A Formal Analysis of Algorithms for Matroids and Greedoids

Mohammad Abdulaziz
King's College London

Thomas Ammer
King's College London

Shriya Meenakshisundaram
King's College London

Adem Rimpapa
Technische Universität München

Abstract

We present a formal analysis, in Isabelle/HOL [30], of optimisation algorithms for matroids, which are useful generalisations of combinatorial structures that occur in optimisation, and greedoids, which are a generalisation of matroids. Although some formalisation work has been done earlier on matroids, our work here presents the first formalisation of results on greedoids, and many results we formalise in relation to matroids are also formalised for the first time in this work. We formalise the analysis of a number of optimisation algorithms for matroids and greedoids. We also derive from those algorithms executable implementations of Kruskal's algorithm for minimum spanning trees, an algorithm for maximum cardinality matching for bi-partite graphs, and Prim's algorithm for computing minimum weight spanning trees.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Theory of computation → Invariants; Theory of computation → Program verification

Keywords and phrases Matroids, Greedoids, Formal Verification, Combinatorial Optimisation, Isabelle/HOL

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

Matroids are algebraic structures that generalise the concepts of graphs and matrices, where a matroid is a pair of sets $(E, \mathcal{F})$ satisfying a number of conditions, e.g. E is a finite set and $\mathcal{F} \subseteq 2^E$. Although they have a generally rich mathematical structure which inherently justifies studying them, there are two main motivations to studying them in the context of combinatorial optimisation. First, optimisation problems defined on matroids generalise many standard combinatorial optimisation problems: e.g. the minimisation problem for matroids over modular objective functions generalises the travelling salesman problem, the shortest-path problem, minimum spanning trees, and Steiner trees, all of which are important standard optimisation problems. This means that verifying an algorithm for the minimisation problem, for instance, allows for deriving algorithms to solve many other problems (modulo some conditions that we will discuss later), and in many cases those derived algorithms are practical, albeit sometimes asymptotically slower than the fastest possible algorithms. The second reason to study matroids is that, since they are an abstraction of other concrete structures, like graphs, reasoning about (algorithms that process) them avoids a lot of the combinatorial reasoning and the case analyses encountered when reasoning about (algorithms processing) the more concrete structures.

In this paper we formalise, in Isabelle/HOL [30], a number of results about matroids, and greedoids, which are a generalisation of matroids. We formally analyse greedy algorithms

for the maximisation problems for both structures and prove guarantees on the optimality of the solutions those greedy algorithms find. We also formally analyse an algorithm for the matroid intersection problem, where one aims to find the largest member of $\mathcal{F}_1 \cap \mathcal{F}_2$, for two matroids $(E, \mathcal{F}_1)$ and $(E, \mathcal{F}_2)$. We then instantiate the verified algorithms with concrete instances to obtain executable algorithms for standard graph problems: we obtain an $O(n^2 \cdot \log^2 n)$ implementation of Kruskal's algorithm from the greedy algorithm for matroids, an $O(n \cdot m \cdot (\log n + \log m))$ algorithm for maximum cardinality bi-partite matching from the matroid intersection algorithm, and an $O(n \cdot m \log n)$ implementation of Prim's algorithm for minimum weight spanning trees from the greedy algorithm for greedoids, where n and m are the number of vertices and edges of the input graph, respectively.[1] In addition to verifying these algorithms, we formalise a number of results from the theory of matroids and greedoids, e.g. the fact that matroids can be fully characterised by the greedy algorithm, and the relationship between greedoids and antimatroids. With the exception of deriving Kruskal's algorithm from the greedy algorithm for maximisation over matroids [14], all the above results were not formalised before in any theorem prover. In our formalisation, we follow Korte and Vygen's textbook [21] and Schrijver's textbook [33], with the former being our main reference.

Availability Our formalisation is available online at `https://github.com/mabdula/Isabelle-Graph-Library`.

2 Background

Matroids A set system $(E, \mathcal{F})$ consists of a *carrier* set E and a collection of *independent sets* $\mathcal{F} \subseteq 2^E$. We only consider finite carriers. We call $(E, \mathcal{F})$ an *independence system* iff (M1) $\emptyset \in \mathcal{F}$ and (M2) $X \subseteq Y$ and $Y \in \mathcal{F}$ implies $X \in \mathcal{F}$. Members of $\mathcal{F}$ are called *independent*. Sets $F \subseteq E$ with $F \notin \mathcal{F}$ are called *dependent*. A *matroid* is an independence system satisfying the *augmentation property* (M3): for $X, Y \in \mathcal{F}$ and $|X| > |Y|$, there is $x \in X \setminus Y$ with $Y \cup \{x\} \in \mathcal{F}$.

A *basis* of a set $X \subseteq E$ is an independent subset of X maximal w.r.t. set inclusion. On the contrary, a *circuit* $C \subseteq E$ is a minimal dependent set, e.g. there are no circuits if $\mathcal{F} = 2^E$. For $X \subseteq E$, the *lower rank* $\rho(X)$ and *(upper) rank* $r(X)$ are the cardinalities of the smallest and largest bases of X, respectively. The *rank quotient* $q(E, \mathcal{F})$ is defined as the minimum of $\frac{\rho(X)}{r(X)}$ over all $X \subseteq E$. For example, if $\mathcal{F} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{2, 3\}\}$ for $E = \{1, 2, 3\}$, $\{1\}$ is a smallest and $\{2, 3\}$ is a largest basis for $X = E$ and $\frac{\rho(X)}{r(X)} = 1$ for all other $X \subseteq E$ leading to $q(E, \mathcal{F}) = \frac{1}{2}$. We have $0 < q(E, \mathcal{F}) \leq 1$ in general, and $q(E, \mathcal{F}) = 1$ iff $(E, \mathcal{F})$ is a matroid. Intuitively, the quotient expresses how close an independence system is to being a matroid.

Generalising planar graph duality (see details of instantiation to directed graphs in the next section), we call $(E, \mathcal{F}^*)$ the *dual* of an independence system $(E, \mathcal{F})$, where a set $F \in \mathcal{F}^*$ iff there is a basis B of $(E, \mathcal{F})$ s.t. $F \cap B = \emptyset$. As a basic property, we have that the dual of the dual $(E, \mathcal{F}^{**})$ is the independence system $(E, \mathcal{F})$ itself. Furthermore, $(E, \mathcal{F}^*)$ is a matroid iff $(E, \mathcal{F})$ is a matroid. For a matroid $(E, \mathcal{F})$ and $F \subseteq E$, $r^*(F) = |F| + r(E \setminus F) - r(E)$, where r^* is the rank of the dual.

Formalisation Keinholz already formalised independence systems, matroids, bases, circuits, lower ranks and basic properties of these in Isabelle/HOL [16] using *locales*. In Isabelle/HOL, locales [7] are named contexts allowing the fixing of constants and the assertion of assumptions

[1] These running times were not verified.

```
definition Frac ::"int ⇒int ⇒rat" where
  "Frac a b = (if a = b then 1 else Fract a b)"
definition rank_quotient ::"rat" where
  "rank_quotient = Min {Frac (int (lower_rank_of X)) (int (rank X)) | X. X ⊆carrier}"
```

Listing 1 Definition of the rank quotient in Isabelle/HOL.

```
locale indep_system =
  fixes carrier ::"'a set" fixes indep ::"'a set ⇒bool"
  assumes carrier_finite: "finite carrier"
  assumes indep_subset_carrier: "indep X ⟹X ⊆carrier"
  assumes indep_ex: "∃X. indep X"
  assumes indep_subset: "indep X ⟹Y ⊆X ⟹indep Y"

locale matroid = indep_system +
  assumes augment_aux:
    "indep X ⟹indep Y ⟹card X = Suc (card Y) ⟹∃x ∈X - Y. indep (insert x Y)"
```

Listing 2 Two locales with assumptions characterising independent systems and matroids.

on their properties, which will then be available within the locale for definitions and proofs. Keinholz gives locales `indep_system` and `matroid` shown in Listing 2, each of which fixes a carrier set `carrier` and an independence predicate `indep` specifying which subsets of `carrier` should be independent. We extend this formalisation with another equivalent definition of the upper rank. For the rank quotient, we explicitly define the argument of the minimum to be 1 when $\rho(X) = r(X)$ to treat division by zero and empty carriers.

