4 vertices and 5 edges. Similarly to the theoretical results of [8], we show that, given a graph G with n vertices and q edges, there is always a strictly decreasing trail of length at least $2 \cdot \lfloor \frac{q}{n} \rfloor$. While such a graph theoretical result has already been announced [8], in this paper we formalize the results in Isabelle/HOL and construct a Isabelle/HOL-verified algorithm computing strictly decreasing trails of length k, whenever such trails exist.

Let us note that proving that a graph G with n vertices and q edges has/does not have decreasing trails is possible for small n, using automated reasoning engines such as Vampire [9] and Z3 [6]. One can restrict the weights to the integers $1, .., q$ and since $q \leq \binom{n}{2}$ there is a finite number of possibilities for each n. Nevertheless, the limit of such an undertaking is reached soon. On our machine[1] even for $n = 7$, both Vampire and Z3 fail proving the existence of strictly decreasing trails, using a 1 hour time limit. This is due to the fact that every combination of edge weights and starting nodes is tested to be a solution. Thus, the provers are not able to contribute to the process of finding an effective proof of the statement. Even for relatively small numbers n, our experiments show that state-of-the-art automated provers are not able to prove whether weighted graphs have a strictly decreasing trail of a certain length.

We also note that this limitation 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 it requires reasoning about $3! = 6$ possibilities for a distribution of a weight function w and then manually constructing concrete trails:

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

Based on such limitations of automative and interactive provers, in this paper 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. To this end, we build upon

[1] standard laptop with 1.7 GHz Dual-Core Intel Core i5 and 8 GB 1600 MHz memory

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 [4]. Subsequently, Graham and Kletman [8] proved that the lower bound of the length of increasing trails is given by $2 \cdot \lfloor \frac{q}{n} \rfloor$, as also mentioned above. In our work, we formalize and verify such results in Isabelle/HOL. Yet, our work is not a straightforward adaptation and formalization of Graham and Kletman's proof [8]. Rather, we focus on decreasing trails instead of increasing trails and give an algorithm computing longest decreasing trails of a given graph (Algorithm 1). By formalizing Algorithm 1 in Isabelle/HOL, we also formally verify the correctness of the trails computed by our approach. Moreover, we prove that any strictly decreasing trail is also an strictly increasing one, allowing this way to use our formalization in Isabelle/HOL also to formalize results of Graham and Kletman [8].

Contributions. This paper brings the following contributions.

(1) We formalize strictly increasing trails and provide basic lemmas about their properties. We improve results of [8] by giving a precise bound on the increase of trail length.

(2) We formalize strictly decreasing trails, in addition to the increasing trail setting of [8]. We prove the duality between strictly increasing and strictly decreasing trails, that is, any such decreasing trail is an increasing one, and vice versa. Thanks to these extensions, unlike [8], we give a constructive proof of the existence of strictly ordered trails (Lemma 1).

(3) We design an algorithm computing longest ordered trails (Algorithm 1), and formally verify its correctness in Isabelle/HOL. We extract our algorithm to 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.

(4) We verify the lower bound on the minimum length of strictly decreasing trails of arbitrary graphs, and of complete graphs in particular.

(5) We build upon the Graph-Theory library by Noschinski [12], that is part of the Archive of Formal Proofs (AFP) and already includes many results on walks and general properties of graphs. We introduce the digital dataset v formalizing properties of graph trails. Our dataset consists of ∼2000 lines of Isabelle code and it took about one month for one person to finish. As far as we know this is the first formalization of ordered trails in a proof assistant.

This paper 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/Ordered_Trail. The rest of the paper is organized as follows. Section 2 recalls basic terminology and properties from graph theory. We prove lower bounds on strictly increasing/decreasing trails in Section 3. We describe our Isabelle/HOL formalization in Isabelle/HOL in Section 4. We discuss further directions in Section 5 and conclude our paper with Section 6.

2 Preliminaries

We briefly recapitulate the basic notions of 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 a weight function. A vertex labelling is a function $L : V \to \mathbb{N}$.

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 there exists a corresponding sequence of vertices $(v_0, ..., v_k)$ where $e_i = v_{i-1}v_i$. 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 $w(e_i) < w(e_{i+1})$. A trail is *strictly-ordered* if it is strictly increasing or strictly decreasing.

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 4.2.

