A Formal Analysis of Algorithms for Matroids and Greedoids

Mohammad Abdulaziz[1], Thomas Ammer[1],
Shriya Meenakshisundaram[1], Adem Rimpapa[2]

[1]King's College London (KCL)
[2]Technical University of Munich (TUM)

December 2, 2025

A Formal Analysis of Algorithms for Matroids and Greedoids

Mohammad Abdulaziz
King's College London, UK

Thomas Ammer
King's College London, UK

Shriya Meenakshisundaram
King's College London, UK

Adem Rimpapa
Technische Universität München, Germany

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

Table of Contents

Background

Background

- combinatorial optimisation: optimisation problems on discrete structures, e.g. graphs

CO Example: Minimum Spanning Tree

CO Example: Minimum Spanning Tree

- undirected (multi-)graph with edges E and costs $c : E \to \mathbb{R}^+$

CO Example: Minimum Spanning Tree

- undirected (multi-)graph with edges E and costs $c : E \to \mathbb{R}^+$
- forest = acyclic subgraph

CO Example: Minimum Spanning Tree

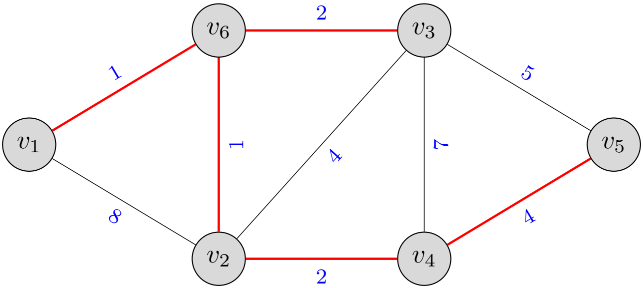
- undirected (multi-)graph with edges E and costs $c : E \to \mathbb{R}^+$
- forest = acyclic subgraph
- tree = forest with a single component

CO Example: Minimum Spanning Tree

- undirected (multi-)graph with edges E and costs $c : E \to \mathbb{R}^+$
- forest = acyclic subgraph
- tree = forest with a single component
- spanning tree minimising/forest maximising accumulated costs

CO Example: Minimum Spanning Tree

- undirected (multi-)graph with edges E and costs $c : E \to \mathbb{R}^+$
- forest = acyclic subgraph
- tree = forest with a single component
- spanning tree minimising/forest maximising accumulated costs



CO Example: Matching

CO Example: Matching

- given an undirected graph

CO Example: Matching

- given an undirected graph
- find a set of vertex-disjoint edges

CO Example: Matching

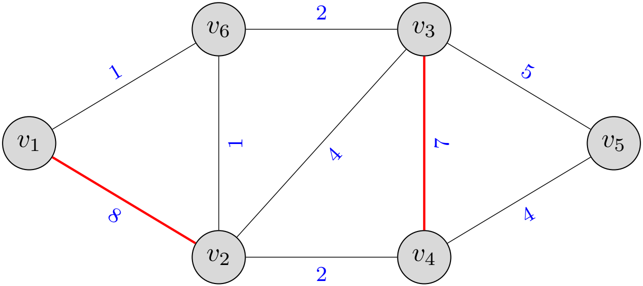
- given an undirected graph
- find a set of vertex-disjoint edges
- while aiming at an optimisation objective

CO Example: Matching

- given an undirected graph
- find a set of vertex-disjoint edges
- while aiming at an optimisation objective
- e.g. mere cardinality, or accumulated costs

CO Example: Matching

- given an undirected graph
- find a set of vertex-disjoint edges
- while aiming at an optimisation objective
- e.g. mere cardinality, or accumulated costs



Other CO Examples

Other CO Examples

- shortest paths

Other CO Examples

- shortest paths
- maximum flow

Other CO Examples

- shortest paths
- maximum flow
- minimum cost flow

Other CO Examples

- shortest paths
- maximum flow
- minimum cost flow
- approximation for NP-hard problems

Other CO Examples

- shortest paths
- maximum flow
- minimum cost flow
- approximation for NP-hard problems

Solution: Algorithms

Solution: Algorithms

- optimisation software for these problems

Solution: Algorithms

- optimisation software for these problems
- existing libraries: Boost Graph Library, LEMON, LEDA (, Gurobi)

Solution: Algorithms

- optimisation software for these problems
- existing libraries: Boost Graph Library, LEMON, LEDA (, Gurobi)
- involved algorithms, correctness!?

Solution: Algorithms

- optimisation software for these problems
- existing libraries: Boost Graph Library, LEMON, LEDA (, Gurobi)
- involved algorithms, correctness!?
- verification

Solution: Algorithms

- optimisation software for these problems
- existing libraries: Boost Graph Library, LEMON, LEDA (, Gurobi)
- involved algorithms, correctness!?
- verification

Isabelle/HOL

Isabelle/HOL

- interactive theorem prover (ITP)

Isabelle/HOL

- interactive theorem prover (ITP)
- write programs and mathematics (proofs!)

Isabelle/HOL

- interactive theorem prover (ITP)
- write programs and mathematics (proofs!)
- functional programming + logic

Isabelle/HOL

- interactive theorem prover (ITP)
- write programs and mathematics (proofs!)
- functional programming + logic
- formalising = writing maths in ITP

Isabelle/HOL

- interactive theorem prover (ITP)
- write programs and mathematics (proofs!)
- functional programming + logic
- formalising = writing maths in ITP

```
fun fac :: nat ⇒ nat where
  fac 0 = 1 |
  fac n = n * fac (n -1)

lemma log-fac-bounds:
  ∃ (c1 :: real) (c1 :: real).
    c1*n*ln n  ≤ ln (fac n)  ≤ c2*n*ln n
       ∧  c1 > 0  ∧  c2 > 0
```

Project Discussed Today

Project Discussed Today

- some optimisation problems can be generalised

Project Discussed Today

- some optimisation problems can be generalised
- formalisation of matroid and greedoid theory with focus on optimisation problems

Project Discussed Today

- some optimisation problems can be generalised
- formalisation of matroid and greedoid theory with focus on optimisation problems
- in the Isabelle/HOL prover

Project Discussed Today

- some optimisation problems can be generalised
- formalisation of matroid and greedoid theory with focus on optimisation problems
- in the Isabelle/HOL prover
- 3 executable and verified optimisation algorithms

Project Discussed Today

- some optimisation problems can be generalised
- formalisation of matroid and greedoid theory with focus on optimisation problems
- in the Isabelle/HOL prover
- 3 executable and verified optimisation algorithms
- yields 3 executable algorithms for minimum spanning tree and maximum cardinality bipartite matching

Project Discussed Today

- some optimisation problems can be generalised
- formalisation of matroid and greedoid theory with focus on optimisation problems
- in the Isabelle/HOL prover
- 3 executable and verified optimisation algorithms
- yields 3 executable algorithms for minimum spanning tree and maximum cardinality bipartite matching
- part of an Isabelle/HOL library on combinatorial optimisation (Abdulaziz, Ammer, Dordjonova, Koller, Madlener, Meenakshisundaram, Mehlhorn, Rimpapa)

Project Discussed Today

- some optimisation problems can be generalised
- formalisation of matroid and greedoid theory with focus on optimisation problems
- in the Isabelle/HOL prover
- 3 executable and verified optimisation algorithms
- yields 3 executable algorithms for minimum spanning tree and maximum cardinality bipartite matching
- part of an Isabelle/HOL library on combinatorial optimisation (Abdulaziz, Ammer, Dordjonova, Koller, Madlener, Meenakshisundaram, Mehlhorn, Rimpapa)
- both theoretical and practical interest

Table of Contents

What's a Matroid?

What's a Matroid?