Verification Methodology As mentioned earlier, a goal of this work is to formalise some of the theory of matroids and greedoids and then obtain executable algorithms to process them. We follow Nipkow's [29] approach of using locales to model *abstract data types (ADTs)* and to implement data structures such as finite sets. The functions and assumptions of an ADT can be encapsulated into a locale. In that approach, one specifies *invariants*, which give the conditions for the data structure to be well-formed, and *abstraction functions*, which convert an instance of the data structure into a value of the abstract mathematical type which it represents (see Listing 3, for instance). Implementing an algorithm defined using ADTs can be done by instantiating the ADTs with executable implementations. The approach thus allows correctness guarantees and a lot of the proofs to be done at an abstract mathematical level, while executable implementations are obtained.

To model iteration, we follow Abdulaziz's [29] approach whereby algorithms involving for- or while-loops are specified in Isabelle/HOL as recursive functions manipulating values of algorithm states, which are modelled as records with fields corresponding to the variables which change as the algorithm runs. The verification of the correctness properties of an algorithm is done using loop invariants, which are shown to hold for the initial state and to be preserved across every distinct execution path of the algorithm. With a custom-defined induction principle and by providing a termination proof for the function, we can deduce the relevant correctness theorems for the function using the invariant properties.

Our approach has limitations when it comes to the efficiency of executable code, as it cannot generate imperative code, it strikes a good balance of obtaining practical verified implementations for algorithms with deep background mathematical theory.

```
locale Card_Set2 = Set2 +
  fixes cardinality ::"'s ⇒nat"
  assumes nonempty_repr: "invar X ⟹X ≠empty ⟹set X ≠set empty"
  assumes set_cardinality: "invar X ⟹cardinality X = card (set X)"
```

Listing 3 An ADT for binary set operations: a set implementation is assumed to be of type `'s`.

```
locale Indep_System_Specs = set: Card_Set2 + ...
begin
definition indep_system_axioms where
  "indep_system_axioms carrier indep_fn =
    ((∀X. set_inv X ⟶indep_fn X ⟶subseteq X carrier) ∧
    (∃X. set_inv X ∧indep_fn X) ∧
    (∀X Y. set_inv X ⟶set_inv Y ⟶indep_fn X ⟶subseteq Y X ⟶indep_fn Y))"
...
lemma indep_system_abs_equiv:
  "indep_system_axioms carrier indep_fn =
     indep_system (carrier_abs carrier) (indep_abs indep_fn)"
```

Listing 4 A locale specifying an ADT for independence systems.

3 Greedy Algorithm for Matroids

In order to implement executable algorithms on independence systems and matroids in Isabelle/HOL, we first provide a specification of these structures that allows for executable implementations. Since the locales `indep_system` and `matroid` work with 'mathematical' sets of the type `'a set`, we specify as a locale an ADT for sets which can be abstracted to a set of type `'a set`, which would allow for executability later on after instantiation. For this, we specify an ADT using a locale extending an existing ADT (`Set2`) for binary set operations with the function `cardinality`. We then define the locale `Indep_System_Specs`, which includes the specification for `Card_Set2`. Within this locale, an independence system will be represented by a carrier set `carrier` of type `'s`, and an independence function `indep_fn` of type `'s ⇒ bool`. `indep_fn` is an *independence oracle*, that is, it takes a set `X` as input and returns a Boolean value indicating whether `X` is independent w.r.t. the matroid.

`Indep_System_Specs` defines a predicate `indep_system_axioms` which gives the independence system properties for `carrier` and `indep_fn`, in terms of the well-formedness invariant and the set operations from `Card_Set2`. We also define two abstraction functions `carrier_abs` and `indep_abs` which convert `carrier` and `indep_fn` to their abstract counterparts of type `'a set` and `'a set ⇒ bool`, respectively. We prove that if the invariants hold, then `indep_system_axioms carrier indep_fn` is equivalent to the abstract independence system properties for the abstractions of `carrier` and `indep_fn`, which allows us to reuse the theory on abstract independence systems for the implementation. We also assume an invariant that states, among other things, that the independence function must return the same value for any two implementation sets that have the same abstraction, which is non-vacuous as one abstract set can have multiple different implementations representing it. This is needed to show that the implementation and abstraction of the independence functions are equivalent. Finally, we define a locale `Matroid_Specs` which has the same specification as `Indep_System_Specs` and contains the additional, analogous predicates and lemmas for matroids.

Algorithm 1 BestInGreedy$(E, \mathcal{F}, c)$

```
1 Sort E := {e_1, ..., e_n} such that c(e_1) ≥ c(e_2) ≥ ... ≥ c(e_n);//sort the carrier set
2 F ← ∅; //initialise result
3 for i := 1 to n do
4 |   //check if F stays independent after adding e_i, add e_i if it stays independent
5 |   if F ∪ {e_i} ∈ F then F ← F ∪ {e_i};
6 return F;
```

3.1 Specification of the Best-In-Greedy Algorithm

The Best-In-Greedy algorithm is used to solve the maximisation problem on independence systems. Here, we consider an independence system $(E, \mathcal{F})$ and a nonnegative cost function $c : E \mapsto \mathbb{R}_+$, and want to find a set $X \in \mathcal{F}$ which maximises the total cost $c(X) := \sum_{e \in X} c(e)$. For the greedy algorithm, we assume the existence of an independence oracle, which given a set $F \subseteq E$, decides whether $F \in \mathcal{F}$ or not. The pseudocode of the Best-In-Greedy algorithm is shown in Algorithm 1.

In order to implement and verify the Best-In-Greedy algorithm, we define a locale `Best_In_Greedy`. It assumes that its input matroid satisfies `Matroid_Specs`. The elements of the carrier are assumed to be of type `'a`. Furthermore, the locale fixes a carrier set `carrier` of type `'s`, an independence function `indep_fn` of type `'s ⇒ bool` and a sorting function `sort_desc` of type `('a ⇒ rat) ⇒ 'a list ⇒ 'a list` which sorts the input list in descending order using the input function as a key.

We define the state for the greedy algorithm as a record type, where `carrier_list` is a list of type `'a list` consisting of the elements of the carrier set (corresponding to E) and `result` is the result set of type `'s` which is constructed over the course of the algorithm (corresponding to F). With this state type, we can specify the algorithm as a function in Isabelle/HOL as shown in Listing 4.

The recursive function `BestInGreedy` implements the loop in the pseudocode. It goes recursively through the sorted carrier list, adding elements to the constructed solution as appropriate. The algorithm requires a cost function `c` and a list `order` containing the elements of the carrier set in an initial arbitrary ordering. These two parameters are not fixed in the locale since we need to quantify over them in the subsequent correctness theorems (both universally and existentially). We require an explicit initial ordering of the elements for the proof of one of the correctness theorems. The initial state of the Best-In-Greedy algorithm is defined by setting `carrier_list` to `sort_desc c order` and `result` to the empty set.

3.2 Verification of the Best-In-Greedy Algorithm

We now describe some of the important aspects of the verification of the Best-In-Greedy algorithm in Isabelle/HOL. Several of the definitions used in the verification of the Best-In-Greedy algorithm are parametrised by `c` and `order`. Whenever these terms appear in a theorem, we assume that `c` is nonnegative and that the elements of `order` correspond exactly to those in the carrier set. Additionally, for all the correctness theorems on the greedy algorithm, we assume the predicates `BestInGreedy_axioms`, `sort_desc_axioms` and `matroid.indep_system_axioms carrier indep_fn`.

`BestInGreedy_axioms` contains the invariants from the locale `Matroid_Specs`, whereas `sort_desc_axioms` states that the function `sort_desc` sorts the input list in non-increasing

```
function BestInGreedy ::"('a, 's) best_in_greedy_state
  ⇒('a, 's) best_in_greedy_state" where
  "BestInGreedy state =
    (case (carrier_list state) of [] ⇒state
    | (x # xs) ⇒(if indep' (set_insert x (result state)) then
                   let new_result = (set_insert x (result state)) in
                     BestInGreedy (state (|carrier_list := xs, result := new_result|))
                 else BestInGreedy (state (|carrier_list := xs|))))"

definition "initial_state c order =
  (|carrier_list = (sort_desc c order), result = set_empty|)"
```

Listing 5 Best-In-Greedy algorithm in Isabelle/HOL and its initial state. Note: for a record `r`, `r (| x:= v|)` is the same as `r`, except with the value of `x` set to `v`.

order according to the input cost function, and that the sort is stable.