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

3 Lower Bounds on Increasing and Decreasing Trails in Weighted Graphs

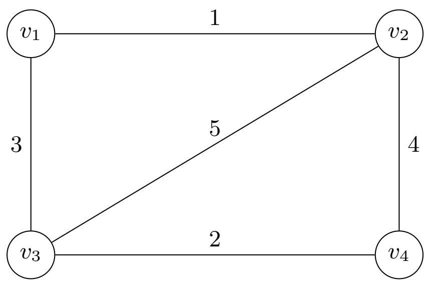
The proof introduced in the following is based on similar ideas as in [8]. However, we diverge from [8] in several aspects. Firstly, we consider strictly decreasing instead of strictly increasing trails, reducing the complexity of the automated proof (see Section 4). Moreover, we add tighter bounds than necessary to give a fully constructive proof in terms of an algorithm for computing the length of these trails (see Section 4.3). We discuss this further at the end of the section.

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* $w : E \to \{1, \ldots, q\}$ *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* $G^i = (V, E^i)$, $n = |V|$ *we define* $L^i : V\{1, \ldots, \frac{n(n-1)}{2}\}$ *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.

In Figure 2 the example graph from Figure 1 is revisited to illustrate these definitions. We need to prove the following property.



Decreasing trails from v_3 are:
$v_3 - v_4$,
$v_3 - v_1 - v_2$,
$v_3 - v_2 - v_1$,
$v_3 - v_2 - v_4 - v3$
Therefore, $L^5(v_3) = 3$.

Decreasing trails from v_1 are:
$v_1 - v_2$
$v_1 - v_3 - v_4$
Therefore, $L^5(v_1) = 2$.

Fig. 2. Graph G^5 with labelling function L^5

Lemma 1. *If $i < q$, then $\sum_{v \in V} L^{i+1}(v) \geq \sum_{v \in V} L^i(v) + 2$.*

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 for an illustration.

Note that $L^{i+1}(v) = L^i(v)$ for $v \in V \setminus \{u_1, u_2\}$, 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,

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

□

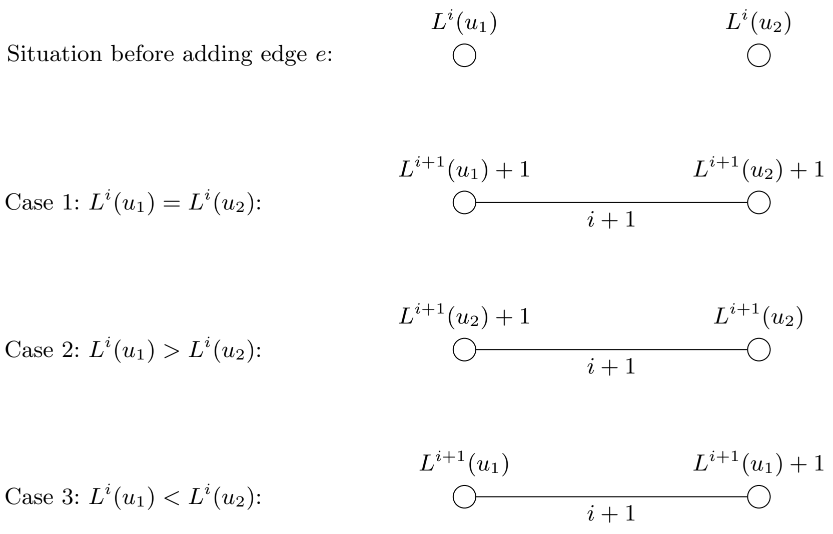


Fig. 3. Case distinction when adding edge e in Lemma 1

Note that the proof of Lemma 1 is constructive, yielding the Algorithm 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.

Algorithm 1: Find Longest Strictly Decreasing Trail

```
for $v \in V$ do
    $L(v) := 0$
end
for $i = 1; i < |E|; i++$ do
    $(u, v) = findEndpoints(G, i)$;
    $temp = L(u)$;
    $L(u) = \max(L(v) + 1, L(u))$ ;
    $L(v) = \max(temp + 1, L(v))$ ;
end
return findMax(L);
```

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

Proof. We proceed 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 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. □

Based on Theorem 1, we get the following results.

