Relational Minimum Spanning Tree Algorithms

Walter Guttmann and Nicolas Robinson-O’Brien

February 23, 2021

Abstract

We verify the correctness of Prim’s, Kruskal’s and Borůvka’s minimum spanning tree algorithms based on algebras for aggregation and minimisation.

Contents

1 Overview 1
  1.1 Prim’s and Kruskal’s minimum spanning tree algorithms . . . 2
  1.2 Borůvka’s minimum spanning tree algorithm . . . . . . . . 2

2 Kruskal’s Minimum Spanning Tree Algorithm 2

3 Prim’s Minimum Spanning Tree Algorithm 4

4 Borůvka’s Minimum Spanning Tree Algorithm 7
  4.1 General results . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
  4.2 An operation to select components . . . . . . . . . . . . . . 9
  4.3 m-k-Stone-Kleene relation algebras . . . . . . . . . . . . . . 10
    4.3.1 Components of forests and big forests . . . . . . . . . . 12
    4.3.2 Identifying arcs . . . . . . . . . . . . . . . . . . . . . . 13
    4.3.3 Comparison of edge weights . . . . . . . . . . . . . . . 13
    4.3.4 Maintenance of algorithm invariants . . . . . . . . . . 15
  4.4 Formalization and correctness proof . . . . . . . . . . . . . . 17

1 Overview

The theories described in this document prove the correctness of Prim’s, Kruskal’s and Borůvka’s minimum spanning tree algorithms. Specifications and algorithms work in Stone-Kleene relation algebras extended by operations for aggregation and minimisation. The algorithms are implemented in a simple imperative language and their proof uses Hoare logic. The correctness proofs are discussed in [3, 5, 6, 8].
1.1 Prim’s and Kruskal’s minimum spanning tree algorithms

A framework based on Stone relation algebras and Kleene algebras and extended by operations for aggregation and minimisation was presented by the first author in [3, 5] and used to formally verify the correctness of Prim’s minimum spanning tree algorithm. It was extended in [6] and applied to prove the correctness of Kruskal’s minimum spanning tree algorithm.

Two theories, one each for Prim’s and Kruskal’s algorithms, prove total correctness of these algorithms. As case studies for the algebraic framework, these two theories combined were originally part of another AFP entry [4].

1.2 Borůvka’s minimum spanning tree algorithm

Otakar Borůvka formalised the minimum spanning tree problem and proposed a solution to it [1]. Borůvka’s original paper is written in Czech; translations of varying completeness can be found in [2, 7].

The theory for Borůvka’s minimum spanning tree algorithm proves partial correctness of this algorithm. This work is based on the same algebraic framework as the proof of Kruskal’s algorithm; in particular it uses many theories from the hierarchy underlying [4].

The theory for Borůvka’s algorithm formally verifies results from the second author’s Master’s thesis [8]. Certain lemmas in this theory are numbered for easy correlation to theorems from the thesis.

2 Kruskal’s Minimum Spanning Tree Algorithm

In this theory we prove total correctness of Kruskal’s minimum spanning tree algorithm. The proof uses the following steps [6]. We first establish that the algorithm terminates and constructs a spanning tree. This is a constructive proof of the existence of a spanning tree; any spanning tree algorithm could be used for this. We then conclude that a minimum spanning tree exists. This is necessary to establish the invariant for the actual correctness proof, which shows that Kruskal’s algorithm produces a minimum spanning tree.

theory Kruskal


begin

context m-kleene-algebra

begin

definition spanning-forest f g ≡ forest f ∨ f ≤ −−g ∧ components g ≤ forest-components f ∨ regular f

definition minimum-spanning-forest f g ≡ spanning-forest f g ∨ (∀ u. spanning-forest u g −→ sum (f ∩ g)) ≤ sum (u ∩ g))
definition kruskal-spanning-invariant \( f \; g \; h \equiv \text{symmetric } g \land h = h^T \land g \cap \neg\neg h \)

definition kruskal-invariant \( f \; g \; h \equiv \text{kruskal-spanning-invariant } f \; g \; h \land (\exists w . \text{minimum-spanning-forest } w \; g \land f \leq w \cup w^T) \)

We first show two verification conditions which are used in both correctness proofs.

