Relation-algebraic Verification of Borůvka's Minimum Spanning Tree Algorithm

Walter Guttmann and Nicolas Robinson-O'Brien

Department of Computer Science and Software Engineering,
University of Canterbury, Christchurch, New Zealand
walter.guttmann@canterbury.ac.nz, nic.robinson-obrien@pg.canterbury.ac.nz

Abstract. Previous work introduced a relation-algebraic framework for reasoning about weighted-graph algorithms. We use this framework to prove partial correctness of a sequential version of Borůvka's minimum spanning tree algorithm. This is the first formal proof of correctness for this algorithm. We also discuss new abstractions that make it easier to reason about weighted graphs.

Keywords: algorithm verification, formal methods, Kleene algebras, relation algebras, Stone algebras, weighted graphs

1 Introduction

The Minimum Spanning Tree (MST) problem is to find a subset of the edges of a graph that form a tree, connecting the graph's vertices, where the sum of the weights of the edges is minimal [32]. In 1926, Otakar Borůvka described the MST problem and gave an algorithm to solve it [6]. He was perhaps the first person to do so [12]. Borůvka's original paper is written in Czech; a translation can be found in [24]. Borůvka's MST algorithm was independently discovered by Choquet [8], Florek et al. [10], and Sollin [2]. Many textbooks do not treat Borůvka's MST algorithm with the same exposure as the algorithms of Prim [29] and Kruskal [21]; nevertheless, it is significant for its influence on running-time complexity improvements for MST algorithms [7,18,34].

Borůvka's MST algorithm computes a minimum spanning tree of a weighted, connected, undirected graph whose edge weights are distinct. The algorithm begins by initialising a forest with n trees, each containing a single vertex, where n is the number of vertices in the graph. While there is more than one tree in that forest, the following step is repeated. For each tree in the forest, find the edge in the graph with the smallest weight among all edges that leave the tree; all edges found in this way are added to the forest in this step.

A relation-algebraic framework for MST problems was introduced in [13] and has been used to formally verify Prim's [14] and Kruskal's [15] MST algorithms. In the present paper, we use this framework to formally verify a sequential version of Borůvka's MST algorithm. Its contributions are

– In Sect. 3 we define components of a graph in terms of a vector, representing a subset of vertices, and an equivalence, representing connectivity. We also introduce an operation, k, to select an arbitrary component of a graph. We show that this operation can be expressed in m-Kleene algebras.
– In Sect. 4 we present axioms for forests modulo an equivalence, a new abstraction that can represent a forest-like structure where clusters of vertices are conceptually collapsed to points forming a forest with the edges that connect them. A number of properties of this abstraction are also given. Additionally, we study paths between vertices in a forest modulo an equivalence and present a theorem for splitting such a path on one of its edges.
– In Sect. 5 we formalise Borůvka's MST algorithm in m-k-Stone-Kleene relation algebras. The formalisation uses the k operation.
– In Sect. 6 we discuss key invariants of our correctness proof of Borůvka's MST algorithm and highlight how we have used the abstractions introduced in previous sections. This is the first formal correctness proof of this algorithm.

Additionally, we have used Isabelle/HOL [27] to formally verify all results in this paper. The corresponding theories are available in the Archive of Formal Proofs [16] and proofs are omitted from this paper. The PDF version of this paper includes links to the relevant theorems and definitions, hosted online. Our proof of Borůvka's MST algorithm uses a Hoare-logic library and verification conditions are generated using a tactic of that library [25,26]. Further details of the correctness proof are described in [30].

There are other recent works verifying MST algorithms in Isabelle/HOL. For example, a functional version of Prim's algorithm was verified in [22] and an imperative version of Kruskal's algorithm was verified in [17]. These verifications use different frameworks than our work. For more examples and further related work see the survey [28] and [30].

2 Basic Definitions

In this section, we define the algebras that will be used in this paper. We are interested in algebras for reasoning about weighted graphs. Unweighted graphs have a straightforward representation as Boolean adjacency matrices so it makes sense that relation algebras, binary relations in particular, have been used to reason about graph algorithms [3,4,11,19,31]. Relation algebras can be generalised to Stone relation algebras to handle edge weights [13]. This is convenient since it does not involve additional structures to represent edge weights. Edge weights are typically numbered and lattices [1] provide a basis for comparing those weights.

Definition 1. *A* bounded distributive lattice, $(S, \sqcup, \sqcap, \bot, \top)$, *is a partial order,* S*, where for all* $x, y, z \in S$ *both a* join, $x \sqcup y$*, and a* meet, $x \sqcap y$ *exist and where* $x \sqcap \bot = \bot$ *and* $x \sqcup \top = \top$*, and finally where*

$$x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z) \qquad x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$$

Unfortunately, Boolean algebras cannot be used to represent edge weights as there is no suitable way to define a complement operation. The pseudo-complement operation of Stone algebras [5] weakens the complement axioms just enough to permit the inclusion of elements representing edge weights.

Definition 2. *A* Stone algebra, $(S, \sqcup, \sqcap, {}^{-}, \bot, \top)$*, is a bounded distributive lattice,* $(S, \sqcup, \sqcap, \bot, \top)$*, with a pseudo-complement operation,* ${}^{-}$*, where for all* $x, y \in S$

$$\overline{x} \sqcup \overline{\overline{x}} = \top \qquad\qquad x \sqcap y = \bot \Leftrightarrow x \leq \overline{y}$$

The pseudo-complement $\overline{y}$ is the greatest element whose meet with y is $\bot$. If $x = \overline{\overline{x}}$ then x is said to be *regular*. If a Stone algebra has only regular elements then it is a Boolean algebra.

Stone relation algebras [13] include much of the structure we require from relation algebras [23,33] but without the restrictions of the complement operation of Boolean algebras.