Definition (Independence System)

A *ground set* E and a family of *independent sets* $\mathcal{F} \subseteq \mathcal{P}(E)$ is an independence system $(E, \mathcal{F})$ iff

M1. $\emptyset \in \mathcal{F}$

M2. $A \in \mathcal{F}$ and $B \subseteq A$ then $B \in \mathcal{F}$

What's a Matroid?

Definition (Independence System)

A *ground set* E and a family of *independent sets* $\mathcal{F} \subseteq \mathcal{P}(E)$ is an independence system $(E, \mathcal{F})$ iff

M1. $\emptyset \in \mathcal{F}$

M2. $A \in \mathcal{F}$ and $B \subseteq A$ then $B \in \mathcal{F}$

Definition (Matroid)

An independence system $(E, \mathcal{F})$ is a matroid iff

M3. $A \in \mathcal{F}$ and $B \in \mathcal{F}$ and $|B| > |A|$
then $\exists x \in B \setminus A.\ A \cup \{x\} \in \mathcal{F}$

What's a Matroid?

Definition (Independence System)

A *ground set* E and a family of *independent sets* $\mathcal{F} \subseteq \mathcal{P}(E)$ is an independence system $(E, \mathcal{F})$ iff

M1. $\emptyset \in \mathcal{F}$

M2. $A \in \mathcal{F}$ and $B \subseteq A$ then $B \in \mathcal{F}$

Definition (Matroid)

An independence system $(E, \mathcal{F})$ is a matroid iff

M3. $A \in \mathcal{F}$ and $B \in \mathcal{F}$ and $|B| > |A|$
then $\exists x \in B \setminus A.\ A \cup \{x\} \in \mathcal{F}$

Definition (Basis)

A basis B of $A \subseteq E$ is an inclusion-maximal independent subset of A. A basis of the independence system $\mathcal{F} \subseteq \mathcal{P}(E)$ is a basis of E.

Why Matroids?

Why Matroids?

- generalisation of linear independence

Why Matroids?

- generalisation of linear independence
- algebraic point of view for some optimisation problems

Why Matroids?

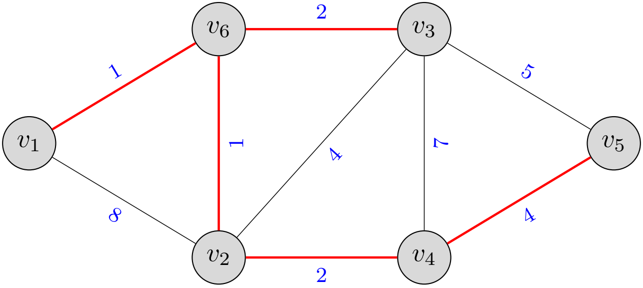
- generalisation of linear independence
- algebraic point of view for some optimisation problems
- weighted matroid optimisation: for costs c, find $X \in \mathcal{F}$ maximising $\sum_{x \in X} c(x)$. (or minimum weight basis)

Why Matroids?

- generalisation of linear independence
- algebraic point of view for some optimisation problems
- weighted matroid optimisation: for costs c, find $X \in \mathcal{F}$ maximising $\sum_{x \in X} c(x)$. (or minimum weight basis)

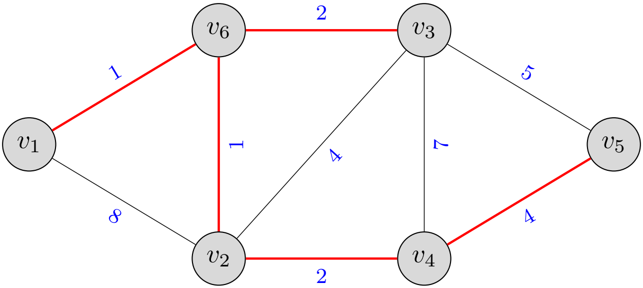
Standard Example: Minimum Spanning Tree and Maximum Weight Forest

- undirected (multi-)graph with edges E and costs $c : E \to \mathbb{R}^+$
- forest = acyclic subgraph
- tree = forest with a single component
- spanning tree minimising/forest maximising accumulated costs



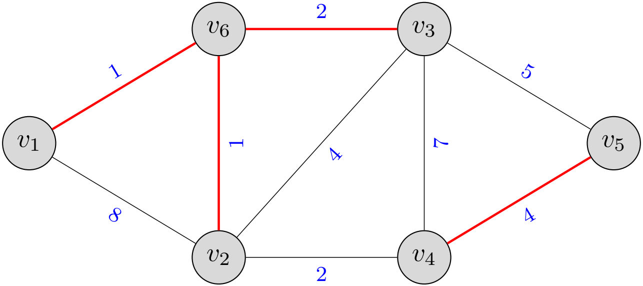
Standard Example: Minimum Spanning Tree and Maximum Weight Forest

- carrier set E
- independent sets: $T \subseteq E$ forming an acyclic subgraph



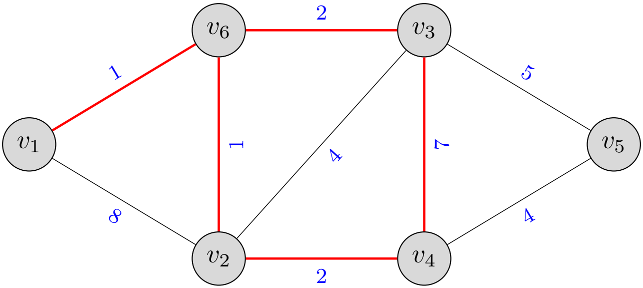
Standard Example: Minimum Spanning Tree and Maximum Weight Forest

- carrier set E
- independent sets: $T \subseteq E$ forming an acyclic subgraph



Standard Example: Minimum Spanning Tree and Maximum Weight Forest

- carrier set E
- independent sets: $T \subseteq E$ forming an acyclic subgraph



Standard Example: Minimum Spanning Tree and Maximum Weight Forest

- carrier set E
- independent sets: $T \subseteq E$ forming an acyclic subgraph

Standard Example: Minimum Spanning Tree and Maximum Weight Forest

- carrier set E
- independent sets: $T \subseteq E$ forming an acyclic subgraph
- matroid axioms satisfied

Standard Example: Minimum Spanning Tree and Maximum Weight Forest

- carrier set E
- independent sets: $T \subseteq E$ forming an acyclic subgraph
- matroid axioms satisfied
- independent sets are forests
- bases are spanning trees

Standard Example: Minimum Spanning Tree and Maximum Weight Forest

- carrier set E
- independent sets: $T \subseteq E$ forming an acyclic subgraph
- matroid axioms satisfied
- independent sets are forests
- bases are spanning trees
- maximum weight forest is maximum weight independent set
- minimum spanning tree is minimum weight basis

Theory of Matroids and Greedoids (Selection)

- Whitney (1935): introduction of matroids
- Tutte (1965): Lectures on Matroids, Homotopy Theorem
- Edmonds (1970, 1971): greedy algorithms, Matroid Intersection Theorem
- Lawler (1975): matroid intersection algorithms
- Seymour (1980): Decomposition Theorem for Regular Matroids
- Korte and Lovasz (1980): greedoids and greedy algorithms
- many concepts: set system, independence system, matroid, basis, circuit, rank, rank quotient, closure operator, greedoid, accessibility, etc. etc.

our main reference:

- *Combinatorial Optimization (6th Edition)* by Korte and Vygen

Formalisation of Matroids

- Mizar: basic matroid theory [Bancerek and Shidama 2008]
- Coq/Rocq: projective geometry and Desargues theorem [Magaud et al. 2012]
- Isabelle/HOL: basic matroid theory [Keinholz 2018], basis for our work
- Isabelle/HOL: Kruskal's Algorithm [Haslbeck et al. 2018], most related
- Lean: matroid theory [Nelson et al. github, 2023 - ongoing]
- Coq/Rocq: matroid-based automated prover [Magaud et al. 2024]