lemma kruskal-vc-1:
assumes symmetric \( g \)
shows kruskal-spanning-invariant bot \( g \) \( g \)
⟨proof⟩

lemma kruskal-vc-2:
assumes kruskal-spanning-invariant \( f \; g \; h \) and \( h \neq \text{bot} \)
shows (minarc \( h \leq \text{forest-components } f \rightarrow \text{kruskal-spanning-invariant } ((f \cap -\text{top} \ast \text{minarc } h \ast f^T) \cup (f \cap \text{top} \ast \text{minarc } h \ast f^T) \cup \text{minarc } h) \) \( g \) \( h \cap \neg\text{minarc } h \cap \neg\text{minarc } h^T \) \land \text{card} \{ x . \text{regular } x \land x \leq \neg\neg h \land x \leq \neg\neg \neg h \} \land \neg \text{minarc } h \leq \text{forest-components } f \rightarrow \text{kruskal-spanning-invariant } f \; g \) \( h \cap \neg\text{minarc } h \cap \neg\text{minarc } h^T \) \land \text{card} \{ x . \text{regular } x \land x \leq \neg\neg h \land x \leq \neg\neg \neg h \} \}
⟨proof⟩

The following result shows that Kruskal’s algorithm terminates and constructs a spanning tree. We cannot yet show that this is a minimum spanning tree.

theorem kruskal-spanning:
VARS \( e \; f \; h \)
[ symmetric \( g \) ]
\( f := \text{bot} ; \)
\( h := g ; \)
\( \text{WHILE } h \neq \text{bot} \)
INV { kruskal-spanning-invariant \( f \; g \; h \) }
VAR { \text{card} \{ x . \text{regular } x \land x \leq \neg\neg h \} }
DO \( e := \text{minarc } h ; \)
\( \text{IF } e \leq \text{forest-components } f \text{ THEN} \)
f := (f \cap -(\text{top} \ast e \ast f^T)) \cup (f \cap \text{top} \ast e \ast f^T) \cup e
ELSE
SKIP
FI;
\( h := h \cap -e \cap -e^T \)
OD
[ spanning-forest \( f \; g \) ]
⟨proof⟩

Because we have shown total correctness, we conclude that a spanning tree exists.
lemma \textit{kruskal-exists-spanning}:
\[ \text{symmetric } g \implies \exists f . \text{spanning-forest } f \ g \]  

This implies that a minimum spanning tree exists, which is used in the subsequent correctness proof.

lemma \textit{kruskal-exists-minimal-spanning}:
\[ \text{assumes symmetric } g \]  
\[ \text{shows } \exists f . \text{minimum-spanning-forest } f \ g \]  

Kruskal’s minimum spanning tree algorithm terminates and is correct. This is the same algorithm that is used in the previous correctness proof, with the same precondition and variant, but with a different invariant and postcondition.

theorem \textit{kruskal}:
\[ \text{VARS } e \ f \ h \]  
\[ \text{[ symmetric } g \]  
\[ f := \text{bot}; \]  
\[ h := g; \]  
\[ \text{WHILE } h \neq \text{bot} \]  
\[ \text{INV } \{ \text{kruskal-invariant } f \ g \ h \} \]  
\[ \text{VAR } \{ \text{card } \{ x : \text{regular } x \land x \leq -h \} \} \]  
\[ \text{DO } e := \text{minarc } h; \]  
\[ \text{IF } e \leq \neg \text{forest-components } f \text{ THEN} \]  
\[ f := (f \cap \neg (\text{top} \times e \times f^T)) \cup (f \cap \text{top} \times e \times f^T)^T \cup e \]  
\[ \text{ELSE} \]  
\[ \text{SKIP} \]  
\[ \text{FI}; \]  
\[ h := h \cap -e \cap -e^T \]  
\[ \text{OD} \]  
\[ \text{[ minimum-spanning-forest } f \ g \] \]  

end

3 Prim’s Minimum Spanning Tree Algorithm

In this theory we prove total correctness of Prim’s minimum spanning tree algorithm. The proof has the same overall structure as the total-correctness proof of Kruskal’s algorithm [6]. The partial-correctness proof of Prim’s algorithm is discussed in [3, 5].

theory \textit{Prim} 

imports \textit{HOL-Hoare.Hoare-Logic Aggregation-Algebras.Aggregation-Algebras}
begin

context m-kleene-algebra
begin

abbreviation component g r ≡ r \top (−−g)^* 

definition spanning-tree t g r ≡ forest t \land t \leq (component g r)\top (component g r) \cap −−g \land component g r \leq r\top \ast \ast \ast \land regular t 

definition minimum-spanning-tree t g r ≡ spanning-tree t g r \land (∀ u . spanning-tree u g r → sum (t \cap g) \leq sum (u \cap g)) 

definition prim-precondition g r ≡ g = g\top \land injective r \land vector r \land regular r 

definition prim-spanning-invariant t v g r ≡ prim-precondition g r \land v\top = r\top \ast t \ast \land spanning-tree t (v \ast v\top \cap g) r 

definition prim-invariant t v g r ≡ prim-spanning-invariant t v g r \land (∃ w . minimum-spanning-tree w g r \land t \leq w) 

lemma span-tree-split: 
  assumes vector r 
  shows t \leq (component g r)\top (component g r) \cap −−g \iff (t \leq (component g r)\top \land t \leq component g r \land t \leq −−g) 
  ⟨proof⟩

lemma span-tree-component: 
  assumes spanning-tree t g r 
  shows component g r = component t r 
  ⟨proof⟩