Definition 3. *A Stone relation algebra,* $(S, \sqcup, \sqcap, \cdot, {}^{-}, {}^{\top}, \bot, \top, 1)$*, is a Stone algebra with operations composition,* $\cdot$*, and transpose,* ${}^{\top}$*, and a constant,* 1*, where for all* $x, y, z \in S$

$$\begin{aligned} (xy)z &= x(yz) & 1x &= x \\ (x \sqcup y)z &= xz \sqcup yz & \bot x &= \bot \\ (x^{\top})^{\top} &= x & \overline{\overline{x}\,\overline{y}} &= \overline{\overline{x}}\,\overline{\overline{y}} \\ (xy)^{\top} &= y^{\top}x^{\top} & \overline{\overline{1}} &= 1 \\ (x \sqcup y)^{\top} &= x^{\top} \sqcup y^{\top} & xy \sqcap z &\leq x(y \sqcap x^{\top}z) \end{aligned}$$

Unless overridden with brackets, the operations have the precedence, from highest to lowest: ${}^{\top}$, ${}^{-}$, $\cdot$, $\sqcap$, $\sqcup$. Composition, $x \cdot y$, is often abbreviated to xy.

An element $x \in S$ is called *reflexive* if $1 \leq x$, *transitive* if $xx \leq x$, *symmetric* if $x = x^{\top}$, an *equivalence* if x is reflexive, transitive, and symmetric, a *vector* if $x\top = x$, *univalent* if $x^{\top}x \leq 1$, *injective* if $xx^{\top} \leq 1$, *surjective* if $1 \leq x^{\top}x$, *bijective* if x is injective and surjective, a *point* if x is a bijective vector and an *arc* if both $x\top$ and $x^{\top}\top$ are bijective.

For graphs, vectors are used to represent sets of vertices, points to represent a single vertex, and arcs to represent edges.

Stone-Kleene relation algebras combine Stone relation algebras with Kleene algebras to allow reasoning about reachability [13]. The unfold and induction axioms of the Kleene star are taken from [20].

Definition 4. *A* Stone-Kleene relation algebra, $(S, \sqcup, \sqcap, \cdot, {}^{-}, {}^{\top}, {}^{*}, \bot, \top, 1)$*, is a Stone relation algebra,* $(S, \sqcup, \sqcap, \cdot, {}^{-}, {}^{\top}, \bot, \top, 1)$*, with an operation,* **, where for all* $x, y, z \in S$ *the unfold and induction axioms hold*

$$\begin{aligned} 1 \sqcup xx^{*} &\leq x^{*} & z \sqcup xy &\leq x \Rightarrow zy^{*} \leq x \\ 1 \sqcup x^{*}x &\leq x^{*} & z \sqcup yx &\leq x \Rightarrow y^{*}z \leq x \end{aligned}$$

and additionally, $(\overline{\overline{x}})^{*} = \overline{\overline{x^{*}}}$.

We abbreviate xx^* as x^+. Furthermore, we call any $x \in S$ *acyclic* if $x^+ \leq \overline{1}$, and a *forest* if x is injective and acyclic.

Structure for reasoning about minimisation and aggregation is provided by m-Kleene algebras [14].

Definition 5. *An* m-Kleene algebra, $(S, \sqcup, \sqcap, \cdot, +, ^-, ^\top, ^*, s, m, \bot, \top, 1)$, *is a Stone-Kleene relation algebra,* $(S, \sqcup, \sqcap, \cdot, ^-, ^\top, ^*, \bot, \top, 1)$, *with operations addition,* $+$, *summation,* s, *and minimum selection,* m, *where for all* $x, y, z \in S$, *the summation properties are satisfied*

$$\begin{aligned} x \neq \bot \wedge s(x) \leq s(y) &\Rightarrow s(z) + s(x) \leq s(z) + s(y) \\ s(x) + s(\bot) &= s(x) \\ s(x) + s(y) &= s(x \sqcup y) + s(x \sqcap y) \\ s(x^\top) &= s(x) \end{aligned}$$

the linear property is satisfied

$$s(x) \leq s(y) \vee s(y) \leq s(x)$$

the minimum-selection properties are satisfied

$$\begin{aligned} m(x) &\leq \overline{\overline{x}} \\ x \neq \bot &\Rightarrow m(x) \text{ is an arc} \\ y \text{ is an arc} \wedge y \sqcap x \neq \bot &\Rightarrow s(m(x) \sqcap x) \leq s(y \sqcap x) \end{aligned}$$

and S *contains only finitely many regular elements.*

For reasoning about weighted graphs, we are interested in an instance of these algebras where the carrier set is comprised of square matrices whose entries are taken from the set of real numbers extended by $\bot$ and $\top$, the least and greatest elements respectively. This may be denoted as $\mathbb{R}'^{A \times A}$ where A is the index set of a square matrix and $\mathbb{R}' = \mathbb{R} \cup \{\bot, \top\}$.

In this model, an entry $\top$ in a matrix denotes an arc with unknown weight, $\bot$ the non-existence of an arc, and the real numbers arcs with weights corresponding to their values. Therefore, the regular elements (matrices over $\bot, \top$) describe the structure of graphs without weight information. The constant matrices $\bot$, $\top$ and 1 are defined as follows: $\bot_{ij} = \bot$ for all $i, j \in A$, $\top_{ij} = \top$ and

$$1_{ij} = \begin{cases} \top & \text{if } i = j, \\ \bot & \text{otherwise} \end{cases}$$

The operations $\sqcap$ and $\sqcup$ are the componentwise minimum and maximum, respectively. The binary $+$ operation on $\mathbb{R}'$ is the standard addition for real numbers and the maximum otherwise; for example, $\bot + \top = \top$. This operation is lifted to matrices, componentwise. The composition operation is defined as $(M \cdot N)_{ij} = \max_{k \in A} \min\{M_{ik}, N_{kj}\}$. The pseudo-complement on $\mathbb{R}'$ yields $\overline{x} = \bot$

for all $x \neq \bot$ and $\overline{\bot} = \top$. It is lifted componentwise to matrices. The $^\top$ operation is the usual transpose of matrices. The * operation describes reachability in a graph and is defined recursively using Conway's construction [9]:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^* = \begin{pmatrix} e^* & a^*bf^* \\ d^*ce^* & f^* \end{pmatrix} \qquad \text{where} \qquad \begin{pmatrix} e \\ f \end{pmatrix} = \begin{pmatrix} a \sqcup bd^*c \\ d \sqcup ca^*b \end{pmatrix}$$