The first important correctness theorem for the Best-In-Greedy algorithm is Lemma 1. It states that for any nonnegative cost function c and any X which is a valid solution to the maximisation problem, the cost of the Best-In-Greedy solution is greater than or equal to the rank quotient of the independence system times the cost of solution X.

▶ **Lemma 1** (Best-In-Greedy cost bound[15, 19]). *Let $(E, \mathcal{F})$ be an independence system, with $c : E \to \mathbb{R}_+$. Let G be the output of* BestInGreedy*. Then $c(G) \geq q(E, \mathcal{F}) \cdot c(X)$ for all $X \in \mathcal{F}$.*

Proof. Let $E := \{e_1, e_2, \ldots, e_n\}$, $c : E \to \mathbb{R}_+$, and $c(e_1) \geq c(e_2) \geq \ldots \geq c(e_n)$. Let G_n be the final solution found by BestInGreedy assuming that the elements are sorted in the given order, and let X_n be an arbitrary solution. Define $E_j := \{e_1, \ldots, e_j\}$, $G_j := G_n \cap E_j$ and $X_j := X_n \cap E_j$ for $j = 0, \ldots, n$. Set $d_n := c(e_n)$ and $d_j := c(e_j) - c(e_{j+1})$ for $j = 1, \ldots, n-1$.

Since $X_j \in \mathcal{F}$, we have $|X_j| \leq r(E_j)$. Since G_j is a basis of E_j, we have $|G_j| \geq \rho(E_j)$. Together with the definition of the rank quotient, we can conclude that

$$\begin{aligned} c(G_n) &= \sum_{j=1}^{n} (|G_j| - |G_{j-1}|) c(e_j) \\ &= \sum_{j=1}^{n} |G_j| d_j \geq \sum_{j=1}^{n} \rho(E_j) d_j \geq q(E, \mathcal{F}) \sum_{j=1}^{n} r(E_j) d_j \geq q(E, \mathcal{F}) \sum_{j=1}^{n} |X_j| d_j \\ &= q(E, \mathcal{F}) \sum_{j=1}^{n} (|X_j| - |X_{j-1}|) c(e_j) = q(E, \mathcal{F}) c(X_n). \end{aligned}$$

◀

Within the appropriate context in the locale `Best_In_Greedy` (which assumes the three axiom predicates described above and that `c` and `order` are valid), the lemma can be formulated as in (Listing 6) in Isabelle/HOL. Here, `c_set c S` denotes the sum of the costs of the elements in a set `S`, where the cost function is `c`. In our formalisation, the proof of this lemma followed the informal proof, with the main argument of the proof consisting of the chain of inequalities on sums. Since we do not explicitly store the current iteration number j in the state, we provide the definition `num_iter`, which extracts the iteration number from a given state using the length of the carrier list. The definitions `carrier_pref` and `pref` are used to represent the prefix sets E_j, G_j and X_j.

```
lemma BestInGreedy_correct_2:
  "valid_solution X ⟹
    c_set c (to_set (result (BestInGreedy (initial_state c order)))) ≥
    indep_system.rank_quotient (matroid.carrier_abs carrier)
      (matroid.indep_abs indep_fn) * c_set c (to_set X)"
```

Listing 6 The first correctness lemma for the Best-In-Greedy algorithm. `t_set` is the abstraction function for the set ADT.

```
definition "invar_4 c order best_in_greedy_state =
  (∀j ∈{0..(num_iter c order best_in_greedy_state)}.
    (indep_system.basis_in (matroid.indep_abs indep_fn) (carrier_pref c order j)
                           (pref c order (to_set (result best_in_greedy_state)) j)))"
```

Listing 7 The main invariant for the Best-In-Greedy algorithm.

A main statement that is often given without further explanation in informal proofs is that for all $j \in \{1, \ldots, n\}$, G_j is a basis of E_j. Intuitively, G_j can be seen to be a maximally independent subset of E_j since it is independent by construction, and since no potential candidate elements are skipped during the algorithm. However, in the context of the formalisation, proving this statement requires some more work. In order to be able to use this fact in the proof, we formulate invariant `invar_4`, which expresses the desired property. For the invariant preservation proofs for `invar_4`, defining some more auxiliary invariants and proving they are preserved across the algorithm is necessary. These are not part of the informal proof, since they are either fairly trivial (e.g. showing that `result` is always a subset of the current carrier prefix) or aspects specific to the formal proof (e.g. showing that `result` always satisfies the set ADT well-formedness invariant). Proving that `invar_4` is preserved across the different execution paths of `BestInGreedy` then boils down to proving that if `result` is a basis of the current carrier prefix, the result set after one state update will still be a basis of the carrier prefix in the next step. We consider the two recursive cases of `BestInGreedy`, in which the current element is either added to `result` if it preserves independence, or left out otherwise. The two lemmas required for the invariant proofs in these two cases are formulated and proved in the context of the abstract `indep_system` locale. Through the theorems connecting the implementation of independence systems to the abstract theory of independence systems, we are able to use these theorems to complete the invariant preservation proofs of `invar_4`.

The second important correctness theorem on the Best-In-Greedy algorithm is Lemma 2, which states that there exists a nonnegative cost function c and a valid solution X for which the bound from Lemma 1 holds.

▶ **Lemma 2** (Best-In-Greedy cost bound tightness[15, 19]). *Let $(E, \mathcal{F})$ be an independence system. There exists a cost function $c : E \to \mathbb{R}_+$ and $X \in \mathcal{F}$ s.t. for the output G of BestInGreedy, $c(G) = q(E, \mathcal{F}) \cdot c(X)$.*

Proof. Choose $F \subseteq E$ and bases B_1, B_2 of F such that $\frac{|B_1|}{|B_2|} = q(E, \mathcal{F})$. Define $c(e) := 1$ for $e \in F$, $c(e) := 0$ for $e \in E \setminus F$ and sort $e_1, \ldots, e_n$ such that $c(e_1) \geq c(e_2) \geq \ldots \geq c(e_n)$ and $B_1 = \{e_1, \ldots, e_{|B_1|}\}$. Then $c(G) = |B_1|$ and $c(X) = |B_2|$, which finishes the proof. ◀

In our formalisation, this lemma is stated with an additional existential quantifier for the `order` parameter. The proof in our formalisation proceeds similarly to the informal proof we

```
lemma BestInGreedy_bound_tight:
  "(∃c. nonnegative c ∧(∃order. valid_order order ∧(∃X. valid_solution X ∧c_set c
    (to_set (result (BestInGreedy (initial_state c order)))) =
    indep_system.rank_quotient (matroid.carrier_abs carrier)
    (matroid.indep_abs indep_fn) * c_set c (to_set X))))"
```

Listing 8 The second correctness lemma for the Best-in-Greedy algorithm.

follow, except that we explicitly construct the initial order list and use the stability of the sorting function to show that the sorting of the elements produces the ordering necessary for the proof. Additionally, proving that the greedy result is B_1 required defining some new loop invariants and showing that they are preserved across the algorithm.

The final correctness theorem on the Best-In-Greedy algorithm concerns the relationship between the performance of the algorithm and matroids:

▶ **Theorem 3** (Characterisation of matroids, Edmonds-Rado[32, 11]). *An independence system $(E, \mathcal{F})$ is a matroid if and only if* BestInGreedy *finds an optimal solution for the maximisation problem for $(E, \mathcal{F}, c)$ for all cost functions $c : E \to \mathbb{R}_+$.*

Proof. By Lemma 1 and Lemma 2 we have $q(E, \mathcal{F}) < 1$ if and only if there exists a cost function $c : E \to \mathbb{R}_+$ for which BestInGreedy does not find an optimum solution. Using the fact that $q(E, \mathcal{F}) < 1$ if and only if $(E, \mathcal{F})$ is not a matroid completes the proof. ◀

Instantiation for Spanning Forests and Oracles For an undirected graph G with a set of edges E, a *spanning tree* T is an acyclic subgraph connecting all vertices of G to one another. A *spanning forest* is acyclic and connects all vertices that are connected via E.