We first show three verification conditions which are used in both correctness proofs.

lemma prim-vc-1: 
  assumes prim-precondition g r 
  shows prim-spanning-invariant bot r g r 
  ⟨proof⟩

lemma prim-vc-2: 
  assumes prim-spanning-invariant t v g r 
  and v \ast −v\top \cap g \neq bot 
  shows prim-spanning-invariant (t \cup minarc (v \ast −v\top \cap g)) (v \cup minarc (v \ast −v\top \cap g)\top \ast top) g r \land card \{ x . regular x \land x \leq component g r \land x \leq −(v \cup minarc (v \ast −v\top \cap g)\top \ast top)\top \} < card \{ x . regular x \land x \leq component g r \land x \leq −v\top \} 
  ⟨proof⟩

lemma prim-vc-3: 
  assumes prim-spanning-invariant t v g r 
  and v \ast −v\top \cap g = bot 
  shows spanning-tree t g r 
  ⟨proof⟩
The following result shows that Prim’s algorithm terminates and constructs a spanning tree. We cannot yet show that this is a minimum spanning tree.

**Theorem** prim-spanning:

\[
\text{VARS } t \ v \ e \\
\text{[ prim-precondition } g \ r \ ] \\
t := \bot; \\
v := r; \\
\text{WHILE } v \neq -v^T \cap g \neq \bot \\
\text{INV } \{ \text{prim-spanning-invariant } t \ v \ g \ r \} \\
\text{VAR } \{ \text{card } \{ x \ . \ \text{regular } x \land x \leq \text{component } g \cap -v^T \} \} \\
\text{DO } e := \text{minarc } (v \neq -v^T \cap g); \\
\text{t := t }\cup\text{ e;} \\
v := v \cup e^T \ast \text{top} \\
\text{OD} \\
\text{[ spanning-tree } t \ g \ r \] \\
\langle \text{proof} \rangle
\]

Because we have shown total correctness, we conclude that a spanning tree exists.

**Lemma** prim-exists-spanning:

\[ \text{prim-precondition } g \ r \implies \exists t \ . \ \text{spanning-tree } t \ g \ r \] \\
\langle \text{proof} \rangle

This implies that a minimum spanning tree exists, which is used in the subsequent correctness proof.

**Lemma** prim-exists-minimal-spanning:

\[ \text{assumes prim-precondition } g \ r \] \\
\text{shows } \exists t \ . \ \text{minimum-spanning-tree } t \ g \ r \\
\langle \text{proof} \rangle

Prim’s minimum spanning tree algorithm terminates and is correct. This is the same algorithm that is used in the previous correctness proof, with the same precondition and variant, but with a different invariant and postcondition.

**Theorem** prim:

\[
\text{VARS } t \ v \ e \\
\text{[ prim-precondition } g \ r \land (\exists w \ . \ \text{minimum-spanning-tree } w \ g \ r) \] \\
t := \bot; \\
v := r; \\
\text{WHILE } v \neq -v^T \cap g \neq \bot \\
\text{INV } \{ \text{prim-invariant } t \ v \ g \ r \} \\
\text{VAR } \{ \text{card } \{ x \ . \ \text{regular } x \land x \leq \text{component } g \cap -v^T \} \} \\
\text{DO } e := \text{minarc } (v \neq -v^T \cap g); \\
\text{t := t }\cup\text{ e;} \\
v := v \cup e^T \ast \text{top} \\
\text{OD} \\
\text{[ minimum-spanning-tree } t \ g \ r \] \\
\langle \text{proof} \rangle
4 Borůvka’s Minimum Spanning Tree Algorithm

In this theory we prove partial correctness of Borůvka’s minimum spanning tree algorithm.

theory Borwska

imports
  Relational-Disjoint-Set-Forests.Disjoint-Set-Forests
  Kruskal

begin

4.1 General results

The proof is carried out in $m$-$k$-Stone-Kleene relation algebras. In this section we give results that hold more generally.