Table of Contents

The Best-In-Greedy Algorithm

The Best-In-Greedy Algorithm

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

[Rado 1957, Edmonds 1971, Jenkyns 1976, Korte and Hausmann 1978]

Sort $E := \{e_1, \ldots, e_n\}$ such that $c(e_1) \geq c(e_2) \geq \ldots \geq c(e_n)$;
$F \leftarrow \emptyset$;
for $i := 1$ *to* n **do**
  if $F \cup \{e_i\} \in \mathcal{F}$ **then** $F \leftarrow F \cup \{e_i\}$;
return F;

The Best-In-Greedy Algorithm

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

[Rado 1957, Edmonds 1971, Jenkyns 1976, Korte and Hausmann 1978]

Sort $E := \{e_1, \ldots, e_n\}$ such that $c(e_1) \geq c(e_2) \geq \ldots \geq c(e_n)$;
$F \leftarrow \emptyset$;
for $i := 1$ *to* n **do**
  if $F \cup \{e_i\} \in \mathcal{F}$ **then** $F \leftarrow F \cup \{e_i\}$;
return F;

- sort elements in descending order of costs

The Best-In-Greedy Algorithm

Algorithm 1: BestInGreedy$(E, \mathcal{F}, c)$
[Rado 1957, Edmonds 1971, Jenkyns 1976, Korte and Hausmann 1978]

Sort $E := \{e_1, \ldots, e_n\}$ such that $c(e_1) \geq c(e_2) \geq \ldots \geq c(e_n)$;
$F \leftarrow \emptyset$;
for $i := 1$ *to* n **do**
  if $F \cup \{e_i\} \in \mathcal{F}$ **then** $F \leftarrow F \cup \{e_i\}$;
return F;

- sort elements in descending order of costs
- process them one by one

The Best-In-Greedy Algorithm

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

[Rado 1957, Edmonds 1971, Jenkyns 1976, Korte and Hausmann 1978]

Sort $E := \{e_1, \ldots, e_n\}$ such that $c(e_1) \geq c(e_2) \geq \ldots \geq c(e_n)$;
$F \leftarrow \emptyset$;
for $i := 1$ *to* n **do**
  if $F \cup \{e_i\} \in \mathcal{F}$ **then** $F \leftarrow F \cup \{e_i\}$;
return F;

- sort elements in descending order of costs
- process them one by one
- add to solution if possible

The Best-In-Greedy Algorithm

Algorithm 1: BestInGreedy$(E, \mathcal{F}, c)$
[Rado 1957, Edmonds 1971, Jenkyns 1976, Korte and Hausmann 1978]

Sort $E := \{e_1, \ldots, e_n\}$ such that $c(e_1) \geq c(e_2) \geq \ldots \geq c(e_n)$;
$F \leftarrow \emptyset$;
for $i := 1$ *to* n **do**
  if $F \cup \{e_i\} \in \mathcal{F}$ **then** $F \leftarrow F \cup \{e_i\}$;
return F;

- sort elements in descending order of costs
- process them one by one
- add to solution if possible
- blackbox *independence oracle:* if $e \in E \setminus F$ and $F \in \mathcal{F}$, is $F \cup \{e\} \in \mathcal{F}$?

The Best-In-Greedy Algorithm

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

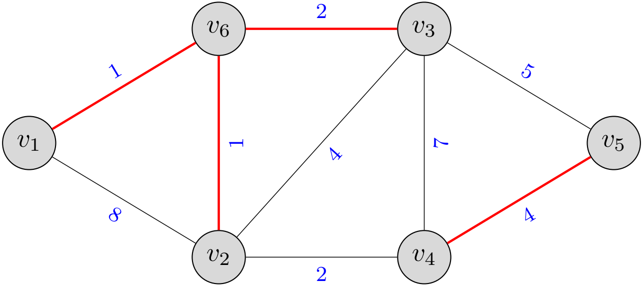
[Rado 1957, Edmonds 1971, Jenkyns 1976, Korte and Hausmann 1978]

Sort $E := \{e_1, \ldots, e_n\}$ such that $c(e_1) \geq c(e_2) \geq \ldots \geq c(e_n)$;
$F \leftarrow \emptyset$;
for $i := 1$ *to* n **do**
  if $F \cup \{e_i\} \in \mathcal{F}$ **then** $F \leftarrow F \cup \{e_i\}$;
return F;

- sort elements in descending order of costs
- process them one by one
- add to solution if possible
- blackbox *independence oracle:* if $e \in E \setminus F$ and $F \in \mathcal{F}$, is $F \cup \{e\} \in \mathcal{F}$?
- concrete problem: focus on implementing oracle

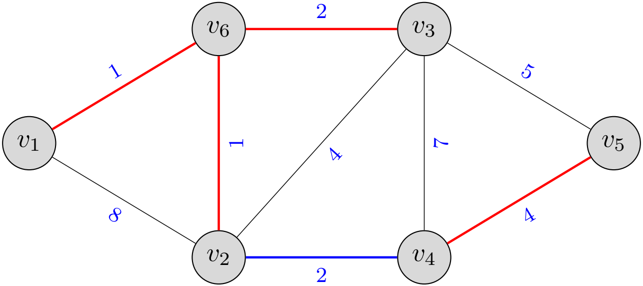
Standard Example: Minimum Spanning Tree

- red edges are acyclic, i.e. independent
- adding blue edge preserves independence



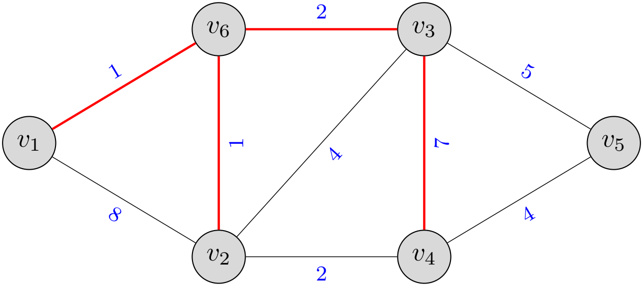
Standard Example: Minimum Spanning Tree

- red edges are acyclic, i.e. independent
- adding blue edge preserves independence



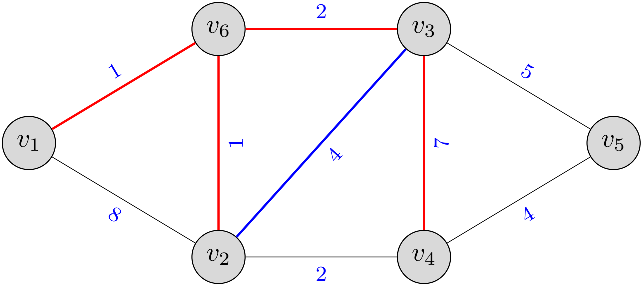
Standard Example: Minimum Spanning Tree

- independent sets: $T \subseteq E$ forming an acyclic subgraph
- blue edge breaks independence



Standard Example: Minimum Spanning Tree

- independent sets: $T \subseteq E$ forming an acyclic subgraph
- blue edge breaks independence



Standard Example: Minimum Spanning Tree

- given $T \subseteq E$ acyclic, $e \in E \setminus T$, is $T \cup \{e\}$ still acyclic?
- yes, iff endpoints x and y of e in different connected components of T
- is there path in T between x and y?
- Depth-First Search (our approach) or Breadth-First Search
- Union-Find for components of T (Haslbeck et al. with Imperative HOL and Refinement Framework)
- this is Kruskal's Algorithm

The Best-In-Greedy Algorithm

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

Sort $E := \{e_1, \ldots, e_n\}$ such that $c(e_1) \geq c(e_2) \geq \ldots \geq c(e_n)$;
$F \leftarrow \emptyset$;
for $i := 1$ *to* n **do**
  if $F \cup \{e_i\} \in \mathcal{F}$ **then** $F \leftarrow F \cup \{e_i\}$;
return F;

- *independence oracle:* if $e \in E \setminus F$ and $F \in \mathcal{F}$, is $F \cup \{e\} \in \mathcal{F}$?