The set of edges E in undirected graphs forms a matroid under acyclicity, i.e. where the sets of edges forming acyclic components is the independence set. This is a so-called *graphic matroid* and it is a widely cited example of matroids [21, 34]. Acyclicity of sets $X \subseteq E$ indeed satisfies the matroid axioms and a circuit would be a cycle in the ordinary sense. We formalise this in Isabelle/HOL, including the equivalence of being a basis w.r.t. acyclicity and being a spanning forest, making the greedy algorithm suitable to solve the minimum spanning forest problem correctly.

The instantiation of the Best-In-Greedy algorithm for graph matroids is also known as *Kruskal's Algorithm* [22]. An independence oracle could check for the absence of cycles by a modified Depth-First Search (DFS). Since the independence of the current solution X is maintained as an invariant, it is enough to check that a new element x does not lead to circuits in $X \cup \{x\}$, i.e. a new edge e does not add a cycle in the case of the graphical matroid. We call this a *weak oracle*. It would be simpler to implement and to verify, and it might even allow for running time improvements e.g. if available, by using a union-find structure to store connected components of X.

We therefore have an extended locale that also specifies the behaviour of a weak oracle, which is that if (a) X is independent and if (b) suitable data structure and auxiliary invariants are satisfied and (c) $x \notin X$, then the oracle tells correctly whether $X \cup \{x\}$ is independent. Subsequently, we verify the greedy algorithm with a weak oracle and prove it equivalent to the first version when called on the empty set. As a result, correctness can be lifted. In the end, we obtain an executable algorithm for minimum spanning forests using a simple DFS as weak oracle. A simplified version of the correctness theorem for Kruskal's Algorithm is stated in Listing 9. The obtained implementation has a running time of $\mathcal{O}(n^2 \cdot \log^2 n)$.

```
definition "max_forest X = (is_spanning_forest(t_set input_G) X ∧
    (∀ Y. is_spanning_forest (t_set input_G) Y ⟶sum c Y ≤sum c X))"

corollary kruskal_computes_max_spanning_forest:
"max_forest(t_set (result (kruskal input_G (kruskal_init c order))))"
```

Listing 9 Kruskal's algorithm's correctness (simplified).

3.3 Greedy Algorithm for Greedoids

Greedoids are a generalisation of matroids. Their definition (conditions (M1) and (M3)) is obtained by dropping the (M2) from the definition of matroids. Similar to matroids, we use a locale to define greedoids, fixing the carrier set, its family of independent subsets, and specifying the axioms of greedoids. Their intuition is derived from the edge set of undirected trees containing a fixed vertex r of an undirected graph and its total set of edges.

Our formalisation of greedoids contains accessible set systems (a set system in which every independent set contains an element removing which the resulting set continues to be independent) and antimatroids (accessible set systems closed under union) as well as the relationships between greedoids, accessible set systems and antimatroids. This is used to prove the correctness of the greedy algorithm for greedoids. Note: we formalise set systems anew because the existing formalisation of independence systems [16] uses a finite carrier set along with the independence predicate. Their formalisation was designed to be extended to matroids using the augmentation property, which would not have worked for greedoids.

We now consider the optimisation of *modular weight functions* $c : 2^E \to \mathbb{R}$ on greedoids. Modular weight functions satisfy $c(A \cup B) = c(a) + c(B) - c(A \cap B)$ for all $A, B \subseteq E$. The algorithm GreedoidGreedy keeps track of a current solution $X \in \mathcal{F}$, initially $\emptyset$. In every iteration, it searches E in the order $e_1, ..., e_n$ for an x s.t. $x \notin X$ and $X \cup \{x\} \in \mathcal{F}$ ('candidate'). It takes the first candidate in $e_1, ..., e_n$ such that $c(\{x\}) \geq c(\{y\})$ for all other candidates y ('candidate search'). If there is no candidate x, the procedure terminates. Otherwise the first best candidate is inserted into X, followed by the next iteration. Similar to Theorem 3, one can characterise certain greedoids with that algorithm.

▶ **Theorem 4** (Characterisation of Strong-Exchange Greedoids [20]). *First, we say that a greedoid $(E, \mathcal{F})$ has the strong exchange property (SEP) iff for all $A, B \in \mathcal{F}$, B basis w.r.t $\mathcal{F}$, $A \subseteq B$ and $x \in E \setminus B$ with $A \cup \{x\} \in \mathcal{F}$, there is y with $A \cup \{y\} \in \mathcal{F}$ and $(B - \{y\}) \cup \{x\} \in \mathcal{F}$. GreedoidGreedy computes a maximum-weight basis in $\mathcal{F}$ for any order of iteration $e_1, ..., e_n$ and any modular cost function $c : 2^E \to \mathbb{R}$ iff $(E, \mathcal{F})$ has the SEP.*

For the second direction ($\Leftarrow$), the usual proof is by constructing a counterexample to show the contrapositive. This heavily depends on the order $e_1, ..., e_n$ for candidate search. In Isabelle/HOL (Listing 10), both costs and the order (formalised as a list) are fixed by a context in which the algorithm is modelled, making $e_1, ..., e_n$ an input to the algorithm, as well. For the sake of brevity we cannot give further details neither on the informal nor the formal proof and the reader may refer to the formalisation instead.

Executability In Isabelle/HOL, we give an executable function that is equivalent to the one in Listing 10. However, candidate search is not performed on all $e \in E$. For the current solution X, it only checks those e where $e \in E \setminus X$ to find a best candidate.

Instantiation An *arborescence* T around a vertex r in a graph with edges E is an acyclic, connected subgraph of E that contains r. If r is in the vertices on which edges in E are incident, the set of arborescences around r forms a greedoid. Bases are spanning trees of the

```
definition "set_system E F = (finite E ∧(∀ X ∈F. X ⊆E))"

locale greedoid = fixes E ::"'a set" and F ::"'a set set"
  assumes contains_empty_set: "{} ∈F"
  assumes third_condition:
    "⋀ X Y. X ∈F ⟹Y ∈F ⟹card X > card Y ⟹∃x ∈X - Y. Y ∪{x} ∈F"
  assumes ss_assum: "set_system E F"

locale greedoid_algorithm = greedoid + (*Specification of oracles*) begin
context fixes es and c begin
definition "find_best_candidate X= foldr
  (λ e acc. if e ∈X ∨¬orcl e X then acc
            else (case acc of Some d ⇒(if c {e} > c {d} then Some e else Some d))
                              | None ⇒Some e)   es None"
function greedoid_greedy::"'a list ⇒'a list" where
  "greedoid_greedy xs = (case (find_best_candidate (set xs))
     of Some e ⇒greedoid_greedy (e#xs) | None ⇒xs)"
end

theorem greedoid_characterisation: "strong_exchange_property E F ⟷
     (∀ c es. valid_modular_weight_func E c ∧E = set es ∧distinct es
         ⟶opt_basis c (set (greedoid_greedy es c Nil)))"
```

Listing 10 Formalisation of greedoids, GreedoidGreedy, and the characterisation theorem.

connected component of r in E. The arborescence greedoid has strong exchange, making the instantiation of GreedoidGreedy, namely, the *Jarnik-Prim Algorithm* [31] optimal. We obtain a simple $\mathcal{O}(n \cdot |E| \cdot \log n)$ implementation to compute maximum weight bases where n is the number of vertices in the component around r.

4 Matroid Intersection

The maximum cardinality matroid intersection problem asks for an X of maximum cardinality in $\mathcal{F}_1 \cap \mathcal{F}_2$ for two matroids $(E, \mathcal{F}_1)$ and $(E, \mathcal{F}_2)$. A formal definition reusing the matroid locale [16] is given in Listing 11. BestInGreedy from the last section adds an element if the solution remains independent. For intersection, adding an element might preserve independence w.r.t. $\mathcal{F}_1$ but not $\mathcal{F}_2$. The insight is thus to add an element, then potentially remove an element to maintain independence w.r.t. $\mathcal{F}_2$, then insert another element. This three-step process is repeated until an element is added that preserves independence w.r.t. both $\mathcal{F}_1$ and $\mathcal{F}_2$. The repetition of this process to improve the solution is called *augmentation*.

The following is an optimality criterion for matroid intersection that involves ranks.