classic context stone-kleene-relation-algebra

begin

definition big-forest $H$ $d$ ≡
  equivalence $H$
  ∧ $d \leq -H$
  ∧ univalent $(H \ast d)$
  ∧ $H \cap d \ast d^T \leq 1$
  ∧ $(H \ast d)^+ \leq -H$

definition bf-between-points $p$ $q$ $H$ $d$ ≡
  point $p$
  ∧ point $q$
  ∧ $p \leq (H \ast d)^+ \ast (H \ast d)$

definition bf-between-arcs $a$ $b$ $H$ $d$ ≡
  arc $a$
  ∧ arc $b$
  ∧ $a^T \ast top \leq (H \ast d)^+ \ast (H \ast b) \ast top$

Theorem 3

lemma He-eq-He-THe-star:
  assumes arc $e$
  and equivalence $H$
  shows $H \ast e = H \ast e \ast (top \ast H \ast e)^+$

(proof)

lemma path-through-components:
  assumes equivalence $H$
  and arc $e$


shows \((H \ast (d \sqcup e))^\ast = (H \ast d)^\ast \sqcup (H \ast d)^\ast \ast H \ast e \ast (H \ast d)^\ast\)

\(\langle \text{proof} \rangle\)

**Lemma simplify-f:**

- **Assumes** regular \(p\)
  - and regular \(e\)
- **Shows** \((f \cap - e^T \cap - p) \sqcup (f \cap - e^T \cap p) \sqcup (f \cap - e^T \cap - p)^T \sqcup e^T \sqcup e^T \sqcup e = f \sqcup f^T \sqcup e \sqcup e^T\)

\(\langle \text{proof} \rangle\)

**Lemma simplify-forest-components-f:**

- **Assumes** regular \(p\)
  - and regular \(e\)
  - and injective \((f \cap - e^T \cap - p) \sqcup (f \cap - e^T \cap p) \sqcup e\)
  - and injective \(f\)
- **Shows** forest-components \(((f \cap - e^T \cap - p) \sqcup (f \cap - e^T \cap p) \sqcup e) = (f \sqcup f^T \sqcup e \sqcup e^T)^\ast\)

\(\langle \text{proof} \rangle\)

**Lemma components-disj-increasing:**

- **Assumes** regular \(p\)
  - and regular \(e\)
  - and injective \((f \cap - e^T \cap - p) \sqcup (f \cap - e^T \cap p) \sqcup e\)
  - and injective \(f\)
- **Shows** forest-components \(f \leq\) forest-components \(((f \cap - e^T \cap - p) \sqcup (f \cap - e^T \cap p) \sqcup e) \sqcup e \sqcup e^T\)

\(\langle \text{proof} \rangle\)

**Lemma fch-equivalence:**

- **Assumes** forest \(h\)
- **Shows** equivalence (forest-components \(h\))

\(\langle \text{proof} \rangle\)

**Lemma big-forest-path-split-1:**

- **Assumes** arc \(a\)
  - and equivalence \(H\)
- **Shows** \((H \ast d)^\ast \ast H \ast a \ast top = (H \ast (d \cap - a))^\ast \ast H \ast a \ast top\)

\(\langle \text{proof} \rangle\)

**Lemma dTransHd-le-1:**

- **Assumes** equivalence \(H\)
  - and univalent \((H \ast d)\)
- **Shows** \(d^T \ast H \ast d \leq 1\)

\(\langle \text{proof} \rangle\)

**Lemma HcompaT-le-compHaT:**

- **Assumes** equivalence \(H\)
  - and injective \((a \ast top)\)
- **Shows** \(-H \ast a \ast top \leq -(H \ast a \ast top)\)
Theorem 4

**Lemma** expand-big-forest:
**Assumes** big-forest \( H d \)
**Shows** \((d^T * H)^* * (H * d)^* = (d^T * H)^* \cup (H * d)^*\)

**Lemma** big-forest-path-bot:
**Assumes** arc \( a \)
**And** \( a \leq d \)
**And** big-forest \( H d \)
**Shows** \((d \cap - a)^T * (H * a * top) \leq bot\)

**Lemma** big-forest-path-split-2:
**Assumes** arc \( a \)
**And** \( a \leq d \)
**And** big-forest \( H d \)
**Shows** \((H * (d \cap - a))^* * H * a * top = (H * ((d \cap - a) \cup (d \cap - a)^T))^* * H * a * top\)

end

4.2 An operation to select components

We introduce the operation **choose-component**.

* Axiom **component-in-v** expresses that the result of **choose-component** is contained in the set of vertices, \( v \), we are selecting from, ignoring the weights.

* Axiom **component-is-vector** states that the result of **choose-component** is a vector.

* Axiom **component-is-regular** states that the result of **choose-component** is regular.