Formalisation of Algorithm

```
locale Best-In-Greedy = matroid: Matroid-Specs
  where set-empty = set-empty for set-empty :: 'set +
  fixes carrier :: 'set and indep :: 'set ⇒ bool
    and sort-desc :: ('set ⇒ rat) ⇒ 'a list ⇒ 'a list
    and indep-oracle::'a ⇒ 'set ⇒ bool
```

Formalisation (Loop)

```
function BestInGreedy ::
    ('a, 'set) best-in-greedy-state
     ⇒ ('a, 'set) best-in-greedy-state
where
BestInGreedy state  =
 (case (carrier-list state) of
  [] ⇒ state |
  (x # xs) ⇒
  (if  indep-oracle  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⦈
```

Formalisation (Independence Oracle, simplified)

- if $e \in E \setminus F$ and $F \in \mathcal{F}$, is $F \cup \{e\} \in \mathcal{F}$?

```
context
 assumes local-indep-oracle:
  ⋀   F :: 'set  e :: 'a.
    ⟦ set-inv F; indep F; subseteq F carrier; e ∉ to-set F ⟧
    ⟹  indep-oracle e F  ⟷  indep (set-insert e F)
 and carrier-inv: set-inv carrier
```

Executability

- data structures to implement sets
- operations and behaviour specified by locale

```
locale Set =
fixes empty :: 's
fixes insert :: 'a ⇒ 's ⇒ 's
fixes delete :: 'a ⇒ 's ⇒ 's
...
fixes set :: 's ⇒ 'a set
fixes invar :: 's ⇒ bool
assumes set-empty:    set empty = {}
assumes set-insert:   invar s
                 ⟹  set(insert x s) = set s ∪ {x}
...
```

Executability

- data structures to implement sets
- operations and behaviour specified by locale

```
locale Set =
fixes empty :: 's
fixes insert :: 'a ⇒ 's ⇒ 's
fixes delete :: 'a ⇒ 's ⇒ 's
...
fixes set :: 's ⇒ 'a set
fixes invar :: 's ⇒ bool
assumes set-empty:    set empty = {}
assumes set-insert:   invar s
                ⟹   set(insert x s) = set s  ∪  {x}
...
```

- abstract data types
  [Wirth 1971, Hoare 1972, Liskov and Zilles 1974]

Executability

Executability

- same for other subprocedures, e.g. oracles

Executability

- same for other subprocedures, e.g. oracles
- instantiation to obtain executable algorithms for concrete problems, e.g. Kruskal's Algorithm (for MWF)

Executability

- same for other subprocedures, e.g. oracles
- instantiation to obtain executable algorithms for concrete problems, e.g. Kruskal's Algorithm (for MWF)
- generic for different implementations and matroids

Executability

- same for other subprocedures, e.g. oracles
- instantiation to obtain executable algorithms for concrete problems, e.g. Kruskal's Algorithm (for MWF)
- generic for different implementations and matroids
- stepwise refinement [Wirth 1971 + Hoare 1972]:
  replace instruction (e.g. $F \cup \{e\} \in \mathcal{F}$?) with more detailed instructions (e.g. does e add a cycle to F?)

Formalisation of the Oracle for Kruskal

```
definition local-indep-oracle e X =
((Card-Set2-RBT.subseteq (vset-insert e X) input-G)  ∧
 (lookup (Kruskal-E-to-G X) (fst e)  ≠  None
   ∧  lookup (Kruskal-E-to-G X) (snd e)  ≠  None ⟶
      return (dfs-impl (Kruskal-E-to-G X) (snd e)
        (dfs-initial-state (fst e))) = NotReachable))
```

Table of Contents

Properties of the Algorithm: Invariants

Properties of the Algorithm: Invariants

- preserved by loop iterations
- induction over loop execution

Properties of the Algorithm: Invariants

- preserved by loop iterations
- induction over loop execution

At the beginning of the loop body

- $F \subseteq E$
- $F \in \mathcal{F}$
- current `carrier-list` is $[e_i, ..., e_n]$ for current iteration i.
  (NB: $i = |E| - |$`carrier-list`$|$)
- current `result` is subset of $\{e_1, ..., e_{i-1}\}$
- $\forall 0 \leq j \leq i-1$:
  $\texttt{result}_j (= \{e_1, ..., e_{j-1}\} \cap \texttt{result})$ is basis of $\{e_1, ..., e_{j-1}\}$.

Properties of the Algorithm

Properties of the Algorithm

Theorem (Cost Bound [Jenkyns 1976, Korte and Hausmann 1978])
Let $(E, \mathcal{F})$ be an independence system, with $c : E \to \mathbb{R}_+$. Let F be the output of BestInGreedy. Then $c(F) \geq q(E, \mathcal{F}) \cdot \max_{X \in \mathcal{F}} c(X)$.

- $q(E, \mathcal{F})$ is the *rank quotient*, a number associated with every independence system
- $q(E, \mathcal{F}) = 1$ iff $(E, \mathcal{F})$ is matroid

Properties of the Algorithm

Theorem (Cost Bound [Jenkyns 1976, Korte and Hausmann 1978])

Let $(E, \mathcal{F})$ be an independence system, with $c : E \to \mathbb{R}_+$. Let F be the output of BestInGreedy*. Then $c(F) \geq q(E, \mathcal{F}) \cdot \max_{X \in \mathcal{F}} c(X)$.*

- $q(E, \mathcal{F})$ is the *rank quotient*, a number associated with every independence system
- $q(E, \mathcal{F}) = 1$ iff $(E, \mathcal{F})$ is matroid

Corollary

Let $(E, \mathcal{F})$ be a matroid, with $c : E \to \mathbb{R}_+$. BestInGreedy *finds X with $c(X)$ maximum.*

Properties of the Algorithm

Theorem (Cost Bound [Jenkyns 1976, Korte and Hausmann 1978])

Let $(E, \mathcal{F})$ be an independence system, with $c : E \to \mathbb{R}_+$. Let F be the output of BestInGreedy*. Then $c(F) \geq q(E, \mathcal{F}) \cdot \max_{X \in \mathcal{F}} c(X)$.*

- $q(E, \mathcal{F})$ is the *rank quotient*, a number associated with every independence system
- $q(E, \mathcal{F}) = 1$ iff $(E, \mathcal{F})$ is matroid

Corollary

Let $(E, \mathcal{F})$ be a matroid, with $c : E \to \mathbb{R}_+$. BestInGreedy *finds X with $c(X)$ maximum.*

- different proof for Corollary 2 already formalised by Haslbeck, Lammich and Biendarra (2018, see AFP).

Properties of the Algorithm

Properties of the Algorithm

Theorem (Tightness [Jenkyns 1976, Korte and Hausmann 1978])

Let $(E, \mathcal{F})$ be an independence system. There exists a cost function $c : E \to \mathbb{R}_+$ s.t. for the output F of BestInGreedy*,*
$c(F) = q(E, \mathcal{F}) \cdot \max_{X \in \mathcal{F}} c(X)$.