▶ **Lemma 5** (Rank Criterion, Edmonds' Intersection Theorem [12]). *For two matroids $(E, \mathcal{F}_1)$ and $(E, \mathcal{F}_2)$ over the same ground set E with rank functions r_1 and r_2, respectively, $X \in \mathcal{F}_1 \cap \mathcal{F}_2$, and $Q \subseteq E$ it holds that $|X| \leq r_1(Q) + r_2(E \setminus Q)$. Therefore, $|X| \leq r_1(X) + r_2(E \setminus X)$, for any $X \in \mathcal{F}_1 \cap \mathcal{F}_2$.*

Proof. Both $X \cap Q$ and $X \setminus Q$ are independent in both matroids. This implies $|X \cap Q| \leq r_1(Q)$ and $|X \setminus Q| \leq r_2(E \setminus Q)$. Both inequalities follow from the fact that $|Y| \leq r(Z)$ if $Y \subseteq Z$ and Y independent, which essentially follows from the definition of the rank r in a matroid. Of course, $|X \cap Q| + |X \setminus Q| = |X|$. ◀

```
locale double_matroid = matroid1: matroid carrier indep1
 + matroid2: matroid carrier indep2
begin
...
definition "is_max X = (indep1 X∧indep2 X∧∄Y. indep1 X∧indep2 X∧card Y > card X)"
definition "A1 =
{(x, y) | x y. y ∈carrier - X ∧x ∈matroid1.the_circuit (insert y X) - {y}}"
definition "A2 = ..."
definition "S = {y | y. y ∈carrier - X ∧indep1 (insert y X)}"
...
context assumes "indep1 X" "indep2 X"
lemma augment_in_both_matroids:
  assumes "∄ q. vwalk_bet (A1 ∪A2) x q y ∧length q < length p" "x ∈S" "y ∈T"
   "vwalk_bet (A1 ∪A2) x p y" "X' = ((X ∪{p ! i | i. i < length p ∧even i})
             -  {p ! i | i. i < length p ∧odd i})"
   shows "indep1 X'" and "indep2 X'" and "card X' = card X + 1"
...
theorem maximum_characterisation:
  "is_max X ⟷¬ (∃ p x y. x ∈S ∧y ∈T ∧(vwalk_bet (A1 ∪A2) x p y ∨x= y))"
```

Listing 11 Augmentation and Matroid Intersection.

The rank criterion cannot be exploited immediately for an algorithm since it does not have an immediate computational interpretation. It can be turned into a criterion that is computationally useful using augmentation, which we do in the next section.

4.1 Augmentation

Fix a single matroid $(E, \mathcal{F})$. For $X \in \mathcal{F}$, $x \notin X$ and $X \cup \{x\} \in \mathcal{F}$ there is a unique circuit in $X \cup \{x\}$. If there were two of these C_1, C_2, there would be a third one $C_3 \subseteq C_1 \cup C_2 \setminus \{x\} \subseteq X$, since both C_1 and C_2 need to contain x. Let $\mathcal{C}(X, x)$ be this unique circuit if $X \cup \{x\} \in \mathcal{F}$, or $\emptyset$, otherwise. We observe that if $x \in \mathcal{C}(X, y)$ and $X \not\ni y \neq x \in X$, then x can be replaced by y, i.e. $(X \setminus \{x\}) \cup \{y\}$ is independent. Under certain conditions, this replacement can be repeated as shown by the following lemma:

▶ **Lemma 6** (Replacement Lemma [13]). *For a matroid $(E, \mathcal{F})$ and $X \in \mathcal{F}$, if (1) $x_1, ..., x_s \in X$, (2) $y_1, ..., y_s \in E \setminus X$, (3) $x_k \in \mathcal{C}(X, y_k)$, for all $1 \leq k \leq s$ and (4) $x_j \notin \mathcal{C}(X, y_k)$, for all $1 \leq j < k \leq s$. Then, $(X \setminus \{x_1, ..., x_s\} \cup \{y_1, ..., y_s\}) \in \mathcal{F}$.*

Proof. We show independence of $X_l := (X \setminus \{x_1, ..., x_l\}) \cup \{y_1, ..., y_l\}$ for $l \leq s$ by induction. The theorem trivially holds for $l = 0$. We assume the claim for X_l where $0 < l < s$: It might be that $X_l \cup \{y_{l+1}\}$ is independent implying independence of X_{l+1} because of (M2). If $X_l \cup \{y_{l+1}\}$ is dependent, however, it contains a unique circuit $\mathcal{C}(X_l, y_{l+1})$ containing y_{l+1} due to X_l's independence. All deleted $x_1, ..., x_l$ cannot have been part of $\mathcal{C}(X, y_{l+1})$ because of (4) and neither any inserted $y_1, ..., y_l$ because of (2), y_{l+1} could, however. This implies $\mathcal{C}(X, y_{l+1}) \subseteq X_{l+1}$. Therefore $\mathcal{C}(X_l, y_{l+1}) = \mathcal{C}(X, y_{l+1})$. Due to (3), $x_{l+1} \in \mathcal{C}(X, y_{l+1})$, implying independence of $(X_l \setminus \{x_{l+1}\}) \cup \{y_{l+1}\} = X_{l+1}$. ◀

The formalisation of the above lemma contains an inductive proof on s within the context of the matroid locale `matroid`. The x_is and y_is are formalised as a list of pairs.

From now on, we assume two matroids $(E, \mathcal{F}_1)$ and $(E, \mathcal{F}_2)$ over the same ground set E, or formally, we work in the `double_matroid` locale (Listing 11). For a set $X \in \mathcal{F}_1 \cap \mathcal{F}_2$, we define

an auxiliary graph G_X with vertices E and edges $A_{X,1} \cup A_{X,2}$ where $A_{X,1} = \{(x, y) \,.\, y \in E \setminus X \wedge x \in \mathcal{C}_1(X, y) \setminus \{y\}\}$ and $A_{X,2} = \{(y, x) \,.\, y \in E \setminus X \wedge x \in \mathcal{C}_2(X, y) \setminus \{y\}\}$. G_X is obviously bipartite between X and $E \setminus X$. We define two sets $S_X = \{y.\ y \in E \setminus X \wedge X \cup \{y\} \in \mathcal{F}_1\}$ and $T_X = \{y.\ y \in E \setminus X \wedge X \cup \{y\} \in \mathcal{F}_2\}$. A path between a vertex from S_X and another from T_X indicates an alternating sequence of insertion and deletion. Due to bipartiteness, the length will be odd allowing for an augmentation. We make this precise as follows.

▶ **Lemma 7** (Augmentation Lemma). *Let $X \in \mathcal{F}_1 \cap \mathcal{F}_2$ be independent in both matroids and $p = y_0x_1y_1x_2y_2...x_sy_s$ a shortest path from $y_0 \in S_X$ to $y_s \in T_X$. Then $X' = (X \cup \{y_0, ..., y_s\}) \setminus \{x_1, ..., x_s\}$ is independent in both matroids, i.e. $X' \in \mathcal{F}_1 \cap \mathcal{F}_2$.*

Proof. We apply Lemma 6 to $X \cup \{y_0\}$, $x_1, ..., x_s$ and $y_1, ..., y_s$ to show $X' \in \mathcal{F}_1$: For all $0 < i \leq s$, x_i is the second vertex of the $(2i-1)$th edge of p. Because the path alternates between $E \setminus X$ and X, the $(2i-1)$th is in $A_{X,2}$ and $x_i \in X \cup \{y_0\}$ for all x_i, yielding (1). For all $0 < i \leq s$, y_i is the second vertex of the $(2i)$th edge in p. Because of alternation and pairwise distinctness of the ys (p is shortest path), $y_i \in E \setminus (X \cup \{y_0\})$, implying (2). Also, for all $0 < i \leq s$, x_i is the first vertex of the $(2i)$th edge, which is part of $A_{X,1}$ because of alternation. Therefore, $x_k \in \mathcal{C}_1(X \cup \{y_0\}, y_k)$ for all $1 \leq k \leq s$ (3). We assume an x_i and x_j where $0 < j < i \leq s$ and $x_j \in \mathcal{C}_1(X \cup \{y_0\}, y_i)$. x_i or x_j is the first vertex of the $2i$th of $2j$th edge of p, respectively. y_j and y_i are the second vertices. Both of those edges are part of $A_{X,1}$. As $y_i \in E \setminus X$, and therefore $x_j \neq y_i$, the edge $(x_j, y_i) \in A_{X,1}$. We could then delete $y_j, ..., x_{i-1}$ giving a shorter S_X-T_X-path. Hence (4) is satisfied and Lemma 6 can be applied.