* Axiom **component-is-connected** states that any two vertices from the result of **choose-component** are connected in \( e \).

* Axiom **component-single** states that the result of **choose-component** is closed under being connected in \( e \).

* Finally, axiom **component-not-bot-when-v-bot-bot** expresses that the operation **choose-component** returns a non-empty component if the input satisfies the given criteria.
class choose-component =
  fixes choose-component :: 'a ⇒ 'a ⇒ 'a

class choose-component-algebra = choose-component + stone-relation-algebra +
  assumes component-in-v: choose-component e v ≤ v
  assumes component-is-vector: vector (choose-component e v)
  assumes component-is-regular: regular (choose-component e v)
  assumes component-is-connected: choose-component e v * (choose-component e v)^T ≤ e
  assumes component-single: choose-component e v = e * choose-component e v
  assumes component-not-bot-when-v-bot-bot:
    regular e
    ∧ equivalence e
    ∧ vector v
    ∧ regular v
    ∧ e * v = v
    ∧ v ≠ bot → choose-component e v ≠ bot

  Theorem 1

  Every m-kleene-algebra is an instance of choose-component-algebra when
  the choose-component operation is defined as follows:

context m-kleene-algebra
begin

definition choose-component-input-condition e v ≡
  regular e
  ∧ equivalence e
  ∧ vector v
  ∧ regular v
  ∧ e * v = v
  ∧ v ≠ bot

definition m-choose-component e v ≡
  if choose-component-input-condition e v then
    e * minarc(v) * top
  else
    bot

sublocale m-choose-component-algebra: choose-component-algebra where
choose-component = m-choose-component
⟨proof⟩

end

4.3 m-k-Stone-Kleene relation algebras

m-k-Stone-Kleene relation algebras are an extension of m-Kleene algebras
where the choose-component operation has been added.

class m-kleene-algebra-choose-component =
A selected-edge is a minimum-weight edge whose source is in a component, with respect to \( h, j \) and \( g \), and whose target is not in that component.

abbreviation selected-edge \( h j g \equiv \minarc \) (choose-component (forest-components \( h \)) \( j \)) \( \ldots \) (choose-component (forest-components \( h \)) \( j^T \cap g \))

A path is any sequence of edges in the forest, \( f \), of the graph, \( g \), backwards from the target of the selected-edge to a root in \( f \).

abbreviation path \( f h j g \equiv \top \ast \) selected-edge \( h j g \ast (f \cap \ast \) selected-edge \( h j g^T)^T \ast \)

definition boruvka-outer-invariant \( f g \equiv \)
\sum symmetric \( g \)
\sum forest \( f \)
\sum f \leq \sum g
\sum regular \( f \)
\sum (\exists w \cdot \) minimum-spanning-forest \( w g \wedge f \leq w \cup w^T \)

definition boruvka-inner-invariant \( j f h g d \equiv \)
boruvka-outer-invariant \( f g \)
\sum g \neq \bot
\sum vector \( j \)
\sum regular \( j \)
\sum boruvka-outer-invariant \( h g \)
\sum forest \( h \)
\sum forest-components \( h \leq \) forest-components \( f \)
\sum \) big-forest \( \) (forest-components \( h \)) \( d \)
\sum d \ast to\( p \leq \sum j \)
\sum forest-components \( h \ast j = \sum j \)
\sum forest-components \( f = \) (forest-components \( h \ast (d \cup d^T))^{\ast \ast} \ast \) forest-components \( h \)
\sum f \cup f^T = h \cup h^T \cup d \cup d^T
\sum (\forall a b \cdot \) bf-between-arcs \( a b \) (forest-components \( h \)) \( d \wedge a \leq \sum \)
\sum (forest-components \( h \)) \( \cap \sum g \wedge b \leq d \)
\sum \ast \) sum \( (b \cap g) \leq \sum \) (\( a \cap g \))
\sum regular \( d \)

lemma expression-equivalent-without-e-complement:
assumes selected-edge \( h j g \leq \sum \) forest-components \( f \)
shows \( f \cap \sum \) (selected-edge \( h j g \)) \( \sum \) f \cap \sum \) (selected-edge \( h j g \)) \( \sum (f \cap \sum \) (selected-edge \( h j g \)) \( \sum (f \cap \sum (path f h j g))^{T} \cup \) selected-edge \( h j g \)

Theorem 2
The source of the selected-edge is contained in \( j \), the vector describing those vertices still to be processed in the inner loop of Borůvka’s algorithm.

