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 Graph Theory in the Archive of Formal Proofs

To increase the reusability of our library we build upon the *Graph-Theory* library by Noschinski [10]. 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: $\bigwedge e.\ e \in parcs\ G \Longrightarrow fst\ e \in pverts\ G$
assumes arc-snd-in-verts: $\bigwedge e.\ e \in parcs\ G \Longrightarrow snd\ e \in pverts\ G$

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

2 Formalization of Trail Properties in Isabelle/HOL

2.1 Increasing and Decreasing Trails in Weighted Graphs

In our work we extend the graph theory framework from Section 1 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 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 pair-pre-digraph* ⇒ (*'a* ×*'a*) *weight-fun* ⇒ (*'a* ×*'a*) *list* ⇒ *bool*
where
incTrail g w [] = *True* |
incTrail g w [e_1] = (e_1 ∈ *parcs g*) |
incTrail 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 incTrail g w (e_2#*es*) *else False*)

definition(**in** *pair-pre-digraph*) *incTrail2* **where**
incTrail2 w es u v ≡ *sorted-wrt* (λ e_1 e_2. *w* e_1 < *w* e_2) *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 [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*))

2.2 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 $\lfloor \frac{g}{2} \rfloor$ 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 **??**

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 **??** 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 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*: ∀ *v*₁ *v*₂. (*v*₁,*v*₂) ∉ *parcs G* ⟷ *w* (*v*₁,*v*₂) = *0*
and *distinct*: ∀ (*v*₁,*v*₂) ∈ *parcs G*. ∀ (*u1*,*u2*) ∈ *parcs G*.
((*v*₁ = *u2* ∧ *v*₂ = *u1*) ∨ (*v*₁ = *u1* ∧ *v*₂ = *u2*)) ⟷ *w* (*v*₁,*v*₂) = *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 $(\forall\, k \in W.\ \exists\, (v_1,v_2) \in (parcs\ G).\ w\ (v_1,v_2) = k)$

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

2.3 Computing a Longest Ordered Trail

We next formally verify Algorithm **??** 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 $(findEdge\ w\ (set\text{-}to\text{-}list\ (parcs\ G))\ k) = (v_1,v_2)$
$\quad \vee\ (findEdge\ w\ (set\text{-}to\text{-}list\ (parcs\ G))\ k) = (v_2,v_1)$

We translate the notion of a labelling function $L^i(v)$ (see Definition **??**) 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 **??** 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* $\Rightarrow$ $('a \times 'a)$ *weight-fun* $\Rightarrow$ *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 **??** is then shown in our formalization, by proving the following theorem:

theorem(**in** *distinct-weighted-pair-graph*) *correctness*:
assumes $\exists\ v \in (pverts\ G).\ getL\ G\ w\ (q\ div\ 2)\ v = k$
shows $\exists\ xs.\ decTrail\ G\ w\ xs \wedge length\ xs = k$

2.4 Minimum Length of Ordered Trails

The algorithm introduced in Section 2.3 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 **??** 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 $k + 1 \in W$
shows
$(\sum\ v \in pverts\ G.\ getL\ G\ w\ (k+1)\ v) \geq (\sum\ v \in pverts\ G.\ getL\ G\ w\ k\ v) + 2$

lemma(**in** *distinct-weighted-pair-graph*) *minimal-increase-total*:
shows $(\sum\ v \in pverts\ G.\ getL\ G\ w\ (q\ div\ 2)\ v) \geq q$

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 $(\exists\ v \in pverts\ G.\ getL\ G\ w\ (q\ div\ 2)\ v \geq q\ div\ n)$

Finally, using lemma *algo-result-min* together with the *correctness* theorem of section 2.3, 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 $\exists\ es.\ decTrail\ G\ w\ es \wedge length\ es = q\ div\ n$

theorem(**in** *distinct-weighted-pair-graph*) *inc-trail-exists*:
shows $\exists\ es.\ incTrail\ G\ w\ es \wedge length\ es = q\ div\ n$

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 2.2.

lemma(**in** *distinct-weighted-pair-graph*) *dec-trail-exists-complete*:
assumes *complete-digraph n G*
shows ($\exists$ *es*. *decTrail G w es* $\wedge$ *length es* = *n*−*1*)

2.5 Example Graph K_4

We return to the example graph from Figure **??** and show that our results from Sections 2.1-2.4 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 **??** 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*:
$\exists$ *xs*. *decTrail ExampleGraph ExampleGraphWeightFunction xs* ∧ *length xs* = *3*

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. 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)
3. 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)
4. Cook, B., Kovács, L., Lachnitt, H.: Personal Communications on Automated Reasoning at AWS (2019)
5. 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)
6. 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)
7. Graham, R., Kleitman, D.: Increasing paths in edge ordered graphs. Periodica Mathematica Hungarica **3**(1-2), 141–148 (1973)
8. Kovács, L., Voronkov, A.: First-order theorem proving and vampire. In: Proc. of CAV. pp. 1–35 (2013)
9. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: a proof assistant for higher-order logic, vol. 2283. Springer Science & Business Media (2002)
10. Noschinski, L.: Graph theory. Archive of Formal Proofs (Apr 2013), http://isa-afp.org/entries/Graph_Theory.html, Formal proof development
11. Roditty, Y., Shoham, B., Yuster, R.: Monotone paths in edge-ordered sparse graphs. Discrete Mathematics **226**(1-3), 411–417 (2001)
12. Yuster, R.: Large monotone paths in graphs with bounded degree. Graphs and Combinatorics **17**(3), 579–587 (2001)