Analogously, we take $X \cup \{y_s\}$, $x_s, ..., x_1$ and $y_{s-1}, ..., y_0$ to show $X' \in \mathcal{F}_2$. ◀

The pairs of xs and ys are formalised as a zipped list. As indexing starts at 0 in Isabelle/HOL, the indices are often off by 1. The statement is in Listing 11.

On top of improvement by augmentation, the absence of an augmenting path characterises maximality of $|X|$ for $X \in \mathcal{F}_1 \cap \mathcal{F}_2$:

▶ **Theorem 8** (Optimality Criterion). *X is a set of maximum cardinality in $\mathcal{F}_1 \cap \mathcal{F}_2$ iff the G_X does not contain a path from some $s \in S_X$ to some $t \in T_X$ (henceforth, an $S_X - T_X$path). Edmonds' max-min-equality [12] is a corollary:* $\max\{|X|.\ X \in \mathcal{F}_1 \cap \mathcal{F}_2\} = \min\{r_1(Q) + r_2(E \setminus Q).\ Q \subseteq E\}$.

Proof. If there is an S_X-T_X path, there is a shortest one as well, which could be used for an augmentation according to Lemma 7, leading to an increase in the cardinality.

Now assume, that there is no S_X-T_X path. We define R as the set of vertices in G_X that are reachable from S_X. Obviously $S_X \subseteq R$ and $R \cap T_X = \emptyset$. There are rank functions r_1 and r_2 w.r.t. $\mathcal{F}_1$ and $\mathcal{F}_2$, respectively.

We prove $r_2(R) = |X \cap R|$ by contradiction: Since $X \cap R$ has to be independent w.r.t. the second matroid and therefore $|X \cap R| = r_2(X \cap R)$, we would have $r_2(X \cap R) < r_2(R)$ since r_2 is monotone. Because of the strict inequality, there is $y \in R \setminus X$ where $(X \cap R) \cup \{y\} \in \mathcal{F}_2$. Otherwise, $X \cap R$ would be a basis of R implying equal ranks. As $R \cap T_X = \emptyset$, $X \cup \{y\} \notin \mathcal{F}_2$. There is $x \in X \setminus R$ with $x \in \mathcal{C}_2(X, y)$ (Otherwise $(X \cap R) \cup \{y\} \notin \mathcal{F}_2$). By definition, $(y, x) \in A_{X,2}$. That makes x part of R, which is a contradiction.

We prove $r_1(E \setminus X) = |X \setminus R|$ by contradiction: Since $X \setminus R$ has to be independent w.r.t. the first matroid and therefore $|X \setminus R| = r_1(X \setminus R)$, we would have $r_1(X \setminus R) < r_1(E \setminus R)$ since r_1 is monotone. Because of the strict inequality, there is $y \in (E \setminus R) \setminus X$ where $(X \setminus R) \cup \{y\} \in \mathcal{F}_1$. Otherwise, $X \setminus R$ would be a basis of $E \setminus R$ implying equal ranks. As $S_X \subseteq R$, $X \cup \{y\} \notin \mathcal{F}_1$. There is $x \in X \cap R$ with $x \in \mathcal{C}_1(X, y)$ (Otherwise $(X \setminus R) \cup \{y\} \notin \mathcal{F}_1$). By definition, $(x, y) \in A_{X,1}$. That makes y part of R, which is a contradiction.

Therefore, $|X| = r_1(E \setminus X) + r_2(X)$. Because of the rank criterion from Lemma 5, X satisfies $|X| \leq r_1(Q) + r_2(E \setminus Q)$ with equality. This gives the max-min equality and makes X a set of maximum cardinality that is independent w.r.t. both matroids. ◀

The formal proof of Theorem 8 is split in two directions. The second direction also shows $r_2(R) = |X \cap R|$, $r_1(E \setminus X) = |X \setminus R|$ and $|X| = r_1(E \setminus X) + r_2(X)$ as statements to prove the max-min equality separately. The final formalisation can be seen in Listing 11.

4.2 Intersection Algorithm

We exploit the previous subsection's results for an algorithm [6, 24] that iteratively applies augmenting paths to solve maximum cardinality matroid intersection. The main invariant of the algorithm MaxMatroidIntersection is $X \in \mathcal{F}_1 \cap \mathcal{F}_2$ for the current solution X. Algorithm 2 contains the pseudocode. In our main reference [21], the circuits $\mathcal{C}_{1/2}(X, y)$, which are minimal dependent sets, are computed as explicit sets by iterating over all $x \in X \cup \{y\}$ and taking those where $X \cup \{y\} \setminus \{x\}$ is independent. G_X is then computed by using the definition from above which requires further iterations over the circuits. Our pseudocode does not need the circuits as an intermediate step and adds an edge (x, y) or (y, x) if $X \cup \{y\}$ is dependent or $X \setminus \{x\} \cup \{y\}$ is independent, respectively.

Algorithm 2 MaxMatroidIntersection(E, $\mathcal{F}_1$, $\mathcal{F}_2$)

```
1  Initialise X ← ∅;
2  while True do
3      compute G_X: Initialise S_X ← ∅; T_X ← ∅; A_{X,1} ← ∅; A_{X,2} ← ∅;
4      for y ∈ E \ X do
5          if X ∪ {y} ∈ F_1 then S_X ← S_X ∪ {y};
6          else for x ∈ X do [ if X \ {x} ∪ {y} ∈ F_1 then A_{X,1} ← A_{X,1} ∪ {(x, y)};]
7          if X ∪ {y} ∈ F_2 then T_X ← T_X ∪ {y};
8          else for x ∈ X do [ if X \ {x} ∪ {y} ∈ F_2 then A_{X,2} ← A_{X,2} ∪ {(y, x)}; ]
9      if ∃ path leading from S_X to T_X via the edges in A_{X,1} ∪ A_{X,2} then
10         find a shortest path P = y_0 x_1 y_1 ... x_s y_s leading from S_X to T_X;
11         augment along P: X ← X ∪ {y_0, ..., y_s} \ {x_1, ..., x_s};
12     else return X as maximum cardinality set in F_1 ∩ F_2;
```

MaxMatroidIntersection is again formalised within a locale that assumes the necessary subprocedures, most notably a function `find_path` for path selection. This will take the graph G_X as an adjacency map. We also have two weak independence oracles `orcl1` and `orcl2`, one for each matroid. We assume a Haskell-style `outer_fold` and `inner_fold` to simulate for-loops over sets in the functional language of Isabelle/HOL. We furthermore have a function to complement sets for the term $E \setminus X$.

Listing 12 shows the formalisation of the for-loops. `treat1` and `treat2` simulate the for-loops in Line 6 and Line 8, respectively. The outer for-loop extending from Lines 4 to 8 is realised by `compute_graph`. We also have a simple function `augment` performing the augmentation, i.e. alternating deletion and insertion to a set. The function `matroid_intersection` (while loop) calls the graph computation, gives $A_{X,1} \cup A_{X,2}$, S_X and T_X to the path selection function and terminates or calls augmentation according to the result of path selection. It maps the solution X before entering the loop to the X after finishing the loop. We use a record with a single variable for the current solution.

```
definition "treat1 y X init_map = inner_fold X (λ x edges.
   if weak_orcl1 y (set_delete x X) then add_edge edges x y else edges) init_map"
definition "treat2 y X init_map = ..."
definition "compute_graph X E_without_X =
 outer_fold E_without_X (λ y (SX, TX, edges). (
  let (SX, TX, edges) = (if weak_orcl1 y X then (insert y SX, TX, edges)
         else (SX, TX, treat1 y X edges));
      (SX, TX, edges) = (if weak_orcl2 y X then (SX, insert y TX, edges)
         else (SX, TX, treat2 y X edges))
  in (SX, TX, edges))) (vset_empty, vset_empty, empty)"

fun augment where ...

function (domintros) matroid_intersection::"'mset intersec_state⇒ 'mset
    intersec_state" where
  "matroid_intersection state =
 (let X = sol state; (SX, TX, graph) = compute_graph X (complement X) in
    (case find_path SX TX graph of None ⇒state |
     Some p ⇒matroid_intersection (state (|sol := augment X p|)) ))"

definition "initial_state = (|sol= set_empty |)"
```

Listing 12 Formalisation of the Intersection Algorithm.