The s-operation computes the sum of all weights in a matrix and is given by applying $+$ to all entries and storing the result in a fixed position of the returned matrix. The remaining entries in the resulting matrix are set to $\bot$. In this model, $s(\bot) = \bot$ and $\bot$ is the unit of $+$, however, there are models where neither holds [14]. The m-operation may be used for selecting an arc with minimum weight. When the input matrix contains at least one non-$\bot$ entry, the m-operation returns a matrix with $\top$ stored in the position corresponding to that of a minimum-weight arc and $\bot$ everywhere else. The result of $m(\bot)$ is $\bot$. This model is an m-Kleene algebra [13,14,15].

3 Component Selection

In graph theory there are notions of strongly-connected or weakly-connected components in a directed graph. We axiomatise an operation to select an arbitrary connected component. A component of a graph may be represented by a set of vertices as a vector.

Definition 6. *Let S be a Stone relation algebra and let $x, v \in S$. Then v represents* vector-classes *of x if x and v are regular, x is an equivalence, v is a vector, $xv \leq v$, and $v \neq \bot$. If v represents vector-classes of x and additionally, $vv^\top \leq x$ then v represents a* unique-vector-class *of x.*

A vector-class corresponds to one or more equivalence classes of x. The condition $xv \leq v$ ensures that v contains either all elements or no elements of each class. A unique-vector-class corresponds to one equivalence class. This can be used to represent the set of vertices of a particular component of a graph whose components are specified by an equivalence. The equivalence yielding the weakly-connected components of a graph, g, is obtained by taking the symmetric reflexive transitive closure, $x = (g \sqcup g^\top)^*$. For another example, the strongly-connected components of g are given by the equivalence $g^* \sqcap g^{\top *}$.

Definition 7. *A k-*Stone relation algebra*, $(S, \sqcup, \sqcap, \cdot, ^{\overline{\ }}, ^\top, k, \bot, \top, 1)$, is a Stone relation algebra, $(S, \sqcup, \sqcap, \cdot, ^{\overline{\ }}, ^\top, \bot, \top, 1)$, with a binary operation k, where for all $x, v \in S$ the element $k(x, v)$ is a regular vector and*

$$k(x, v) \leq \overline{\overline{v}} \tag{1}$$

$$k(x, v) \cdot k(x, v)^\top \leq x \tag{2}$$

$$x \cdot k(x, v) \leq k(x, v) \tag{3}$$

and, if v represents vector-classes of x then

$$k(x, v) \neq \bot \tag{4}$$

If v represents vector-classes of x the element $k(x, v)$ is a vector representing an arbitrary component that is connected according to x and contained in v. Axiom (1) expresses that the result of k is contained in the set of vertices we are selecting from, ignoring the weights. Axiom (2) makes any two vertices from the result of k connected in x. Axiom (3) expresses that the result of k is closed under being connected in x. This means that either all vertices of a component of x are included in the output of k, or none are. Axiom (4) requires that k returns a non-empty component if v represents vector-classes of x. If this is the case, the output of k represents a unique-vector-class.

Theorem 1. Let S be an m-Kleene algebra with a function k defined as

$$k(x,v) = \begin{cases} x \cdot m(v)\top & \text{if } v \text{ represents vector-classes of } x, \\ \bot & \text{otherwise} \end{cases}$$

Then S is a k-Stone relation algebra.

This particular implementation of k does not select an arbitrary component but rather a component containing a minimum-weight arc in v.

Definition 8. *An* m-k-Stone-Kleene relation algebra, $(S, \sqcup, \sqcap, \cdot, +, {}^{-}, {}^{\top}, {}^{*}, s, m, k, \bot, \top, 1)$, *is an* m-*Kleene algebra,* $(S, \sqcup, \sqcap, \cdot, +, {}^{-}, {}^{\top}, {}^{*}, s, m, \bot, \top, 1)$, *with a component selection operation,* k, *such that the reduct,* $(S, \sqcup, \sqcap, \cdot, {}^{-}, {}^{\top}, k, \bot, \top, 1)$, *is a* k-*Stone relation algebra.*

Previous work shows that matrices over $\mathbb{R}'$ form a model of m-Kleene algebras [13,14,15]. Every m-Kleene algebra is an m-k-Stone-Kleene relation algebra by Theorem 1. The correctness of Borůvka's MST algorithm will be proved in m-k-Stone-Kleene relation algebras, hence it holds in the weighted-graph model described in Sect. 2 and in many other models [14].

4 Forests Modulo an Equivalence

We generalise forests by giving axioms to treat a graph, d, as a forest modulo an equivalence, x. The arcs in d form a forest-like structure where each equivalence class of x forms a strongly-connected component. The intent of this abstraction is to provide an algebraic basis for reasoning about connectivity while forgetting about some of the structure of a graph.

4.1 Axioms and Properties

First, we define forests modulo an equivalence and give a number of properties for such structures.