**Lemma et-below-j:**
- Assumes vector \( j \)
- and regular \( j \)
- and \( j \neq \text{bot} \)
- Shows selected-edge \( h j g \ast \text{top} \leq j \)

**4.3.1 Components of forests and big forests**

We prove a number of properties about big-forest and forest-components.

**Lemma fc-j-eq-j-inv:**
- Assumes forest \( h \)
- and forest-components \( h \ast j = j \)
- Shows forest-components \( h \ast (j \cap - \text{choose-component (forest-components} h) j) = j \cap - \text{choose-component (forest-components} h) j \)

**Theorem 5**

There is a path in the big-forest between edges \( a \) and \( b \) if and only if there is either a path in the big-forest from \( a \) to \( b \) or one from \( a \) to \( c \) and one from \( c \) to \( b \).

**Lemma big-forest-path-split-disj:**
- Assumes equivalence \( H \)
- and arc \( c \)
- and regular \( a \) and regular \( b \) and regular \( c \) and regular \( d \) and regular \( H \)
- Shows bf-between-arcs \( a b H (d \sqcup c) \leftrightarrow \text{bf-between-arcs} a b H d \lor \text{bf-between-arcs} a c H d \land \text{bf-between-arcs} c b H d \)

**Lemma dT-He-eq-bot:**
- Assumes vector \( j \)
- and regular \( j \)
- and \( d \ast \text{top} \leq - j \)
- and forest-components \( h \ast j = j \)
- and \( j \neq \text{bot} \)
- Shows \( dT \ast \text{forest-components} h \ast \text{selected-edge} h j g \leq \text{bot} \)

**Lemma big-forest-d-U-e:**
- Assumes forest \( f \)
- and vector \( j \)
- and regular \( j \)
- and forest \( h \)
- and forest-components \( h \leq \text{forest-components} f \)
- and big-forest \( (\text{forest-components} h) d \)
- and \( d \ast \text{top} \leq - j \)
and forest-components $h \ast j = j$
and $f \sqcup f^T = h \sqcup h^T \sqcup d \sqcup d^T$
and selected-edge $h \ast j \ast g \leq \neg$ forest-components $f$
and selected-edge $h \ast j \ast g \neq \bot$
and $j \neq \bot$
shows big-forest $(\text{forest-components } h) \sqcup \text{selected-edge } h \ast j \ast g$

\section*{4.3.2 Identifying arcs}

The expression $d \sqcap e^T H \sqcap (Hd^T)^* Ha^T \sqcap$ identifies the edge incoming to the component that the selected-edge, $e$, is outgoing from and which is on the path from edge $a$ to $e$. Here, we prove this expression is an arc.

\textbf{lemma shows-arc-x:}
\begin{itemize}
\item \textbf{assumes} big-forest $H \ast d$
\item and $bf\text{-between-arcs } a e H d$
\item and $H \ast d \ast (H \ast d)^* \leq \neg H$
\item and $\neg a^T \ast top \leq H \ast e \ast top$
\item and regular $a$
\item and regular $e$
\item and regular $H$
\item and regular $d$
\end{itemize}

shows arc $(d \sqcap top \ast e^T \ast H \sqcap (H \ast d^T)^* \ast H \ast a^T \ast top)$

\section*{4.3.3 Comparison of edge weights}

In this section we compare the weight of the selected-edge with other edges of interest. Theorems 8, 9, 10 and 11 are supporting lemmas. For example, Theorem 8 is used to show that the selected-edge has its source inside and its target outside the component it is chosen for.
Theorem 8

Lemma e-leq-c-c-complement-transpose-general:
assumes \( e = \minarc (c * -(c)^T \cap g) \)
and regular \( c \)
shows \( e \leq c * -(c)^T \)
⟨proof⟩

Theorem 9

Lemma x-leq-c-transpose-general:
assumes forest \( h \)
and vector \( c \)
and \( x^T * top \leq \text{forest-components}(h) * e * top \)
and \( e \leq c * -(c)^T \)
and \( c = \text{forest-components}(h) * e \)
shows \( x \leq c^T \)
⟨proof⟩

Theorem 10

Lemma x-leq-c-complement-general:
assumes vector \( c \)
and \( c * c^T \leq \text{forest-components} h \)
and \( x \leq c^T \)
and \( x \leq -\text{forest-components} h \)
shows \( x \leq -c \)
⟨proof⟩

Theorem 11

Lemma sum-e-below-sum-x-when-outgoing-same-component-general:
assumes \( e = \minarc (c * -(c)^T \cap g) \)
and regular \( c \)
and forest \( h \)
and vector \( c \)
and \( x^T * top \leq (\text{forest-components} h) * e * top \)
and \( c = (\text{forest-components} h) * e \)
and \( c * c^T \leq \text{forest-components} h \)
and \( x \leq -\text{forest-components} h \cap -g \)
and symmetric \( g \)
and arc \( x \)
and \( c \neq \text{bot} \)
shows \( \text{sum} (e \cap g) \leq \text{sum} (x \cap g) \)
⟨proof⟩