Again, we use invariants for loop verification. We have independence of X as the major invariant and data structure well-formedness invariants as minor ones. As in the case of BestInGreedy, we use the approach of proving single-step preservation for execution paths that is combined by an induction. The first lemma in Listing 13 says that by executing one step (a) X remains in $\mathcal{F}_1 \cap \mathcal{F}_2$, (b) X continues to satisfy its data structure invariant, (c) $X \subseteq E$, (d) X's cardinality increases by 1 and (e) neither S_X nor T_X are empty. The Listing also contains the lemma certifying optimality if the terminating branch of `matroid_intersection` is taken. The second lemma together with (a), (b) and (c) can be combined by the induction principle into correctness for general inputs X. When (d) and (e) are used, there is termination. Finally, one can prove total correctness when `matroid_intersection` is called on `initial_state`.

Running Time and Circuit Oracles The number of iterations to build G_X is $\mathcal{O}(|E|^2)$. Each matroid comes with an oracle which is assumed to be ι_1 or ι_2, respectively. This results in $\mathcal{O}(|E|^3 \cdot (\iota_1 + \iota_2))$ overall running time. We might have to check $|X| \cdot (|E| - |X|)$ pairs of vertices. If - depending on the problem - the size of the circuits σ_1 and σ_2 is assumed small, e.g. $\mathcal{O}(\log |E|)$, most of the inner for-loops' iterations were wasted. If the circuits can be computed in time $\kappa_1, \kappa_2 \in o(|E|)$, the time would be the more efficient $\mathcal{O}(|E|^2 \cdot (\iota_1 + \iota_2 + \kappa_1 + \kappa_2 + \sigma_1 + \sigma_2))$.

In Isabelle/HOL, weak circuit oracles are assumed to return data structures storing $\mathcal{C}_{1/2}(X, y) \setminus \{y\}$ provided that X is independent and $X \cup \{y\}$ is dependent. These are added to the locale for MaxMatroidIntersection, as well. A variant of the algorithm using circuit oracles is verified using the same methodology as before.

Selecting an S_X-T_X Path There is a verified implementation of Breadth-First Search (BFS) that works on adjacency maps by Abdulaziz [29]. It takes a set of sources and builds another adjacency map modelling a DAG of those edges that are actually explored by the search. If a vertex v is reachable from a source s, there is a path p in this DAG leading from a source

```
definition "indep_invar state =
   (indep1 (to_set (sol state)) ∧(indep2 (to_set (sol state))))"

lemma indep_invar_recurse_improvement:
  assumes "matroid_intersection_recurse_cond state" "indep_invar state"
          "set_invar (sol state)" "to_set (sol state) ⊆carrier"
  shows   "indep_invar (matroid_intersection_recurse_upd state)"
          "set_invar (sol (matroid_intersection_recurse_upd state))"
          "to_set (sol (matroid_intersection_recurse_upd state)) ⊆carrier"
          "card (to_set (sol (matroid_intersection_recurse_upd state)) =
             card (to_set (sol state)) + 1"
          "S (to_set (sol state)) ≠{}" "T (to_set (sol state)) ≠{}"

lemma indep_invar_max_found:
  assumes "matroid_intersection_terminates_cond state" "indep_invar state"...
  shows   "is_max (to_set (sol state))"

lemma matroid_intersection_correctness_general:
  assumes "indep_invar state" "set_invar (sol state)" "to_set (sol state) ⊆carrier"
    shows "is_max (to_set (sol (matroid_intersection state)))"

lemma matroid_intersection_terminates_general:
  assumes "indep_invar state" "set_invar (sol state)" "to_set (sol state) ⊆carrier"
          "m = card carrier - card (to_set (sol state))"
  shows   "matroid_intersection_dom state"

lemma matroid_intersection_total_correctness:
     "is_max (to_set (sol (matroid_intersection initial_state)))"
     "matroid_intersection_dom initial_state"
```

Listing 13 Invariants, Termination and Total Correctness of the Intersection Algorithm. Note that `f_dom input` for a function `f` means that `f` terminates on `input`.

s' to v such that p is a shortest path in the actual graph leading from s' to v. `find_path` runs BFS for the sources S_X and uses a DFS to obtain a path in the DAG and, hence, a shortest S_X-T_X path. For a concrete intersection problem and together with appropriate oracles, `find_path` can be used to instantiate the locale of the intersection algorithm.

Bipartite Matching An example for matroid intersection is *maximum cardinality bipartite matching*. Consider a bipartite graph with edges E between two disjoint sets of vertices L and R. For $e \in E$, we call the endpoint in L the *left* and the one in R the *right endpoint*. We say that $M \subseteq E$ is independent w.r.t. L iff no edges in M share a left endpoint. Independence w.r.t. R is defined analogously. Valid *matchings* are sets of vertex-disjoint edges in E and are thus exactly those M that are independent w.r.t. both L and R. The two independence predicates satisfy the matroid axioms making this a matroid intersection problem. Circuits w.r.t. L or R would be edges $e, d \in E$ sharing a left or right endpoint, respectively.

For $M \subseteq E$, we maintain a set data structure $[M]$, two maps M_L and M_R, where M_L associates every $x \in L$ with $e \in M$ for which x is the left endpoint, and analogously for M_R. Weak independence and circuit oracles are very simple: If M is independent w.r.t. L (R), and $e \notin M$, we check in M_L (M_R) that e's left (right) endpoint is not associated to any another edge. If $M \cup \{e\}$ would be dependent w.r.t. L or R, we would return the edges associated to e's left or right endpoint as $\mathcal{C}_L(M, e) \setminus \{e\}$ or $\mathcal{C}_R(M, e) \setminus \{e\}$, respectively. We used these oracles to

instantiate MaxMatroidIntersection to compute maximum matchings in bipartite graphs. The running time is $\mathcal{O}(\min\{|L|, |R|\} \cdot m \cdot (\log |R| + \log |L| + \log m))$ where the logarithms are due to using tree data structures, which is dominated by $\mathcal{O}(n \cdot m \cdot (\log n + \log m))$. Without circuit oracles, we would have another multiplicative factor of $\min\{|L|, |R|\}$.

5 Discussion

We presented a formal analysis of algorithms to solve two optimisation problems for matroids: (a) computing the largest (in terms of accumulated weight) independent set in a given matroid and (b) computing the largest (in terms of cardinality) set that is independent w.r.t. two matroids. The two algorithms are of great importance to practitioners and are implemented in a number of computer algebra systems, e.g. in Sage [1] and Macaulay [2]. Additionally, we also briefly (due to space constraints) presented a formalisation of the analysis of an algorithm to solve an optimisation problem for greedoids, which are a generalisation of matroids. In addition to formalising significant parts of the theory of matroids, greedoids, and the analysis of the algorithms, we showed that our approach can also be used to obtain practical executable verified implementations of important graph algorithms. This demonstrates the potential practical role of matroids and greedoids to design well-factored formal mathematical libraries whereby proofs of multiple algorithms are done in one go that captures their mathematical essence, and that is later instantiated for different concrete examples. The formalisation overall totalled 17.4K lines of proof script, with 11K lines dedicated to the theory of matroids and greedoids, 2.9K lines connecting graph theoretic problems to matroids and/or greedoids, and 3.5K lines on defining and verifying Kruskal's, Prim's, and the bi-partite matching algorithm. Our formalisation builds on and is part of an ongoing effort to formalise combinatorial optimisation in Isabelle/HOL[4].