Definition 9. *Let* S *be a Stone-Kleene relation algebra and let* $x, d \in S$. *Then,* d *is a* forest modulo x, *if* x *is an equivalence,* xd *is univalent, and*

$$x \sqcap dd^{\top} \leq 1 \qquad\qquad x \sqcap (xd)^{+} \leq \bot$$

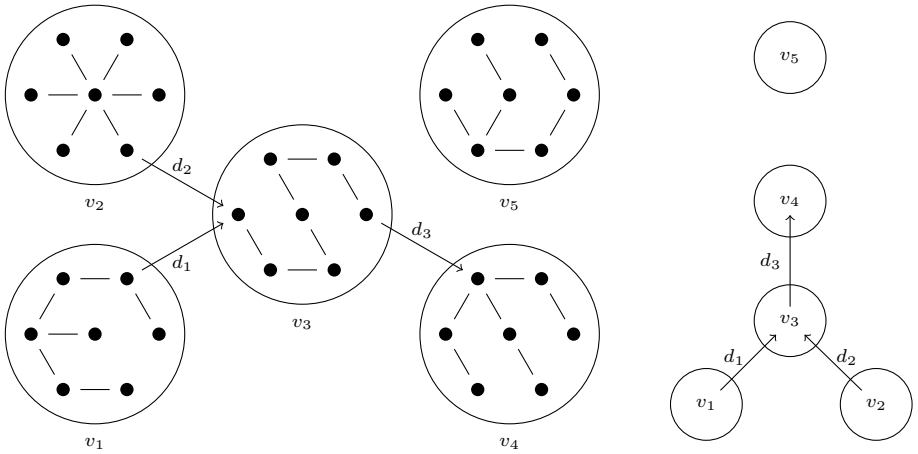

Fig. 1. An example of a forest modulo an equivalence (left) and a view of the quotient set of components (right). The equivalence classes are enclosed in circles and labelled v_1 to v_5. The arcs in d, labelled d_1 to d_3, form a forest modulo that equivalence.

These axioms describe a forest-like structure where the arcs in d are directed towards their respective root component(s). For example, the forest modulo an equivalence in Fig. 1 has two root components: v_4 and v_5. The special case of a forest modulo 1 is the transpose of a forest, that is, univalent and acyclic.

Theorem 2. Let S be a Stone-Kleene relation algebra and let $d, x \in S$ and let d be a forest modulo x. Then d is acyclic and univalent and

2.1. $d \sqcup d^\top \leq \overline{x}$
2.2. $dx \sqcup xd \leq \overline{x}$
2.3. $x(d \sqcup d^\top)x \leq \overline{x}$
2.4. $(dx)^+ \leq \overline{x}$
2.5. $(xd)^+ \leq \overline{x}$
2.6. $(xd)^* \sqcap (xd)^\top = \bot$
2.7. $d^\top xd \leq 1$
2.8. $(xd^\top)^+ xdxd^\top \leq (xd^\top)^+$
2.9. $(d^\top x)^*(xd)^* = (d^\top x)^* \sqcup (xd)^*$

Furthermore, let $a \in S$ be an arc and let $a \leq d$, then

2.10. $(d \sqcap \overline{a})^\top (xa\top) \leq \bot$
2.11. $(x(d \sqcap \overline{a}))^* xa\top = \big(x\big((d \sqcap \overline{a}) \sqcup (d \sqcap \overline{a})^\top\big)\big)^* xa\top$

Theorems 2.1 to 2.5 describe how d separates equivalence classes of x. For example, Theorem 2.4 states that taking steps $(dx)^+$ that involve one or more d-arcs leads to a different component. Theorem 2.6 follows from the acyclic-like structure of the forest modulo x. Theorem 2.7 is derivable from xd being univalent. Theorem 2.9 follows from x being an equivalence and xd being univalent. This theorem states that taking any number of steps backwards in the forest modulo x (away from the roots) followed by any number of steps forwards in the forest modulo x (towards the roots) is the same as going either forwards or backwards. Consider the situation if we take a step backwards between components of a forest modulo x without using an arc a, which is in d. Then, Theorem

2.10 states there is no sequence of steps we can take in the component we find ourselves in to next take a step along arc a.

4.2 Paths in Forests Modulo an Equivalence

Next, we define a general expression for the existence of a path between two vertices in a forest modulo an equivalence.

Definition 10. *Let S be an m-k-Stone-Kleene relation algebra and let $a, b, d, x \in S$. Then, $a \rightsquigarrow_x^d b$ holds if a and b are arcs, x is an equivalence, and*

$$a^\top \top \leq (xd)^* xb\top \tag{5}$$

Property (5) states that there is a path from the target of a, represented by the point $a^\top \top$, to the source of b, represented by the point $b\top$, in the forest d modulo x.

The following result states that for an arc e there is a path in $d \sqcup e$ between a and b if and only if there is either a path in d from a to b, or a path in d from a to e and one from e to b.

Theorem 3. Let S be an m-k-Stone-Kleene relation algebra where $a, b, e, d, x \in S$ are regular and let e be an arc and x be an equivalence. Then

$$a \rightsquigarrow_x^{d \sqcup e} b \Leftrightarrow a \rightsquigarrow_x^d b \vee \left(a \rightsquigarrow_x^d e \wedge e \rightsquigarrow_x^d b \right)$$

Theorem 3 allows us to split a path. An example use of this is given in Sect. 6.1.

5 Relational Formalisation of Borůvka's MST Algorithm

In this section, we formalise Borůvka's MST algorithm as a while-program, shown in Fig. 2. The variables of the program are elements of an m-k-Stone-Kleene relation algebra.

The input to the program is an undirected graph, g, modelled by a symmetric matrix. With the exception of g, all variables are regular elements. Graph g does not need to be connected. The algorithm constructs a directed minimum spanning forest f (of g) that is initialised as empty (line 2).

The outer while-loop executes until there are no arcs in g that could connect components of f (line 3). On lines 4 to 6, variables used by the inner while-loop are initialised. The forest h maintains a stable representation of f in each iteration of the inner while-loop. The vector j tracks the components still to be considered by the inner while-loop. The variable d tracks the arcs that have been added to f in each iteration of the inner while-loop. This variable is not required by the algorithm but is used in the correctness proof.