Properties of the Algorithm

Theorem (Tightness [Jenkyns 1976, Korte and Hausmann 1978])

Let $(E, \mathcal{F})$ be an independence system. There exists a cost function $c : E \to \mathbb{R}_+$ s.t. for the output F of BestInGreedy, $c(F) = q(E, \mathcal{F}) \cdot \max_{X \in \mathcal{F}} c(X)$.

Theorem (Characterisation [Rado 1957, Edmonds 1971])

An independence system $(E, \mathcal{F})$ is a matroid if and only if BestInGreedy *finds an optimal solution for the maximum weight independent set problem for $(E, \mathcal{F}, c)$ for all cost functions $c : E \to \mathbb{R}_+$.*

Table of Contents

Greedoids

Greedoids

Definition (Greedoid)

A *ground set* E and a family of *independent sets* $\mathcal{F} \subseteq \mathcal{P}(E)$ is a greedoid iff

M1. $\emptyset \in \mathcal{F}$

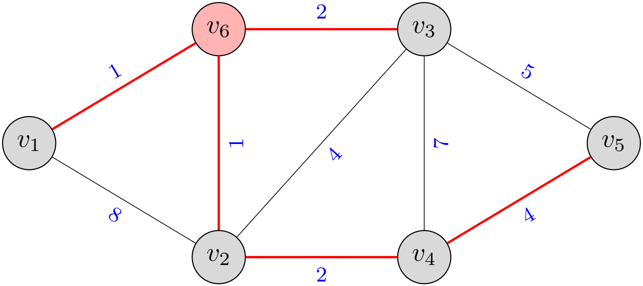
M3. $A \in \mathcal{F}$ and $B \in \mathcal{F}$ and $|B| > |A|$
then $\exists x \in B \setminus A.\ A \cup \{x\} \in \mathcal{F}$

Formalisation

```
locale greedoid =
  fixes E :: 'a set
  fixes F :: 'a set set
  assumes contains-empty-set: {} ∈ F
  and finite-E: finite E
  and in-F-in-E: ⋀ X. X ∈ F ⟹ X ⊆ E
  and third-condition:
     ⋀ X Y. ⟦ (X ∈ F);(Y ∈ F) ; (card X > card Y) ⟧
     ⟹ ∃x ∈ X - Y. Y ∪ {x} ∈ F
```

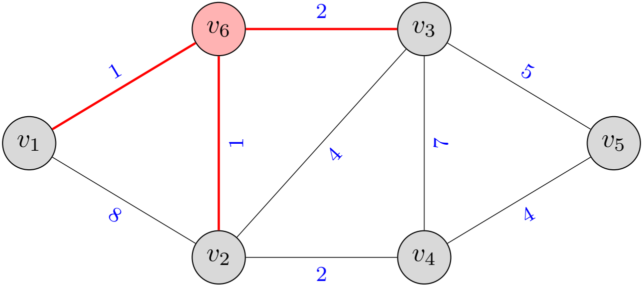
Greedoids: An Example

- for a vertex r, $T \subseteq E$ is an *arborescence* around r iff
  - T is a tree
  - r is in T
- for fixed r, set of arborescences satisifies M1+M3



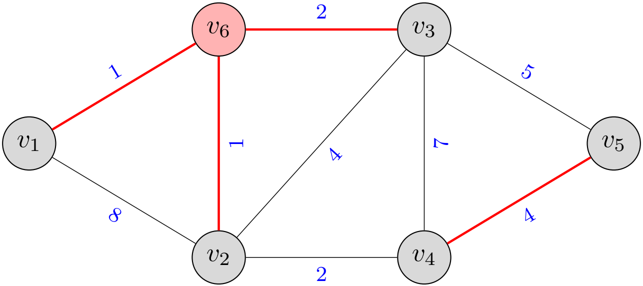
Greedoids: An Example

- for a vertex r, $T \subseteq E$ is an *arborescence* around r iff
  - T is a tree
  - r is in T
- for fixed r, set of arborescences satisifies M1+M3



Greedoids: An Example

- for a vertex r, $T \subseteq E$ is an *arborescence* around r iff
  - T is a tree
  - r is in T
- for fixed r, set of arborescences satisifies M1+M3



Properties of Greedoids

- greedoids are *accessible*
- *antimatroids* are greedoids
- *closure* operators
- *anti-exchange* property
- *strong exchange* property
- an algorithm ...

Greedoid Algorithm

Algorithm 2: GreedoidGreedy$(E, \mathcal{F}, c)$

$F \leftarrow \emptyset$;
while $\exists e \in E \setminus F.\ F \cup \{e\} \in \mathcal{F}$ **do**
  find $e \in E \setminus F$ with $F \cup \{e\} \in \mathcal{F}$ where $c(F \cup \{e\})$ is maximum;
  $F \leftarrow F \cup \{e\}$;
return F;

- as long as $\exists$ element to extend F, add the element e maximising $c(F \cup \{e\})$.
- independence oracle: if $e \in E \setminus F$ and $F \in \mathcal{F}$, is $F \cup \{e\} \in \mathcal{F}$?

Formalisation

```
definition find-best-candidate es c F' =  foldr (λ e acc.
  if e ∈ F' ∨ ¬ (orcl e F') then acc
  else (case acc of None ⇒ Some e |
                    Some d ⇒
    (if elements-costs c e > elements-costs c d then Some e
     else Some d))) es None

function (domintros) greedoid-greedy ::
  'a list ⇒ ('a set ⇒ real) ⇒ 'a list ⇒ 'a list  where
  greedoid-greedy es c  xs =
 (case  (find-best-candidate es c  (set xs)) of
      Some e ⇒ greedoid-greedy es c  (e#xs)
    | None ⇒ xs)
```

Properties of Greedoid Algorithm

Properties of Greedoid Algorithm

- invariants, similar to BestInGreedy

Properties of Greedoid Algorithm

- invariants, similar to BestInGreedy

Theorem (Korte and Vygen: Characterisation of Strong-Exchange Greedoids)

We fix a greedoid $(E, \mathcal{F})$*.* GreedoidGreedy *computes a maximum-weight basis in* $\mathcal{F}$ *for any order of iteration* $e_1, ..., e_n$ *and any modular cost function* $c : \mathcal{P}(E) \to \mathbb{R}$ *iff* $(E, \mathcal{F})$ *has the strong exchange property (SEP).*

- modular weight function: $c(A \cup B) = c(a) + c(B) - c(A \cap B)$ for all $A, B \subseteq E$
- SEP: 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 $(B - \{y\}) \cup \{x\} \in \mathcal{F}$.

```
theorem greedoid-characterisation:
(∀ c es. valid-modular-weight-func E c  ∧   E = set es
         ∧  distinct es
         ⟶ opt-basis c (set (greedoid-greedy es c Nil)))
 ⟷  strong-exchange-property E F
```

Table of Contents

Matroid Intersection

Matroid Intersection

- two matroids $(E, \mathcal{F}_1)$ and $(E, \mathcal{F}_2)$

Matroid Intersection

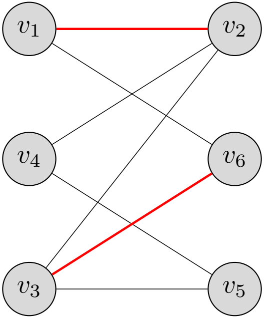
- two matroids $(E, \mathcal{F}_1)$ and $(E, \mathcal{F}_2)$
- find $X \in \mathcal{F}_1 \cap \mathcal{F}_2$ with maximum $|X|$

Matroid Intersection

- two matroids $(E, \mathcal{F}_1)$ and $(E, \mathcal{F}_2)$
- find $X \in \mathcal{F}_1 \cap \mathcal{F}_2$ with maximum $|X|$
- example: maximum cardinality bipartite matching