Related work Most relevant to us is the formalisation of matroids in Isabelle/HOL by Keinholz [16]. Haslbeck et al. [14] formalised a classic proof of the correctness of the Best-In-Greedy Algorithm and obtained a verified, imperative implementation of Kruskal's algorithm. They used a weak oracle based on a verified imperative union-find implementation by Löwenberg [25] to store the tree's connected components leading to their implementation having the best possible running time of $\mathcal{O}(m\alpha(n))$, where α is the inverse Ackermann function. In Lean, there is an ongoing effort to formalise matroid theory [28], including more advanced results about e.g. minors, isomorphism, Tutte's excluded minor theorem for finitary binary matroids, or Edmonds' Intersection Theorem (Lemma 5), but (as of the time of writing) there seems to be no verified practical algorithms in that library. Matroids were also formalised in Coq [26, 9] and Mizar [8], however, both do not go very far. Set systems were formalised in Isabelle/HOL by Edmonds and Paulson [10], who did that in the context of verifying combinatorial designs. Additionally, there are formalisations of matching algorithms [3, 5] and Prim's algorithm [23], in Isabelle/HOL, and of Dijkstra, Kruskal, and Prim [27], in Coq, that are not based on matroids or greedoids, where all reasoning is done at the graph-theoretic level. Readers interested in comparing the style of formal reasoning when using matroids or greedoids, which is algebraic, to the one performed directly on graphs, which is more intuitive but combinatorial, should refer to those formalisations.

Future work There is one 'main' missing algorithm from our library which would be the most imminent piece of future work: the weighted version of the matroid intersection problem, whereby, instead of finding the largest cardinality set that is independent w.r.t. two matroids, one is to find the largest in terms of the weight of such a set, for a modular cost function.

Routine functions that check whether a matroid indeed satisfies the assumptions of a

matroid, compute the dual of a matroid, etc., which are implemented in most computer algebra packages, are missing from our library and are an important next step for our work. Naive implementations of those functions would have a worst-case exponential running time, so utilising sophisticated enumeration algorithms [17, 18] would help with making that more practical.

References

1 The abstract Matroid class - Matroid Theory. URL: https://doc.sagemath.org/html/en/reference/matroids/sage/matroids/matroid.html.

2 maxWeightBasis – maximum weight basis using greedy algorithm. URL: https://macaulay2.com/doc/Macaulay2/share/doc/Macaulay2/Matroids/html/_max__Weight__Basis.html.

3 Mohammad Abdulaziz. A Formal Correctness Proof of Edmonds' Blossom Shrinking Algorithm, 2024. arXiv:2412.20878, doi:10.48550/arXiv.2412.20878.

4 Mohammad Abdulaziz. https://github.com/mabdula/Isabelle-Graph-Library, 2024. URL: https://github.com/mabdula/Isabelle-Graph-Library.

5 Mohammad Abdulaziz and Christoph Madlener. A Formal Analysis of RANKING. In *The 14th Conference on Interactive Theorem Proving (ITP)*, 2023. doi:10.48550/arXiv.2302.13747.

6 Martin Aigner and Thomas A. Dowling. Matching theory for combinatorial geometries. *Transactions of the American Mathematical Society*, 158(1):231–245, 1971. URL: http://www.jstor.org/stable/1995784.

7 Clemens Ballarin. Locales: A Module System for Mathematical Theories. *J. Autom. Reason.*, (2):123–153, 2014. doi:10.1007/s10817-013-9284-7.

8 Grzegorz Bancerek and Yasunari Shidama. Introduction to matroids. 2008. URL: https://api.semanticscholar.org/CorpusID:18192408.

9 David Braun, Nicolas Magaud, and Pascal Schreck. A matroid-based automatic prover and coq proof generator for projective incidence geometry. *Journal of Automated Reasoning*, 68(1):3, Jan 2024. doi:10.1007/s10817-023-09690-2.

10 Chelsea Edmonds and Lawrence C. Paulson. A Modular First Formalisation of Combinatorial Design Theory. In *Intelligent Computer Mathematics*, pages 3–18, 2021. doi:10.1007/978-3-030-81097-9_1.

11 Jack Edmonds. Matroids and the greedy algorithm. *Math. Program.*, 1(1):127–136, December 1971. doi:10.1007/BF01584082.

12 Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In *Combinatorial Optimization*, 2001. URL: https://api.semanticscholar.org/CorpusID:14909675.

13 András Frank. A weighted matroid intersection algorithm. *Journal of Algorithms*, 2(4):328–336, 1981. URL: https://www.sciencedirect.com/science/article/pii/0196677481900328, doi:10.1016/0196-6774(81)90032-8.

14 Maximilian P. L. Haslbeck, Peter Lammich, and Julian Biendarra. Kruskal's algorithm for minimum spanning forest. *Archive of Formal Proofs*, February 2019. https://isa-afp.org/entries/Kruskal.html, Formal proof development.

15 T. A. Jenkyns. The efficacy of the "greedy" algorithm. In *Proceedings of the 7th Southeastern Conference on Combinatorics, Graph Theory and Computing*, pages 341–350, 1976. URL: https://cir.nii.ac.jp/crid/1570572699713216768.

16 Jonas Keinholz. Matroids. *Archive of Formal Proofs*, November 2018. https://isa-afp.org/entries/Matroids.html, Formal proof development.

17 Donald E. Knuth. *The Art of Computer Programming: Combinatorial Algorithms, Part 1*. Addison-Wesley Professional, 1st edition, 2011.

18 Donald E. Knuth. *The Art of Computer Programming: Combinatorial Algorithms; Vol.4B - Part 2*. Pearson –, United States of America –, 1st ed. edition, 2022.

19 Bernhard Korte and Dirk Hausmann. An analysis of the greedy heuristic for independence systems. *Annals of discrete mathematics*, 2:65–74, 1978. URL: https://api.semanticscholar.org/CorpusID:62882671.

20 Bernhard Korte and László Lovász. Greedoids and linear objective functions. *SIAM J. Algebraic Discrete Methods*, 5(2):229–238, June 1984. doi:10.1137/0605024.

21 Bernhard Korte and Jens Vygen. *Combinatorial Optimization*. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-24488-9.

22 Joseph B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956. URL: http://www.jstor.org/stable/2033241.

23 Peter Lammich and Tobias Nipkow. Proof Pearl: Purely Functional, Simple and Efficient Priority Search Trees and Applications to Prim and Dijkstra. In *10th International Conference on Interactive Theorem Proving, ITP 2019, September 9-12, 2019, Portland, OR, USA*, pages 23:1–23:18, 2019. doi:10.4230/LIPICS.ITP.2019.23.

24 Eugene L. Lawler. Matroid intersection algorithms. *Mathematical Programming*, 9(1):31–56, Dec 1975. doi:10.1007/BF01681329.

25 Adrian Löwenberg and Maximilian Haslbeck. Bachelor thesis: Proof of the amortized time complexity of an efficient implementation of the union-find data structure in isabelle/hol. URL: https://github.com/adrilow/Proof-of-the-amortized-time-complexity-of-the-Union-Find-data-structure-in-Isabelle-HOL/tree/master.

26 Nicolas Magaud, Julien Narboux, and Pascal Schreck. A case study in formalizing projective geometry in coq: Desargues theorem. *Computational Geometry*, 45(8):406–424, 2012. Geometric Constraints and Reasoning. URL: https://www.sciencedirect.com/science/article/pii/S0925772112000247, doi:10.1016/j.comgeo.2010.06.004.

27 Anshuman Mohan, Wei Xiang Leow, and Aquinas Hobor. Functional Correctness of C Implementations of Dijkstra's, Kruskal's, and Prim's Algorithms. In *Computer Aided Verification*, pages 801–826, 2021. doi:10.1007/978-3-030-81688-9_37.

28 Peter Nelson, Mattias Ehatamm, Tom Iagovet, fair8 (pseudonym), Elan2004 (pseudonym), Max Jiang, and Kim Morrison. Matroids. URL: https://github.com/apnelson1/Matroid.

29 Tobias Nipkow, Mohammad Abdulaziz, Jasmin Blanchette, Manuel Eberl, Alejandro Gomez-Londono, Peter Lammich, Lawrence Paulson, Christian Sternagel, Simon Wimmer, and Bohua Zhan. Functional data structures and algorithms. A proof assistant approach, 2025.

30 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*. Springer, 2002.

31 R. C. Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957. doi:10.1002/j.1538-7305.1957.tb01515.x.

32 R. Rado. Note on independence functions. *Proceedings of the London Mathematical Society*, s3-7(1):300–320, 01 1957. doi:10.1112/plms/s3-7.1.300.

33 A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Number 24. Springer, 2003.

34 A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. 2003.