The inner while-loop exits when all components have been processed (line 7). As discussed in Sect. 7.1 of [15], for a directed graph, x, the weakly-connected

```
1  input g
2  f ← ⊥
3  while \overline{f^{⊤*} f^*} ⊓ g ≠ ⊥ do
4  |  d ← ⊥
5  |  h ← f
6  |  j ← ⊤
7  |  while j ≠ ⊥ do
8  |  |  v ← k(c(h), j)
9  |  |  e ← m(v\overline{v}^⊤ ⊓ g)
10 |  |  if e ≤ \overline{f^{⊤*} f^*} then
11 |  |  |  f ← f ⊓ \overline{e}^⊤
12 |  |  |  f ← (f ⊓ \overline{⊤ e f^{⊤*}}) ⊔ (f ⊓ ⊤ e f^{⊤*})^⊤ ⊔ e
13 |  |  |  d ← d ⊔ e
14 |  |  end
15 |  |  j ← j ⊓ \overline{v}
16 |  end
17 end
18 output f
```

Fig. 2. A relational formalisation of Borůvka's MST algorithm.

components are given by $(x \sqcup x^\top)^*$. Furthermore, if x is a forest, this can be simplified to $c(x) = x^{\top *} x^*$. This is because, ignoring arc direction, two vertices are connected in a forest if there is a path from one vertex backwards in x, towards a root, and then forwards in x to the other vertex. An arbitrary component, $v = k(c(h), j)$, is selected from those that have not yet been considered (line 8). The k operation introduced in Definition 7 is used. The vector j represents the components not yet processed by the inner while-loop and the forest, h, represents f as it was when the current iteration of the outer loop started. The equivalence $c(h)$ describes the weakly-connected components of f, as they were at the start of the current iteration of the outer while-loop. The component v is then weakly connected in h and among those still to be processed by the inner while-loop. Since j is reduced by v at the end of each iteration of the inner while-loop and it continues until j is empty, every component of f is processed exactly once in each iteration of the outer while-loop.

A minimum-weighted arc, e, is selected from among the arcs whose source is in v and whose target is outside v (line 9). If e is not contained in a component of f (line 10) then f is updated (lines 11 and 12), otherwise, it is not. Before e is added to f, the algorithm ensures that any transpose of e, which may have been added in a previous iteration of the inner while-loop, is removed from f (line 11). The update on line 12 adds e to f and at the same time reverses certain arcs of f to maintain that f is a forest. These two updates give a new value for f, f', that is:

$$f' = \left(f \sqcap \overline{e}^\top \sqcap \overline{\top e \left(f \sqcap \overline{e}^\top\right)^{\top *}}\right) \sqcup \left(f \sqcap \overline{e}^\top \sqcap \top e \left(f \sqcap \overline{e}^\top\right)^{\top *}\right)^\top \sqcup e \tag{6}$$

The variable d is updated to track the arcs that have been added to f in this iteration of the inner while-loop (line 13). The processed component is removed from j so that it is not considered in subsequent iterations of the inner while-loop (line 15). When the outer while-loop exits the algorithm terminates returning f (line 18) which contains the structure of a minimum spanning forest of g without weight information. The weighted version of the output may be obtained by taking the meet with g. The undirected version of the output can be obtained by taking the symmetric closure.

Borůvka's MST algorithm requires the input graph's arc weights to be distinct. Because our formalisation does not require this, we have added a check in the inner loop to ensure that no cycle is created (line 10). This check is also performed in the relational version of Kruskal's algorithm before adding an arc with minimal weight to the forest variable [15]. The relational version of Kruskal's algorithm iterates over the arcs of the graph while the inner while-loop of the algorithm we present iterates over component trees. This means that here we are often working with the properties of vectors. Both approaches keep track of the desired output by growing a minimum spanning forest, represented as a directed graph whose components are rooted directed trees. This structure is useful for maintaining that the output is injective and acyclic, properties used to conclude that the result is a forest. We select a minimum-weight arc whose source is in component v and whose target is outside v with $m(v\overline{v}^\top \sqcap g)$. This expression was used in [14] in a relational version of Prim's MST algorithm to select an arc with minimum weight that leaves a set of visited vertices.

The complexity of Equation (6) results from representing f as a directed forest, which is also the approach taken in [15]. The advantage of this approach is that it is more simple to give a specification for being acyclic than if f was undirected.

6 Correctness Proof

In this section, we discuss the partial-correctness proof of the formalisation presented in Sect. 5. We work in, and our proof holds for any instance of, m-k-Stone-Kleene relation algebras. In particular, it holds for weighted matrices, $S = \mathbb{R}'^{A\times A}$.

We reuse the specification from [15] that was used to verify Kruskal's MST algorithm.

Definition 11. *Let S be an m-Kleene algebra where $f, g \in S$. Then, f is a* spanning forest *of g if f is a regular forest and*

$$f \leq \overline{\overline{g}}$$
$$\overline{\overline{g}}^* \leq c(f)$$

The spanning forest, f, is a minimum spanning forest *of g if for all $u \in S$ where u is a spanning forest of g, the following holds:*

$$s(f \sqcap g) \leq s(u \sqcap g)$$

Intuitively, a spanning forest is a maximal acyclic subset of arcs of a graph, that is, composed of spanning trees, one for each component of the graph. A minimum spanning forest is one where the sum of arc weights is minimal among all possible spanning forests of a graph.

Our correctness proof assumes that both while-loops terminate. In future work, we may eliminate this assumption by taking a similar approach as in [15]. Presently, under the assumption that both while-loops terminate, the following theorem gives the preconditions and invariants we use to establish the postcondition of the outer while-loop that f is a minimum spanning forest of g.

Theorem 4. Let S be an m-k-Stone-Kleene relation algebra and let $g \in S$ be symmetric. Then, the following invariant holds throughout Borůvka's MST algorithm:

4.1. g is symmetric;
4.2. f is a regular forest;
4.3. $f \leq \overline{\overline{g}}$, meaning that f is contained in g, ignoring arc weights;
4.4. there is a minimum spanning forest, w, of g, such that $f \leq w \sqcup w^\top$.