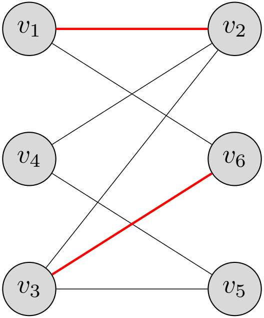
Matroid Intersection: Example

- matching: set of vertex disjoint edges
- bipartite: edges only between left and right



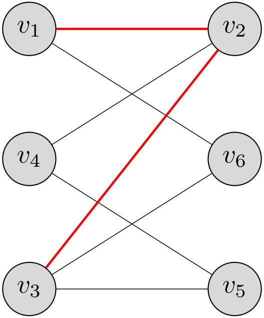
Matroid Intersection: Example

- two matroids $(E, \mathcal{F}_L)$ and $(E, \mathcal{F}_R)$
- $F \in \mathcal{F}_L$ iff no $e,\ f \in F$ share left endpoint
- $F \in \mathcal{F}_R$ iff no $e,\ f \in F$ share right endpoint



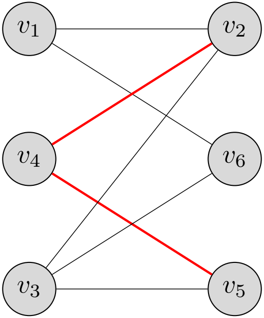
Matroid Intersection: Example

- two matroids $(E, \mathcal{F}_L)$ and $(E, \mathcal{F}_R)$
- $F \in \mathcal{F}_L$ iff no $e,\ f \in F$ share left endpoint
- $F \in \mathcal{F}_R$ iff no $e,\ f \in F$ share right endpoint



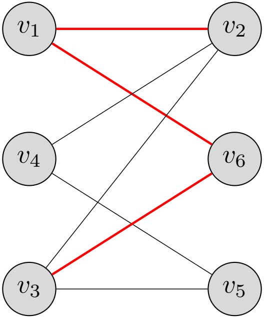
Matroid Intersection: Example

- two matroids $(E, \mathcal{F}_L)$ and $(E, \mathcal{F}_R)$
- $F \in \mathcal{F}_L$ iff no $e, \ f \in F$ share left endpoint
- $F \in \mathcal{F}_R$ iff no $e, \ f \in F$ share right endpoint



Matroid Intersection: Example

- two matroids $(E, \mathcal{F}_L)$ and $(E, \mathcal{F}_R)$
- $F \in \mathcal{F}_L$ iff no $e,\ f \in F$ share left endpoint
- $F \in \mathcal{F}_R$ iff no $e,\ f \in F$ share right endpoint



Theorem (Rank Criterion, Edmonds' Intersection Theorem)

For two matroids $(E, \mathcal{F}_1)$ *and* $(E, \mathcal{F}_2)$ *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* $X \in \mathcal{F}_1 \cap \mathcal{F}_2$*.*

Theorem (Rank Criterion, Edmonds' Intersection Theorem)

For two matroids $(E, \mathcal{F}_1)$ *and* $(E, \mathcal{F}_2)$ *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* $X \in \mathcal{F}_1 \cap \mathcal{F}_2$*.*

- we know: $|X| = r_1(X) + r_2(E \setminus X)$ for $X \in \mathcal{F}_1 \cap \mathcal{F}_2$ implies optimality

Theorem (Rank Criterion, Edmonds' Intersection Theorem)

For two matroids $(E, \mathcal{F}_1)$ and $(E, \mathcal{F}_2)$ 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 $X \in \mathcal{F}_1 \cap \mathcal{F}_2$.

- we know: $|X| = r_1(X) + r_2(E \setminus X)$ for $X \in \mathcal{F}_1 \cap \mathcal{F}_2$ implies optimality
- not (immediately) useful for algorithm

Theorem (Rank Criterion, Edmonds' Intersection Theorem)

For two matroids $(E, \mathcal{F}_1)$ and $(E, \mathcal{F}_2)$ 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 $X \in \mathcal{F}_1 \cap \mathcal{F}_2$.

- we know: $|X| = r_1(X) + r_2(E \setminus X)$ for $X \in \mathcal{F}_1 \cap \mathcal{F}_2$ implies optimality
- not (immediately) useful for algorithm

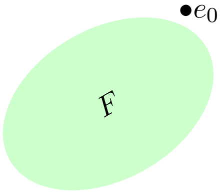
Table of Contents

Intersection Algorithm: Idea

- M2: $A \in \mathcal{F}$ and $B \subseteq A \implies B \in \mathcal{F}$
- M3: $A \in \mathcal{F}$ and $B \in \mathcal{F}$ and $|B| > |A|$
  $\implies \exists x \in B \setminus A.\ A \cup \{x\} \in \mathcal{F}$
- assume $F \in \mathcal{F}_1 \cap \mathcal{F}_2$ with $|F| < |F'|$ where $F' \in \mathcal{F}_1 \cap \mathcal{F}_2$
- by M3: find e_0 with $F \cup \{e_0\} \in \mathcal{F}_1$
- if $F \cup \{e_0\} \in \mathcal{F}_2$, add e_0 to F

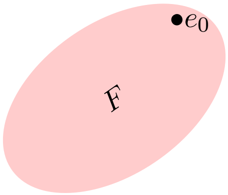
Intersection Algorithm: Idea

- M2: $A \in \mathcal{F}$ and $B \subseteq A \Longrightarrow B \in \mathcal{F}$
- M3: $A \in \mathcal{F}$ and $B \in \mathcal{F}$ and $|B| > |A|$
  $\Longrightarrow \exists x \in B \setminus A.\ A \cup \{x\} \in \mathcal{F}$
- $F \cup \{e_0\} \in \mathcal{F}_1$ but $F \cup \{e_0\} \notin \mathcal{F}_2$



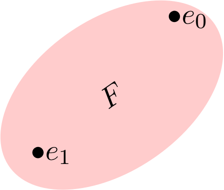
Intersection Algorithm: Idea

- M2: $A \in \mathcal{F}$ and $B \subseteq A \implies B \in \mathcal{F}$
- M3: $A \in \mathcal{F}$ and $B \in \mathcal{F}$ and $|B| > |A|$
  $\implies \exists x \in B \setminus A.\ A \cup \{x\} \in \mathcal{F}$
- $F \cup \{e_0\} \in \mathcal{F}_1$ but $F \cup \{e_0\} \notin \mathcal{F}_2$
- add e_0 to F, breaks independence w.r.t. $\mathcal{F}_2$



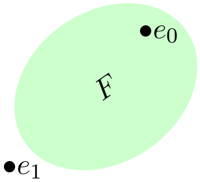
Intersection Algorithm: Idea

- M2: $A \in \mathcal{F}$ and $B \subseteq A \implies B \in \mathcal{F}$
- M3: $A \in \mathcal{F}$ and $B \in \mathcal{F}$ and $|B| > |A|$
  $\implies \exists x \in B \setminus A.\ A \cup \{x\} \in \mathcal{F}$
- $F \cup \{e_0\} \in \mathcal{F}_1$ but $F \cup \{e_0\} \notin \mathcal{F}_2$
- remove some e_1 from F



Intersection Algorithm: Idea

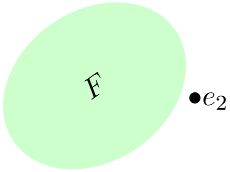
- M2: $A \in \mathcal{F}$ and $B \subseteq A \Longrightarrow B \in \mathcal{F}$
- M3: $A \in \mathcal{F}$ and $B \in \mathcal{F}$ and $|B| > |A|$
  $\Longrightarrow \exists x \in B \setminus A.\ A \cup \{x\} \in \mathcal{F}$
- $F \cup \{e_0\} \in \mathcal{F}_1$ but $F \cup \{e_0\} \notin \mathcal{F}_2$
- remove some e_1 from $F \Rightarrow$ independence w.r.t. $\mathcal{F}_2$