Lemma sum-e-below-sum-x-when-outgoing-same-component:
assumes symmetric \( g \)
and vector \( j \)
and forest \( h \)
and \( x \leq -\text{forest-components} h \cap -g \)
and \( x^T * top \leq \text{forest-components} h * \text{selected-edge} h j g * top \)
and \( j \neq \text{bot} \)
and arc x
shows \(\text{sum} (\text{selected-edge} h \ j \ g \cap g) \leq \text{sum} (x \cap g)\)

\(\langle\text{proof}\rangle\)

If there is a path in the big-forest from an edge between components, \(a\), to the selected-edge, \(e\), then the weight of \(e\) is no greater than the weight of \(a\). This is because either,

* the edges \(a\) and \(e\) are adjacent the same component so that we can use \textit{sum-e-below-sum-x-when-outgoing-same-component}, or

* there is at least one edge between \(a\) and \(e\), namely \(x\), the edge incoming to the component that \(e\) is outgoing from. The path from \(a\) to \(e\) is split on \(x\) using \textit{big-forest-path-split-disj}. We show that the weight of \(e\) is no greater than the weight of \(x\) by making use of lemma \textit{sum-e-below-sum-x-when-outgoing-same-component}. We define \(x\) in a way that we can show that the weight of \(x\) is no greater than the weight of \(a\) using the invariant. Then, it follows that the weight of \(e\) is no greater than the weight of \(a\) owing to transitivity.

\textbf{lemma} a-to-e-in-bigforest:
\textbf{assumes} symmetric \(g\)
\textbf{and} \(f \leq \{\{\}\}\)
\textbf{and} vector \(j\)
\textbf{and} forest \(h\)
\textbf{and} big-forest \(\langle\text{forest-components} h \rangle d\)
\textbf{and} \(f \sqcup f^T = h \sqcup h^T \sqcup d \sqcup d^T\)
\textbf{and} \((\forall a b . \text{bf-between-arcs} a b \langle\text{forest-components} h \rangle d \wedge a \leq (\text{forest-components} h) \cap \{\{\}\} \wedge b \leq d \rightarrow \text{sum}(b \cap g) \leq \text{sum}(a \cap g))\)
\textbf{and} regular \(d\)
\textbf{and} \(j \neq \text{bot}\)
\textbf{and} \(b = \text{selected-edge} h \ j \ g\)
\textbf{and} arc \(a\)
\textbf{and} \(\text{bf-between-arcs} a b \langle\text{forest-components} h \rangle (d \sqcup \text{selected-edge} h j g)\)
\textbf{and} \(a \leq \text{forest-components} h \cap \{\{\}\}\)
\textbf{and} regular \(h\)
\textbf{shows} \(\text{sum} (b \cap g) \leq \text{sum} (a \cap g)\)
\(\langle\text{proof}\rangle\)

\subsection{4.3.4 Maintenance of algorithm invariants}

In this section, most of the work is done to maintain the invariants of the inner and outer loops of the algorithm. In particular, we use \textit{exists-a-w} to maintain that \(f\) can be extended to a minimum spanning forest.

\textbf{lemma} boruvka-exchange-spanning-inv:
\textbf{assumes} forest \(v\)
\textbf{and} \(v^* \ast e^T = e^T\)
\textbf{and} \(i \leq v \cap \text{top} \ast e^T \ast w^T^*\)
and arc i
and arc e
and \( v \leq -g \)
and \( w \leq -g \)
and \( e \leq -g \)
and components \( g \leq \text{forest-components } v \)
shows \( i \leq (v \cap -i)^T * e^T * \text{top} \)

\( \langle \text{proof} \rangle \)