Establishing Theorems 4.1 to 4.3 at the start of the algorithm is easy. The variable g is symmetric as a result of the precondition; f is a regular forest since $\bot$ is regular, injective and acyclic; and $f \leq \overline{\overline{g}}$ since $\bot$ is the least element. We reuse the proof from [15] to establish Theorem 4.4.

In contrast to Prim's and Kruskal's algorithms, Borůvka's MST algorithm has a second while-loop that we must establish and maintain an invariant for.

Theorem 5. Let S be an m-k-Stone-Kleene relation algebra and let $j \in S$. Then, the following invariant holds throughout the inner while-loop of Borůvka's MST algorithm:

5.1. $g \neq \bot$, meaning that the graph has at least one arc;
5.2. d is regular;
5.3. j is a regular vector;
5.4. h is a regular forest;
5.5. $c(h)j = j$, meaning that j contains each component of h entirely or not at all;
5.6. d is a forest modulo $c(h)$;
5.7. $d\top \leq \overline{j}$, meaning that the sources of the arcs in d are not in the set of vertices still to be processed;
5.8. $f \sqcup f^\top = h \sqcup h^\top \sqcup d \sqcup d^\top$, meaning that, ignoring direction, f can be obtained by taking the join of h and d;
5.9. $\forall a, b : a \rightsquigarrow^d_{c(h)} b \wedge a \leq \overline{c(h)} \sqcap \overline{\overline{g}} \wedge b \leq d \Rightarrow s(b \sqcap g) \leq s(a \sqcap g)$, meaning that, for any arcs a and b, if there is a path from a to b in forest d modulo $c(h)$ and a is in the graph (ignoring weight) and is not contained in the components of h and b is contained in d then the weight of b is less than or equal to the weight of a.

The requirements of the invariant for the inner while-loop are also easy to establish owing to the values that the variables are initialised to.

Most of the work to maintain the invariants is to maintain the inner while-loop invariant since, aside from variable initialisation, the logic of the inner while-loop makes up the entirety of the outer while-loop.

The following result states the correctness of the algorithm.

Theorem 6. Let S be an m-k-Stone-Kleene relation algebra and let $g \in S$ be symmetric. Then, the following postcondition is established for Borůvka's MST algorithm: f is a minimum spanning forest of g.

In the remainder of this section we give two examples of how the invariant is maintained.

6.1 Maintaining Arc Weight Comparison in a Forest Modulo $c(h)$

To show that invariant 5.9 of Theorem 5 is maintained we must show that

$$a \rightsquigarrow^{d \sqcup e}_{c(h)} b \wedge a \leq \overline{c(h)} \sqcap \overline{\overline{g}} \wedge b \leq d \sqcup e \Rightarrow s(b \sqcap g) \leq s(a \sqcap g) \tag{7}$$

for any arcs a, b. To this end, we assume that invariant 5.9 holds (for the previous iteration of the inner while-loop) and that the antecedent of (7) is true.

Our proof is by case distinctions but within each case reasoning is algebraic. There are six cases to consider and we discuss one of these in more detail. We first make a case distinction where $b \neq e$ and $e \not\leq d$. Next, we use Theorem 3 to further split into two cases. The first is $a \rightsquigarrow^{d}_{c(h)} b$, that is, there is a path from a to b in d modulo $c(h)$, in which case we can conclude $s(b \sqcap g) \leq s(a \sqcap g)$ immediately from invariant 5.9 of Theorem 5. The second is $a \rightsquigarrow^{d}_{c(h)} e$ and $e \rightsquigarrow^{d}_{c(h)} b$, that is, there is a path in d modulo $c(h)$ from a to e and one from e to b.

We first treat the path from e to b. We have $e \leq \overline{c(h)} \sqcap \overline{\overline{g}}$, since e is contained in the graph and not contained in the components of h. Additionally, $b \leq d$, since e and b are arcs and $b \neq e$ and $e \not\leq d$ and $b \leq d \sqcup e$. Together with $e \rightsquigarrow^{d}_{c(h)} b$, it follows from invariant 5.9 of Theorem 5 that $s(b \sqcap g) \leq s(e \sqcap g)$.

Next, we treat the path from a to e. Either, the target of a is in the same component as the source of e or not. If it is then we have $a^\top \top \leq c(h) e \top$ and apply the following theorem.

Theorem 7. Let S be an m-k-Stone-Kleene relation algebra and let $a, e, g, v, x \in S$ where g is symmetric, v represents a unique-vector-class of x, $e = m(v\overline{v}^\top \sqcap g)$, a is an arc, $a \leq \overline{x} \sqcap \overline{\overline{g}}$, and $a^\top \top \leq x e \top$. Then, $s(e \sqcap g) \leq s(a \sqcap g)$.

This result allows us to show that the selected arc, e, that is outgoing from a component, v, must have a weight less than or equal to any other arc incoming to that component in d with respect to the forest modulo x. This is a consequence of m selecting a minimum-weighted arc.

To apply Theorem 7, we set $x = c(h)$ and $e = m(v\overline{v}^\top \sqcap g)$ where $v = k(c(h), j)$. Then, we have that g is symmetric and a is an arc from the invariant, and $c(h)$

is an equivalence. From the axioms of the k operation and since $j \neq \bot$, we know that v represents a unique-vector-class of $c(h)$. From the antecedent we have $a \leq \overline{c(h)} \sqcap \overline{\overline{g}}$. Hence, in this case, $s(e \sqcap g) \leq s(a \sqcap g)$.