- set $F = F \cup \{e_0\} \setminus \{e_1\}$

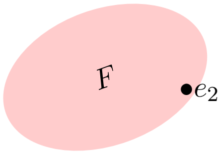
Intersection Algorithm: Idea

- M2: $A \in \mathcal{F}$ and $B \subseteq A \implies B \in \mathcal{F}$
- M3: $A \in \mathcal{F}$ and $B \in \mathcal{F}$ and $|B| > |A|$ $\implies \exists x \in B \setminus A.\ A \cup \{x\} \in \mathcal{F}$
- find $F \cup \{e_2\} \in \mathcal{F}_1$ but $F \cup \{e_2\} \notin \mathcal{F}_2$



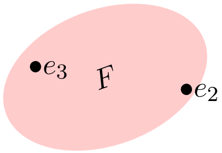
Intersection Algorithm: Idea

- M2: $A \in \mathcal{F}$ and $B \subseteq A \Longrightarrow B \in \mathcal{F}$
- M3: $A \in \mathcal{F}$ and $B \in \mathcal{F}$ and $|B| > |A|$
  $\Longrightarrow \exists x \in B \setminus A.\ A \cup \{x\} \in \mathcal{F}$
- find $F \cup \{e_2\} \in \mathcal{F}_1$ but $F \cup \{e_2\} \notin \mathcal{F}_2$



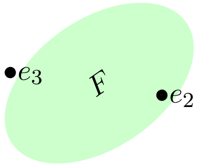
Intersection Algorithm: Idea

- M2: $A \in \mathcal{F}$ and $B \subseteq A \implies B \in \mathcal{F}$
- M3: $A \in \mathcal{F}$ and $B \in \mathcal{F}$ and $|B| > |A|$ $\implies \exists x \in B \setminus A.\ A \cup \{x\} \in \mathcal{F}$
- find $F \cup \{e_2\} \in \mathcal{F}_1$ but $F \cup \{e_2\} \notin \mathcal{F}_2$



Intersection Algorithm: Idea

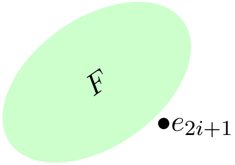
- M2: $A \in \mathcal{F}$ and $B \subseteq A \implies B \in \mathcal{F}$
- M3: $A \in \mathcal{F}$ and $B \in \mathcal{F}$ and $|B| > |A|$
  $\implies \exists x \in B \setminus A.\ A \cup \{x\} \in \mathcal{F}$
- find $F \cup \{e_2\} \in \mathcal{F}_1$ but $F \cup \{e_2\} \notin \mathcal{F}_2$



- set $F = F \cup \{e_2\} \setminus \{e_3\}$

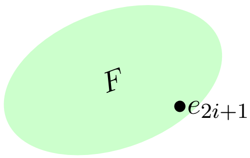
Intersection Algorithm: Idea

- continue until we find e_{2i+1} with $F \cup \{e_{2i+1}\} \in \mathcal{F}_1 \cap \mathcal{F}_2$



Intersection Algorithm: Idea

- continue until we find e_{2i+1} with $F \cup \{e_{2i+1}\} \in \mathcal{F}_1 \cap \mathcal{F}_2$



Intersection Algorithm: Idea

- *augmenting sequence/augmentation*: alternating insertion and deletion/complementary addition and subtraction
- sequences $x_0, y_1 ..., y_i, x_i$ with $F \cup \{x_0, ..., x_i\} \setminus \{y_1, ..., y_i\} \in \mathcal{F}_1 \cap \mathcal{F}_2$
- how to find?

Intersection Algorithm: Idea

- define auxiliary graph G_X with vertices E and find path between
  - $S_X = \{y.\ y \in E \setminus X \wedge X \cup \{y\} \in \mathcal{F}_1\}$
  - $T_X = \{y.\ y \in E \setminus X \wedge X \cup \{y\} \in \mathcal{F}_2\}$

  for $X \in \mathcal{F}_1 \cap \mathcal{F}_2$
- edges omitted
- involves various oracles
- shortest paths $x_0 y_1 x_1 \dots y_k x_k$ between S_X and T_X are augmenting paths, i.e. $X \setminus \{y_1, \dots, y_k\} \cup \{x_0, x_1, \dots, x_k\} \in \mathcal{F}_1 \cap \mathcal{F}_2$

Optimality Criterion

Theorem (Optimality Criterion by Korte and Vygen)

X is a set of maximum cardinality in $\mathcal{F}_1 \cap \mathcal{F}_2$ iff G_X does not contain a path from some $s \in S_X$ to some $t \in T_X$.

```
definition is-max X = (indep1 X  ∧  indep2 X  ∧
   (∄ Y. indep1 Y  ∧  indep2 Y  ∧  card Y > card X))

theorem maximum-characterisation:
  is-max X  ⟷
   ¬ (∃ p x y. x ∈ S  ∧  y ∈ T  ∧
     (vwalk-bet (A1  ∪  A2) x p y ∨ x = y))
```

Optimality Criterion

Optimality Criterion

- uses length minimality and Rank Criterion in proof

Optimality Criterion

- uses length minimality and Rank Criterion in proof
- computationally straightforward test for optimality

Optimality Criterion

- uses length minimality and Rank Criterion in proof
- computationally straightforward test for optimality
- analogue to e.g. Maxflow-Mincut Theorem or Berge's Lemma

Optimality Criterion

- uses length minimality and Rank Criterion in proof
- computationally straightforward test for optimality
- analogue to e.g. Maxflow-Mincut Theorem or Berge's Lemma

Algorithm 2: MaxMatroidIntersection(E, $\mathcal{F}_1$, $\mathcal{F}_2$)
[Lawler 1975, Korte and Vygen]

Initialise $X \leftarrow \emptyset$;
while $True$ **do**
  compute G_X*:* Initialise $S_X \leftarrow \emptyset$; $T_X \leftarrow \emptyset$; $A_{X,1} \leftarrow \emptyset$; $A_{X,2} \leftarrow \emptyset$;
  for $y \in E \setminus X$ **do**
    if $X \cup \{y\} \in \mathcal{F}_1$ **then** $S_X \leftarrow S_X \cup \{y\}$;
    else for $x \in X$ **do** [**if** $X \setminus \{x\} \cup \{y\} \in \mathcal{F}_1$ **then** $A_{X,1} \leftarrow A_{X,1} \cup \{(x, y)\}$;]
    if $X \cup \{y\} \in \mathcal{F}_2$ **then** $T_X \leftarrow T_X \cup \{y\}$;
    else for $x \in X$ **do** [**if** $X \setminus \{x\} \cup \{y\} \in \mathcal{F}_2$ **then** $A_{X,2} \leftarrow A_{X,2} \cup \{(y, x)\}$;]
  if $\exists$ *path leading from* S_X *to* T_X *via the edges in* $A_{X,1} \cup A_{X,2}$ **then**
    find a shortest path $P = x_0 y_1 x_1 \ldots y_s x_s$ leading from S_X to T_X;
    augment along P: $X \leftarrow X \cup \{x_0, \ldots, x_s\} \setminus \{y_1, \ldots, y_s\}$;
  else return X as maximum cardinality set in $\mathcal{F}_1 \cap \mathcal{F}_2$;

Table of Contents

Conclusion

Conclusion

- greedoids formalised for the first time

Conclusion

- greedoids formalised for the first time
- maximum cardinality matroid intersection