\textbf{lemma} exists-a-w:
\textbf{assumes} symmetric \( g \)
and forest \( f \)
and \( f \leq -g \)
and regular \( f \)
and \( (\exists w. \text{minimum-spanning-forest } w, g \land f \leq w \cup w^T) \)
and vector \( j \)
and regular \( j \)
and forest \( h \)
and forest-components \( h \leq \text{forest-components } f \)
and big-forest (forest-components \( h \)) \( d \)
and \( d * \text{top} \leq -j \)
and forest-components \( h * j = j \)
and forest-components \( f = (\text{forest-components } h * (d \cup d^T))^T * \)
forest-components \( h \)
and \( f \cup f^T = h \cup h^T \cup d \cup d^T \)
and \((\forall a b. \text{bf-between-arcs } a, b (\text{forest-components } h)) \ d \land a \leq -g \land b \leq d \implies \text{sum}(b \cap g) \leq \text{sum}(a \cap g)) \)
and regular \( d \)
and selected-edge \( h, j, g \leq - \text{forest-components } f \)
and selected-edge \( h, j, g \neq \text{bot} \)
and \( j \neq \text{bot} \)
and regular \( h \)
and \( h \leq -g \)
shows \( (\exists w. \text{minimum-spanning-forest } w, g \land f \cap - (\text{selected-edge } h, j, g)^T \cap - (\text{path } f, h, j, g) \cup (f \cap - (\text{selected-edge } h, j, g)^T \cap (\text{path } f, h, j, g)) \leq w \cup w^T \)
\( \langle \text{proof} \rangle \)

\textbf{lemma} boruvka-outer-invariant-when-e-not-bot:
\textbf{assumes} boruvka-inner-invariant \( j, f, h, g, d \)
and \( j \neq \text{bot} \)
and selected-edge \( h, j, g \leq - \text{forest-components } f \)
and selected-edge \( h, j, g \neq \text{bot} \)
shows boruvka-outer-invariant (f \cap - selected-edge h, j, g)^T \cap - path f, h, j, g \cup (f \cap - selected-edge h, j, g)^T \cap path f, h, j, g)^T \cup selected-edge h, j, g) \ g \)
\( \langle \text{proof} \rangle \)

\textbf{lemma} second-inner-invariant-when-e-not-bot:
\textbf{assumes} boruvka-inner-invariant \( j, f, h, g, d \)
and \( j \neq \text{bot} \)
and selected-edge \( h \ j \ g \preceq -\text{forest-components} \ f \)
and selected-edge \( h \ j \ g \neq \text{bot} \)

shows boruvka-inner-invariant

\[(j \cap - \text{choose-component} (\text{forest-components} h) j)\]
\[(f \cap - \text{selected-edge} h \ j \ g^T \cap - \text{path} f \ h \ j \ g \cup\]
\[(f \cap - \text{selected-edge} h \ j \ g^T \cap \text{path} f \ h \ j \ g)^T \cup\]
selected-edge \( h \ j \ g \)
\[h \ g \ (d \cup \text{selected-edge} h \ j \ g)\]

(proof)

lemma second-inner-invariant-when-e-bot:
assumes selected-edge \( h \ j \ g = \text{bot} \)
and selected-edge \( h \ j \ g \leq -\text{forest-components} f \)
and boruvka-inner-invariant \( j f h g d \)

shows boruvka-inner-invariant

\[(j \cap - \text{choose-component} (\text{forest-components} h) j)\]
\[(f \cap - \text{selected-edge} h \ j \ g^T \cap - \text{path} f \ h \ j \ g \cup\]
\[(f \cap - \text{selected-edge} h \ j \ g^T \cap \text{path} f \ h \ j \ g)^T \cup\]
selected-edge \( h \ j \ g \)
\[h \ g \ (d \cup \text{selected-edge} h \ j \ g)\]

(proof)

4.4 Formalization and correctness proof

The following result shows that Borůvka’s algorithm constructs a minimum spanning forest. We have the same postcondition as the proof of Kruskal’s minimum spanning tree algorithm. We show only partial correctness.

theorem boruvka-mst:

VARS \( f \ j \ h \ c \ e \ d \)
\{ symmetric \( g \) \}
\( f := \text{bot}; \)

WHILE \( \neg (\text{forest-components} \ f) \cap g \neq \text{bot} \)

INV \{ boruvka-outer-invariant \( f \ g \) \}

DO
\( j := \text{top}; \)
\( h := f; \)
\( d := \text{bot}; \)

WHILE \( j \neq \text{bot} \)

INV \{ boruvka-inner-invariant \( j f h g d \) \}

DO
\( c := \text{choose-component} (\text{forest-components} h) j; \)
\( e := \text{minarc}(e \ast -c^T \cap g); \)
IF \( e \leq -(\text{forest-components} f) \) THEN
\( f := f \cap -c^T; \)
\( f := (f \cap - (\text{top} \ast e \ast f^{T*})) \cup (f \cap \text{top} \ast e \ast f^{T*})^T \cup e; \)
\( d := d \cup e \)
ELSE
SKIP

Proof
\begin{verbatim}
FI;
j := j \setminus -c
OD
\{ minimum-spanning-forest f g \}
(proof)
end
end
\end{verbatim}

References