If the target of a is not in the same component of $c(h)$ as the source of e then we describe an arc, y, that is on the path from a to e in d modulo $c(h)$ and whose target is in the same component of $c(h)$ as the source of e. The arc y is defined as

$$y = d \sqcap \top e^{\top} c(h) \sqcap \left(c(h) d^{\top}\right)^{*} c(h) a^{\top} \top$$

The meet with d ensures that y is an arc between components of $c(h)$. The second part of this expression, $\top e^{\top} c(h)$, ensures that the target of y is in the same component of $c(h)$ as the source of e. The last part of this expression, $\left(c(h) d^{\top}\right)^{*} c(h) a^{\top} \top$, ensures that the source of y is reachable from the target of a by taking any number of steps in the forest d modulo $c(h)$. We can show that $s(y \sqcap g) \leq s(a \sqcap g)$ using invariant 5.9 of Theorem 5 in a similar manner as described above. Then, since we can apply Theorem 7 to show $s(e \sqcap g) \leq s(y \sqcap g)$, it follows that $s(e \sqcap g) \leq s(a \sqcap g)$.

Finally, the result for the path from a to e and the result for the path from e to b are combined to conclude that $s(b \sqcap g) \leq s(e \sqcap g) \leq s(a \sqcap g)$.

6.2 Extending f to a Minimum Spanning Forest

The key property of the invariant of the outer loop that must be maintained is that the forest, f, can be extended to a minimum spanning forest of the graph, g, ignoring arc direction, that is, there exists a minimum spanning forest, w, such that $f \leq w \sqcup w^{\top}$. We were able to reuse work from [15] in the maintenance of this invariant except the arc selected for replacement is changed.

If the arc added to f is not also in w then the added arc must replace an arc in w to ensure that w remains acyclic. In [15] the arc selected for replacement was the arc whose source was in the same component of f as the target of e. This arc does not suit our purposes because we do not have a convenient way to compare its weight with the weight of e. However, there is an easy comparison to be made between e and the arc, i, shown in Fig. 3, whose target is in the same component of f as the source of e. Namely, the weight of e is at least as small as the weight of i, since i is among those arcs that the algorithm chose e from with the minimum selection $m(v\overline{v}^{\top} \sqcap g)$.

Let $q = w \sqcap \top e w^{\top *}$, that is, the path from the root of w to the target of e and let $r = (w \sqcap \overline{q}) \sqcup q^{\top}$, that is, w with the path q reversed. Then, the desired forest, w', that extends f' is r, with i removed and e added. That is, $w' = (r \sqcap \overline{i}) \sqcup e$. The arc i is defined as

$$i = r \sqcap \overline{c(f)} e \top \sqcap \top e^{\top} c(f)$$

The meet with r limits i to only those arcs in the rooted directed forest w with the path q reversed. The second part of this expression, $\overline{c(f)} e \top$, specifies that the source of i cannot be in the same component of f as the source of e. Finally,

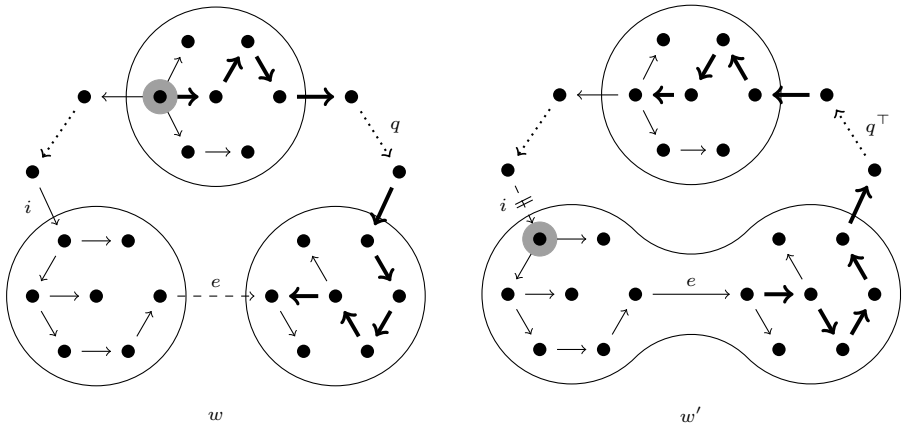

Fig. 3. Maintaining the invariant that f can be extended to a minimum spanning forest, w, before (left) and after (right) adding arc, e. The path, q, to the root of the rooted directed forest is reversed to maintain injectivity. The arc, i, whose target is in the same component of f as the source of e, is removed to maintain that w' is acyclic. The vertices enclosed in a circle denote a component, in f. The root of the rooted directed forest is highlighted grey.

the last part of the expression, $\top e^\top c(f)$, requires that the target of i is in the same component of f as the source of e.

We prove that these requirements uniquely identify an arc, i. After the update, the target of i becomes the root of w' in the component that e is in. Furthermore, we show that $s(e \sqcap g) \leq s(i \sqcap g)$ using Theorem 7.

7 Conclusion

We have formalised Borůvka's MST algorithm and proved its correctness. While we have benefited from the relation-algebraic framework introduced in [13] and the theorems subsequently developed in that framework, substantial additional proof work was required to complete our verification. To better structure the reasoning, we have axiomatised an operation, k, to select an arbitrary component. This axiomatisation uses a new definition for a component of a graph in terms of an equivalence, describing connectivity, and a vector, describing the subset of vertices we are selecting from. An implementation of k is given in m-Kleene algebras.

We have introduced a new abstraction, forests modulo an equivalence, that helps us to reason about forest-like structures of graphs by ignoring some arcs in a graph and focusing on others. The proof of our formalisation of Borůvka's MST algorithm applies this abstraction by considering the arcs connecting the components that are constructed by the inner while-loop of the algorithm as a forest modulo the components.

Much of our proof of Borůvka's MST algorithm used only the axioms of Stone-Kleene relation algebras and our proof holds for instances other than the weighted-graph model described in this paper. Different instances of m-Kleene algebras give rise to formalisations of various other algorithms, for example, the minimum bottleneck spanning tree problem [14]. Our proof holds for any instance that satisfies the axioms the proof is conducted in. This means Borůvka's MST algorithm is correct for various related MST problems.