Corollary 1. *It 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. Then, there exists a trail of length at least $n - 1$, i.e., $f_i(n) = f_d(n) \geq n - 1$.* □

In [8] the authors present a non-constructive proof. As in Lemma 1 they argue that the sum of the lengths of all increasing trails is at least 2. Thus, they overestimate the increase. We however, use the exact increase therefore making the proof constructive and obtaining Algorithm 1.

4 Formalization of Trail Properties in Isabelle/HOL

4.1 Graph Theory in the Archive of Formal Proofs

To increase the reusability of our library we build upon the *Graph-Theory* library by Noschinski [12]. 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 [1]. 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: $\bigwedge e.\ e \in \mathit{parcs}\ G \Longrightarrow \mathit{fst}\ e \in \mathit{pverts}\ G$
  assumes arc-snd-in-verts: $\bigwedge e.\ e \in \mathit{parcs}\ G \Longrightarrow \mathit{snd}\ e \in \mathit{pverts}\ G$

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 $::\ 'a \Rightarrow\ 'b\ \mathit{awalk} \Rightarrow\ 'a \Rightarrow \mathit{bool}$
$\mathit{awalk}\ u\ p\ v \equiv u \in \mathit{verts}\ G \wedge \mathit{set}\ p \subseteq \mathit{arcs}\ G \wedge \mathit{cas}\ u\ p\ v$

We also reuse the type synonym *weight-fun* introduced in *Weighted-Graph*.

type-synonym $'b$ *weight-fun* $=$ $'b \Rightarrow \mathit{real}$

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

4.2 Increasing and Decreasing Trails in Weighted Graphs

In our work we extend the graph theory framework from Section 4.1 with new features enabling reasoning about ordered 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 4.1.

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* $::\ 'a\ \mathit{pair\text{-}pre\text{-}digraph} \Rightarrow ('a \times 'a)\ \mathit{weight\text{-}fun} \Rightarrow ('a \times 'a)\ \mathit{list} \Rightarrow \mathit{bool}$
where
$\mathit{incTrail}\ g\ w\ [] = \mathit{True}\ |$
$\mathit{incTrail}\ g\ w\ [e_1] = (e_1 \in \mathit{parcs}\ g)\ |$
$\mathit{incTrail}\ g\ w\ (e_1\#e_2\#es) = (\mathit{if}\ w\ e_1 < w\ e_2 \wedge e_1 \in \mathit{parcs}\ g \wedge \mathit{snd}\ e_1 = \mathit{fst}\ e_2$
$\qquad \mathit{then}\ \mathit{incTrail}\ g\ w\ (e_2\#es)\ \mathit{else}\ \mathit{False})$

definition(in *pair-pre-digraph*) *incTrail2* **where**
$\mathit{incTrail2}\ w\ es\ u\ v \equiv \mathit{sorted\text{-}wrt}\ (\lambda\ e_1\ e_2.\ w\ e_1 < w\ e_2)\ es \wedge (es = [] \vee \mathit{awalk}\ u\ es\ v)$

fun *decTrail* $::\ 'a\ \mathit{pair\text{-}pre\text{-}digraph} \Rightarrow ('a \times 'a)\ \mathit{weight\text{-}fun} \Rightarrow ('a \times 'a)\ \mathit{list} \Rightarrow \mathit{bool}$
where

decTrail g w [] = *True* |
decTrail g w [e_1] = (e_1 ∈ *parcs g*) |
decTrail g w (e_1#e_2#*es*) = (*if w* e_1 > *w* e_2 ∧ e_1 ∈ *parcs g* ∧ *snd* e_1 = *fst* e_2
    then decTrail g w (e_2#*es*) *else False*)

definition(**in** *pair-pre-digraph*) *decTrail2* **where**
decTrail2 w es u v ≡ *sorted-wrt* (λ e_1 e_2. *w* e_1 > *w* e_2) *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 ∀ v_1 v_2. *w* (v_1,v_2) = *w*(v_2,v_1)
  shows *decTrail G w es* ⟷ *incTrail G w* (*rev* (*map* (λ(v_1,v_2). (v_2,v_1)) *es*))

lemma(**in** *pair-sym-digraph*) *incTrail-eq-rev-decTrail*:
  assumes ∀ v_1 v_2. *w* (v_1,v_2) = *w*(v_2,v_1)
  shows *incTrail G w es* ⟷ *decTrail G w* (*rev* (*map* (λ(v_1,v_2). (v_2,v_1)) *es*))