Conclusion

- greedoids formalised for the first time
- maximum cardinality matroid intersection
- algorithmic characterisations of matroids and greedoids

Conclusion

Conclusion

- two main approaches for polynomial-time algorithms

Conclusion

- two main approaches for polynomial-time algorithms
  - greedy

Conclusion

- two main approaches for polynomial-time algorithms
  - greedy
  - augmentation (also for flows and matching)

Conclusion

- two main approaches for polynomial-time algorithms
  - greedy
  - augmentation (also for flows and matching)
- optimality criterion for matroid intersection similar to

Conclusion

- two main approaches for polynomial-time algorithms
  - greedy
  - augmentation (also for flows and matching)
- optimality criterion for matroid intersection similar to
  - maximum flow: maximum flow iff $\nexists$ augmenting path (part of Maxflow-Mincut Theorem)

Conclusion

- two main approaches for polynomial-time algorithms
  - greedy
  - augmentation (also for flows and matching)
- optimality criterion for matroid intersection similar to
  - maximum flow: maximum flow iff $\nexists$ augmenting path (part of Maxflow-Mincut Theorem)
  - minimum cost flow: minimum costs iff $\nexists$ augmenting cycle

Conclusion

- two main approaches for polynomial-time algorithms
  - greedy
  - augmentation (also for flows and matching)
- optimality criterion for matroid intersection similar to
  - maximum flow: maximum flow iff $\nexists$ augmenting path (part of Maxflow-Mincut Theorem)
  - minimum cost flow: minimum costs iff $\nexists$ augmenting cycle
  - matching: maximum matching iff $\nexists$ augmenting path (Berge's Lemma)

Conclusion

- two main approaches for polynomial-time algorithms
  - greedy
  - augmentation (also for flows and matching)
- optimality criterion for matroid intersection similar to
  - maximum flow: maximum flow iff $\nexists$ augmenting path (part of Maxflow-Mincut Theorem)
  - minimum cost flow: minimum costs iff $\nexists$ augmenting cycle
  - matching: maximum matching iff $\nexists$ augmenting path (Berge's Lemma)

Conclusion

Conclusion

- executable algorithms obtained

Conclusion

- executable algorithms obtained

Conclusion

- executable algorithms obtained
  - spanning forest (BestInGreedy, Kruskal's Algorithm)

Conclusion

- executable algorithms obtained
  - spanning forest (BestInGreedy, Kruskal's Algorithm)
  - spanning tree (GreedoidGreedy, Prim's Algorithm)

Conclusion

- executable algorithms obtained
  - spanning forest (BestInGreedy, Kruskal's Algorithm)
  - spanning tree (GreedoidGreedy, Prim's Algorithm)
  - bipartite matching (MaxMatroidInter, naive bipartite matching)

Conclusion

- executable algorithms obtained
  - spanning forest (BestInGreedy, Kruskal's Algorithm)
  - spanning tree (GreedoidGreedy, Prim's Algorithm)
  - bipartite matching (MaxMatroidInter, naive bipartite matching)

Conclusion

Conclusion

- integrated into an Isabelle/HOL library on combinatorial optimisation
  ($\Rightarrow$ verified optimisation software)

Conclusion

- integrated into an Isabelle/HOL library on combinatorial optimisation
  ($\Rightarrow$ verified optimisation software)
- part of reasoning conducted at abstract level/algebraic point of view:

Conclusion

- integrated into an Isabelle/HOL library on combinatorial optimisation
  ($\Rightarrow$ verified optimisation software)
- part of reasoning conducted at abstract level/algebraic point of view:
  - no inaccuracies found

Conclusion

- integrated into an Isabelle/HOL library on combinatorial optimisation
  ($\Rightarrow$ verified optimisation software)
- part of reasoning conducted at abstract level/algebraic point of view:
  - no inaccuracies found
  - reasoning simplified: optimality criterion for matroid intersection vs. Berge's Lemma

Conclusion

- integrated into an Isabelle/HOL library on combinatorial optimisation
  (⇒ verified optimisation software)
- part of reasoning conducted at abstract level/algebraic point of view:
  - no inaccuracies found
  - reasoning simplified: optimality criterion for matroid intersection vs. Berge's Lemma
  - shared reasoning and instantiation for concrete problems (library!)

Conclusion

- integrated into an Isabelle/HOL library on combinatorial optimisation
  ($\Rightarrow$ verified optimisation software)
- part of reasoning conducted at abstract level/algebraic point of view:
  - no inaccuracies found
  - reasoning simplified: optimality criterion for matroid intersection vs. Berge's Lemma
  - shared reasoning and instantiation for concrete problems (library!)
- 17.4K lines (matroids, greedoids, algorithms: 11K, graphs: 2.9K, instantiation: 3.5K)

Conclusion

- integrated into an Isabelle/HOL library on combinatorial optimisation
  ($\Rightarrow$ verified optimisation software)
- part of reasoning conducted at abstract level/algebraic point of view:
  - no inaccuracies found
  - reasoning simplified: optimality criterion for matroid intersection vs. Berge's Lemma
  - shared reasoning and instantiation for concrete problems (library!)
- 17.4K lines (matroids, greedoids, algorithms: 11K, graphs: 2.9K, instantiation: 3.5K)
- disadvantage: performance loss possible

Conclusion

Conclusion

- methodology:

Conclusion

- methodology:
  - oracles: stepwise refinement [Wirth 1971, Hoare 1972]

Conclusion

- methodology:
  - oracles: stepwise refinement [Wirth 1971, Hoare 1972]
  - locales for stepwise refinement [Nipkow 2015, Abdulaziz, Mehlhorn and Nipkow 2019, Maric 2020]

Conclusion

- methodology:
  - oracles: stepwise refinement [Wirth 1971, Hoare 1972]
  - locales for stepwise refinement [Nipkow 2015, Abdulaziz, Mehlhorn and Nipkow 2019, Maric 2020]
  - abstract data types by locales [Wirth 1971, Hoare 1972, Liskov and Zilles 1974]

Conclusion

- methodology:
  - oracles: stepwise refinement [Wirth 1971, Hoare 1972]
  - locales for stepwise refinement [Nipkow 2015, Abdulaziz, Mehlhorn and Nipkow 2019, Maric 2020]
  - abstract data types by locales [Wirth 1971, Hoare 1972, Liskov and Zilles 1974]
  - functional programming

Conclusion

- methodology:
  - oracles: stepwise refinement [Wirth 1971, Hoare 1972]
  - locales for stepwise refinement [Nipkow 2015, Abdulaziz, Mehlhorn and Nipkow 2019, Maric 2020]
  - abstract data types by locales [Wirth 1971, Hoare 1972, Liskov and Zilles 1974]
  - functional programming
  - recursive functions to model loops

Conclusion

- methodology:
  - oracles: stepwise refinement [Wirth 1971, Hoare 1972]
  - locales for stepwise refinement [Nipkow 2015, Abdulaziz, Mehlhorn and Nipkow 2019, Maric 2020]
  - abstract data types by locales [Wirth 1971, Hoare 1972, Liskov and Zilles 1974]
  - functional programming
  - recursive functions to model loops
  - program states as records

Conclusion

- methodology:
  - oracles: stepwise refinement [Wirth 1971, Hoare 1972]
  - locales for stepwise refinement [Nipkow 2015, Abdulaziz, Mehlhorn and Nipkow 2019, Maric 2020]
  - abstract data types by locales [Wirth 1971, Hoare 1972, Liskov and Zilles 1974]
  - functional programming
  - recursive functions to model loops
  - program states as records
  - mathematically involved invariants

THANK YOU!

Mohammad Abdulaziz Thomas Ammer

Shriya Meenakshisundaram Adem Rimpapa