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)

October 1, 2025

Table of Contents

The Project

The Project

- formalisation of matroid and greedoid theory with focus on optimisation problems

The Project

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

The Project

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

The Project

- 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

The Project

- 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)

The Project

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

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?

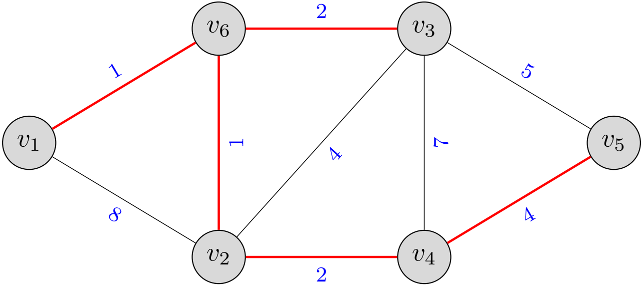
- generalisation of linear independence
- algebraic point of view for some optimisation problems
- weighted matroid optimisation: for costs c, find $X \in \mathcal{F}$ maximising $\sum_{x \in X} c(x)$. (or minimum weight basis)

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)

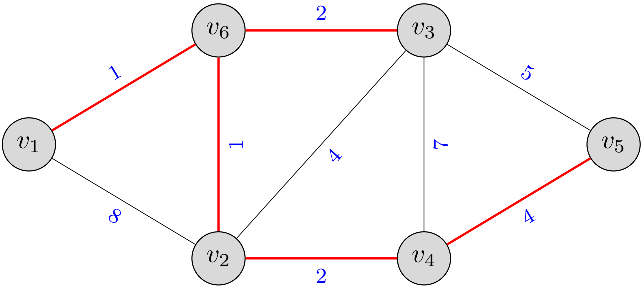
Standard Example: Minimum Spanning Tree and Maximum Weight Forest

- 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



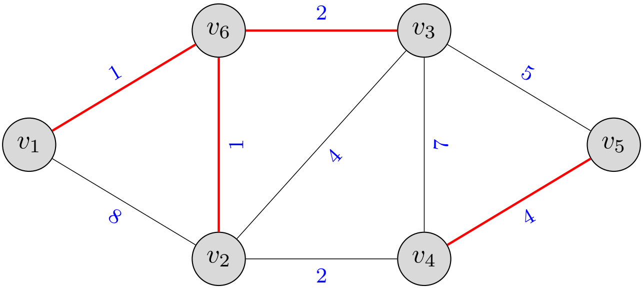
Standard Example: Minimum Spanning Tree and Maximum Weight Forest

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



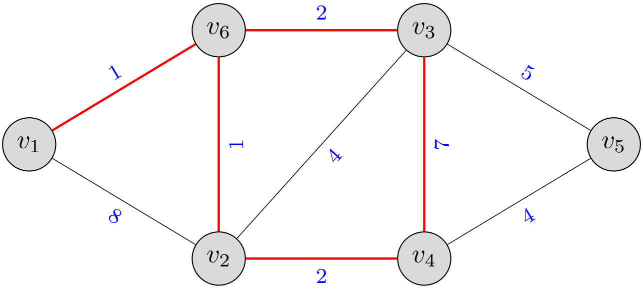
Standard Example: Minimum Spanning Tree and Maximum Weight Forest

- 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

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

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 $\exists x \in B \setminus A.\ A \cup \{x\} \in \mathcal{F}$

Properties of Greedoid Algorithm

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* $c : \mathcal{P}(E) \rightarrow \mathbb{R}$ *iff* $(E, \mathcal{F})$ *has the strong exchange property (SEP).*

```
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

Optimality Criterion

Optimality Criterion

- for $X \in \mathcal{F}_1 \cap \mathcal{F}_2$, define G_X, S_X, T_X (omitted)
- use oracles

Optimality Criterion

- for $X \in \mathcal{F}_1 \cap \mathcal{F}_2$, define G_X, S_X, T_X (omitted)
- use oracles

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))
```

Intersection Algorithm: Idea

Intersection Algorithm: Idea

- shortest S_X-T_X-paths in G_X of the form $x_0 y_1 x_1 ... y_i x_i$

Intersection Algorithm: Idea

- shortest S_X-T_X-paths in G_X of the form $x_0 y_1 x_1 ... y_i x_i$
- these are augmenting sequences: for $X \in \mathcal{F}_1 \cap \mathcal{F}_2$, $X \cup \{x_0, ..., x_i\} \setminus \{y_1, ..., y_i\} \in \mathcal{F}_1 \cap \mathcal{F}_2$

Intersection Algorithm: Idea

- shortest S_X-T_X-paths in G_X of the form $x_0 y_1 x_1 ... y_i x_i$
- these are augmenting sequences: for $X \in \mathcal{F}_1 \cap \mathcal{F}_2$, $X \cup \{x_0, ..., x_i\} \setminus \{y_1, ..., y_i\} \in \mathcal{F}_1 \cap \mathcal{F}_2$
- this is an augmentation

Intersection Algorithm: Idea

- shortest S_X-T_X-paths in G_X of the form $x_0 y_1 x_1 ... y_i x_i$
- these are augmenting sequences: for $X \in \mathcal{F}_1 \cap \mathcal{F}_2$, $X \cup \{x_0, ..., x_i\} \setminus \{y_1, ..., y_i\} \in \mathcal{F}_1 \cap \mathcal{F}_2$
- this is an augmentation

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 $P = x_0 y_1 x_1 ... y_s x_s$ leading from S_X to T_X;

augment along P: $X \leftarrow X \cup \{x_0, ..., x_s\} \setminus \{y_1, ..., y_s\}$;

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
- uses augmentation (common in combinatorial optimisation)

Conclusion

- greedoids formalised for the first time
- maximum cardinality matroid intersection
- uses augmentation (common in combinatorial optimisation)
- algorithmic characterisations of matroids and greedoids

Conclusion

Conclusion

- executable algorithms obtained

Conclusion

- executable algorithms obtained
- integrated into an Isabelle/HOL library on combinatorial optimisation

Conclusion

- executable algorithms obtained
- integrated into an Isabelle/HOL library on combinatorial optimisation
- suitable for library: part of reasoning
  conducted at abstract level/algebraic point of view

Conclusion

- executable algorithms obtained
- integrated into an Isabelle/HOL library on combinatorial optimisation
- suitable for library: part of reasoning
  conducted at abstract level/algebraic point of view
- 17.4K lines (matroids, greedoids, algorithms: 11K, graphs: 2.9K, instantiation: 3.5K)

Conclusion

- executable algorithms obtained
- integrated into an Isabelle/HOL library on combinatorial optimisation
- suitable for library: part of reasoning
  conducted at abstract level/algebraic point of view
- 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]
  - function package 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]
  - function package 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]
  - function package to model loops
  - program states as records

THANK YOU!

Mohammad Abdulaziz Thomas Ammer

Shriya Meenakshisundaram Adem Rimpapa