4.3 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*: $e \in parcs\ G \longrightarrow w\ e \in real$ ' $\{1..card\ (parcs\ G)\ div\ 2\}$
    and *vert-ge*: $card\ (pverts\ G) \geq 1$

We introduce some useful abbreviations, according to the ones in Section 2

abbreviation(**in** *weighted-pair-graph*) $q \equiv card\ (parcs\ G)$
abbreviation(**in** *weighted-pair-graph*) $n \equiv card\ (pverts\ G)$
abbreviation(**in** *weighted-pair-graph*) $W \equiv \{1..q\ div\ 2\}$

Note an important difference between Section 3 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 $card\ (parcs\ G) \leq n*(n-1)$

lemma (**in** *weighted-pair-graph*) *even-arcs*:
  shows *even q*

The below 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 $(v_1 = u_2 \wedge v_2 = u_1) \vee (v_1 = u_1 \wedge v_2 = u_2)$. 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*: $\forall\ v_1\ v_2.\ (v_1,v_2) \notin parcs\ G \longleftrightarrow w\ (v_1,v_2) = 0$
    and *distinct*: $\forall\ (v_1,v_2) \in parcs\ G.\ \forall\ (u_1,u_2) \in parcs\ G.$
    $((v_1 = u_2 \wedge v_2 = u_1) \vee (v_1 = u_1 \wedge v_2 = u_2)) \longleftrightarrow w\ (v_1,v_2) = w\ (u_1,u_2)$

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*:
  $\forall k \in W. \exists (v_1,v_2) \in \{(p1,p2).\ (p1,p2) \in parcs\ G \wedge p2 < p1\}.\ w\ (v_1,v_2) = k$

lemma(**in** *distinct-weighted-pair-graph*) *weight-fun-surjective*:
  shows $\forall k \in W. \exists (v_1,v_2) \in parcs\ G.\ w\ (v_1,v_2) = k$

4.4 Computing a Longest Ordered Trail

We next formally verify Algorithm 1 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* :: $('a\times'a)$ *weight-fun* $\Rightarrow$ $('a\times'a)$ *list* $\Rightarrow$ *real* $\Rightarrow$ $('a\times'a)$ **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 $k \in W$ **and** $w\ (v_1,v_2) = k$ **and** $(parcs\ G) \neq \{\}$
  shows (*findEdge w* (*set-to-list* (*parcs G*)) k) = (v_1,v_2)
      $\vee$ (*findEdge w* (*set-to-list* (*parcs G*)) k) = (v_2,v_1)