References

1. Balbes, R., Dwinger, P.: Distributive Lattices. University of Missouri Press (1974)
2. Berge, C., Ghouila-Houri, A.: Programming, Games and Transportation Networks. Methuen, Wiley (1965)
3. Berghammer, R., von Karger, B., Wolf, A.: Relation-algebraic derivation of spanning tree algorithms. In: Jeuring, J. (ed.) Mathematics of Program Construction (MPC 1998). LNCS, vol. 1422, pp. 23–43. Springer (1998)
4. Berghammer, R., Rusinowska, A., de Swart, H.: Computing tournament solutions using relation algebra and RelView. European Journal of Operational Research **226**(3), 636–645 (2013)
5. Birkhoff, G.: Lattice Theory, Colloquium Publications, vol. XXV. American Mathematical Society, third edn. (1967)
6. Borůvka, O.: O jistém problému minimálním. Práce moravské přírodovědecké společnosti **3**(3), 37–58 (1926)
7. Chazelle, B.: A minimum spanning tree algorithm with inverse-Ackermann type complexity. Journal of the ACM **47**(6), 1028–1047 (2000)
8. Choquet, G.: Étude de certains réseaux de routes. Comptes rendus hebdomadaires des séances de l'Académie des Sciences **206**, 310–313 (1938)
9. Conway, J.H.: Regular Algebra and Finite Machines. Chapman and Hall (1971)
10. Florek, K., Łukaszewicz, J., Perkal, J., Steinhaus, H., Zubrzycki, S.: Sur la liaison et la division des points d'un ensemble fini. Colloquium Mathematicum **2**(3–4), 282–285 (1951)
11. Frias, M.F., Aguayo, N., Novak, B.: Development of graph algorithms with fork algebras. In: XIX Conferencia Latinoamericana de Informática. pp. 529–554 (1993)
12. Graham, R.L., Hell, P.: On the history of the minimum spanning tree problem. Annals of the History of Computing **7**(1), 43–57 (1985)
13. Guttmann, W.: Relation-algebraic verification of Prim's minimum spanning tree algorithm. In: Sampaio, A., Wang, F. (eds.) Theoretical Aspects of Computing (ICTAC 2016). LNCS, vol. 9965, pp. 51–68. Springer (2016)
14. Guttmann, W.: An algebraic framework for minimum spanning tree problems. Theoretical Computer Science **744**, 37–55 (2018)
15. Guttmann, W.: Verifying minimum spanning tree algorithms with Stone relation algebras. Journal of Logical and Algebraic Methods in Programming **101**, 132–150 (2018)
16. Guttmann, W., Robinson-O'Brien, N.: Relational minimum spanning tree algorithms. Archive of Formal Proofs (2020), formal proof development, `https://isa-afp.org/entries/Relational_Minimum_Spanning_Trees.html`
17. Haslbeck, M.P.L., Lammich, P., Biendarra, J.: Kruskal's algorithm for minimum spanning forest. Archive of Formal Proofs (2019), formal proof development, `https://isa-afp.org/entries/Kruskal.html`

18. Karger, D.R., Klein, P.N., Tarjan, R.E.: A randomized linear-time algorithm to find minimum spanning trees. Journal of the ACM **42**(2), 321–328 (1995)
19. Kehden, B., Neumann, F.: A relation-algebraic view on evolutionary algorithms for some graph problems. In: Gottlieb, J., Raidl, G.R. (eds.) Evolutionary Computation in Combinatorial Optimization (EvoCOP 2006). LNCS, vol. 3906, pp. 147–158. Springer (2006)
20. Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. Information and Computation **110**(2), 366–390 (1994)
21. Kruskal, Jr., J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society **7**(1), 48–50 (1956)
22. Lammich, P., Nipkow, T.: Proof Pearl: Purely Functional, Simple and Efficient Priority Search Trees and Applications to Prim and Dijkstra. In: Harrison, J., O'Leary, J., Tolmach, A. (eds.) 10th International Conference on Interactive Theorem Proving (ITP 2019). LIPIcs: Leibniz International Proceedings in Informatics, vol. 141, pp. 23:1–23:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2019)
23. Maddux, R.D.: Relation Algebras. Elsevier B.V. (2006)
24. Nešetřil, J., Milková, E., Nešetřilová, H.: Otakar Borůvka on minimum spanning tree problem – Translation of both the 1926 papers, comments, history. Discrete Mathematics **233**(1–3), 3–36 (2001)
25. Nipkow, T.: Winskel is (almost) right: Towards a mechanized semantics textbook. Formal Aspects of Computing **10**(2), 171–186 (1998)
26. Nipkow, T.: Hoare logics in Isabelle/HOL. In: Schwichtenberg, H., Steinbrüggen, R. (eds.) Proof and System-Reliability. pp. 341–367. Kluwer Academic Publishers (2002)
27. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
28. Nipkow, T., Eberl, M., Haslbeck, M.P.L.: Verified textbook algorithms. A biased survey. In: Hung, D.V., Sokolsky, O. (eds.) Automated Technology for Verification and Analysis (ATVA 2020). LNCS, vol. 12302, pp. 25–53. Springer (2020)
29. Prim, R.C.: Shortest connection networks and some generalizations. The Bell System Technical Journal **36**(6), 1389–1401 (1957)
30. Robinson-O'Brien, N.: A Formal Correctness Proof of Borůvka's Minimum Spanning Tree Algorithm. Master's thesis, University of Canterbury (2020), `https://doi.org/10.26021/10196`
31. Schmidt, G., Ströhlein, T.: Relations and Graphs. Springer (1993)
32. Tarjan, R.E.: Data Structures and Network Algorithms, CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 44. SIAM (1983)
33. Tarski, A.: On the calculus of relations. The Journal of Symbolic Logic **6**(3), 73–89 (1941)
34. Yao, A.C.C.: An $O(|E| \log \log |V|)$ algorithm for finding minimum spanning trees. Information Processing Letters **4**(1), 21–23 (1975)