We translate the notion of a labelling function $L^i(v)$ (see Definition 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* $\Rightarrow$ $('a\times'a)$ *weight-fun*
      $\Rightarrow$ *nat* $\Rightarrow$ $'a$ $\Rightarrow$ *nat* **where**
getL g w 0 v = *0* |
getL g w (*Suc i*) v = (*let* (v_1,v_2) = (*findEdge w* (*set-to-list* (*arcs g*)) (*Suc i*)) *in*
    (*if* $v = v_1$ *then max* ((*getL g w i* v_2)+*1*) (*getL g w i v*) *else*
    (*if* $v = v_2$ *then max* ((*getL g w i* v_1)+*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 1 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
```

4.5 Minimum Length of Ordered Trails

The algorithm introduced in Section 4.4 is already useful on its own. Additionally, it can 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 is translated into Isabelle as the lemma *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 4.4, 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 1 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.

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*

4.6 Example Graph K_4

We return to the example graph from Figure 1 and show that our results from Sections 4.2-4.5 can be used to prove existence of trails of length k, in particular $k = 3$ in K_4. 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 = {(v_1,v_2). v_1 ∈ {*1*,*2*,*3*,(*4*::*nat*)} ∧ v_2 ∈ {*1*,*2*,*3*,(*4*::*nat*)} ∧ $v_1 \neq v_2$}
|)

abbreviation *ExampleGraphWeightFunction* :: (*nat*×*nat*) *weight-fun* **where**
ExampleGraphWeightFunction ≡ (λ(v_1,v_2).
  (*if* (v_1 = *1* ∧ v_2 = *2*) ∨ (v_1 = *2* ∧ v_2 = *1*) *then 1 else*
  (*if* (v_1 = *1* ∧ v_2 = *3*) ∨ (v_1 = *3* ∧ v_2 = *1*) *then 3 else*
  (*if* (v_1 = *1* ∧ v_2 = *4*) ∨ (v_1 = *4* ∧ v_2 = *1*) *then 6 else*
  (*if* (v_1 = *2* ∧ v_2 = *3*) ∨ (v_1 = *3* ∧ v_2 = *2*) *then 5 else*
  (*if* (v_1 = *2* ∧ v_2 = *4*) ∨ (v_1 = *4* ∧ v_2 = *2*) *then 4 else*
  (*if* (v_1 = *3* ∧ v_2 = *4*) ∨ (v_1 = *4* ∧ v_2 = *3*) *then 2 else 0*)))))))

We show that the graph K_4 of Figure 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 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*:
  ∃ *xs*. *decTrail ExampleGraph ExampleGraphWeightFunction xs* ∧ *length xs* = *3*

5 Discussion and Related Work

Our theory *Ordered-Trail* builds on top of the *Graph-Theory* library presented in [12]. However, this library does not formalize strictly ordered trails, nor the special weighted graphs we introduced in the locale *distinct−weighted−pair−graph*. Furthermore, our formalization extends [12] with definitions on strictly decreasing and increasing trails and provides many basic lemmas on them. Some of the main challenges in this context were the reasoning on the surjectivity of the weight function as well the correctness proof of the algorithm.

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 4.2 are the key to many other variants of trail properties.

Graham et al. [8] also showed upper bounds for trails in complete graphs by decomposing them into either into cycles or 1-factors. We are currently working on formalizing and certifying the result that

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

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 $n-1$. Therefore, the lower bound that we showed in this paper is equal to the exact length with exception of two special cases. We believe that formalizing this result would be a valuable extension to the theory *Ordered-Trail*.

Another direction for further investigation are monotone paths. Graham et al. [8] 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 [3], Milans [10] and Bucić et al. [2].

Recently, other classes of graphs have been considered, e.g., trees and planar graphs [13], on random edge-ordering [14] or on hypercubes [7].

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 verified algorithm and 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.

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.

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.

References

1. Ballarin, C.: Tutorial to locales and locale interpretation. In: Contribuciones científicas en honor de Mirian Andrés Gómez. pp. 123–140. Universidad de La Rioja (2010)
2. Bucic, M., Kwan, M., Pokrovskiy, A., Sudakov, B., Tran, T., Wagner, A.Z.: Nearly-linear monotone paths in edge-ordered graphs. arXiv preprint arXiv:1809.01468 (2018)
3. Calderbank, A.R., Chung, F.R., Sturtevant, D.G.: Increasing sequences with nonzero block sums and increasing paths in edge-ordered graphs. Discrete Mathematics **50**, 15–28 (1984)
4. Chavtal, V., Komlos, J.: Some combinatorial theorems on monocity. In: Notices of the American Mathematical Society. vol. 17, p. 943. Amer Mathematical Soc 201 Charles St, Providence, RI 02940-2213 (1970)
5. Cook, B., Kovács, L., Lachnitt, H.: Personal Communications on Automated Reasoning at AWS (2019)
6. De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: International conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer (2008)
7. De Silva, J., Molla, T., Pfender, F., Retter, T., Tait, M.: Increasing paths in edge-ordered graphs: the hypercube and random graphs. arXiv preprint arXiv:1502.03146 (2015)
8. Graham, R., Kleitman, D.: Increasing paths in edge ordered graphs. Periodica Mathematica Hungarica **3**(1-2), 141–148 (1973)
9. Kovács, L., Voronkov, A.: First-order theorem proving and Vampire. In: Proc. of CAV. pp. 1–35 (2013)
10. Milans, K.G.: Monotone paths in dense edge-ordered graphs (2015)
11. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: a proof assistant for higher-order logic, vol. 2283. Springer Science & Business Media (2002)
12. Noschinski, L.: Graph theory. Archive of Formal Proofs (Apr 2013), http://isa-afp.org/entries/Graph_Theory.html, Formal proof development
13. Roditty, Y., Shoham, B., Yuster, R.: Monotone paths in edge-ordered sparse graphs. Discrete Mathematics **226**(1-3), 411–417 (2001)
14. Yuster, R.: Large monotone paths in graphs with bounded degree. Graphs and Combinatorics **17**(3), 579–587 (2001)