A Combinatorial Polynomial Algorithm for Weighted Abstract Cut Packing

S. Thomas McCormick   Britta Peis

Sauder School of Business, UBC; TU Berlin

Ga Tech, 22 March 2012

S. Thomas McCormick
Sauder School of Business
University of British Columbia

Submodularity in Supply Chain Management

Selected applications of submodularity in SCM:

- Shen, Coullard, and Daskin (2003) model a facility location-inventory problem where column generation uses SFM.

Submodularity in Supply Chain Management

Selected applications of submodularity in SCM:

- Shen, Coullard, and Daskin (2003) model a facility location-inventory problem where column generation uses SFM.
- Huh and Roundy (2005) model capacity expansion sequencing decisions in semiconductor fabs, where determining an optimal sequence with general costs uses a (parametric) SFM subroutine.

Submodularity in Supply Chain Management

Selected applications of submodularity in SCM:

- Shen, Coullard, and Daskin (2003) model a facility location-inventory problem where column generation uses SFM.
- Huh and Roundy (2005) model capacity expansion sequencing decisions in semiconductor fabs, where determining an optimal sequence with general costs uses a (parametric) SFM subroutine.
- Koole and van der Sluis (2003) use multimodularity ($L^\natural$-convexity) to schedule a call center.

Submodularity in Supply Chain Management

Selected applications of submodularity in SCM:

- Shen, Coullard, and Daskin (2003) model a facility location-inventory problem where column generation uses SFM.
- Huh and Roundy (2005) model capacity expansion sequencing decisions in semiconductor fabs, where determining an optimal sequence with general costs uses a (parametric) SFM subroutine.
- Koole and van der Sluis (2003) use multimodularity ($L^\natural$-convexity) to schedule a call center.
- Begen and Queyranne (2011) use L-convexity to schedule stochastic appointments for, e.g., surgeries.

Outline

1. Combinatorial Optimization
   - Packing problems

Outline

1. Combinatorial Optimization
   - Packing problems
2. Hoffman's Models
   - Lattice Polyhedra
   - Blocking

Outline

1. Combinatorial Optimization
   - Packing problems
2. Hoffman's Models
   - Lattice Polyhedra
   - Blocking
3. Algorithms
   - Primal-Dual Algorithm
   - P-D for WACP

Outline

1. Combinatorial Optimization
   - Packing problems
2. Hoffman's Models
   - Lattice Polyhedra
   - Blocking
3. Algorithms
   - Primal-Dual Algorithm
   - P-D for WACP
4. Conclusion
   - Open questions

Outline

1. Combinatorial Optimization
   - Packing problems
2. Hoffman's Models
   - Lattice Polyhedra
   - Blocking
3. Algorithms
   - Primal-Dual Algorithm
   - P-D for WACP
4. Conclusion
   - Open questions

Packing problems

A generic packing problem has

- A finite set E of elements

Packing problems

A generic packing problem has

- A finite set E of elements
- A family $\mathcal{D}$ of subsets of E, i.e., $D \in \mathcal{D} \implies D \subseteq E$.

Packing problems

A generic packing problem has

- A finite set E of elements
- A family $\mathcal{D}$ of subsets of E, i.e., $D \in \mathcal{D} \implies D \subseteq E$.
- A vector $u \in \mathbb{Z}^E$ of capacities on elements.

Packing problems

A generic packing problem has

- A finite set E of elements
- A family $\mathcal{D}$ of subsets of E, i.e., $D \in \mathcal{D} \implies D \subseteq E$.
- A vector $u \in \mathbb{Z}^E$ of capacities on elements.
- A vector $r \in \mathbb{Z}^{\mathcal{D}}$ of rewards on subsets.

Packing problems

A generic packing problem has

- A finite set E of elements
- A family $\mathcal{D}$ of subsets of E, i.e., $D \in \mathcal{D} \implies D \subseteq E$.
- A vector $u \in \mathbb{Z}^E$ of capacities on elements.
- A vector $r \in \mathbb{Z}^{\mathcal{D}}$ of rewards on subsets.
- The decision is to choose a weight y_D to put on each $D \in \mathcal{D}$ such that the total weight packed into e is at most u_e $\forall$ $e \in E$.

Packing problems

A generic packing problem has

- A finite set E of elements
- A family $\mathcal{D}$ of subsets of E, i.e., $D \in \mathcal{D} \implies D \subseteq E$.
- A vector $u \in \mathbb{Z}^E$ of capacities on elements.
- A vector $r \in \mathbb{Z}^{\mathcal{D}}$ of rewards on subsets.
- The decision is to choose a weight y_D to put on each $D \in \mathcal{D}$ such that the total weight packed into e is at most $u_e \; \forall \; e \in E$.
- And among such feasible packings, find one that maximizes $r^T y$.

Packing problems

A generic packing problem has

- A finite set E of elements
- A family $\mathcal{D}$ of subsets of E, i.e., $D \in \mathcal{D} \implies D \subseteq E$.
- A vector $u \in \mathbb{Z}^E$ of capacities on elements.
- A vector $r \in \mathbb{Z}^{\mathcal{D}}$ of rewards on subsets.
- The decision is to choose a weight y_D to put on each $D \in \mathcal{D}$ such that the total weight packed into e is at most $u_e \ \forall \ e \in E$.
- And among such feasible packings, find one that maximizes $r^T y$.
- We are usually interested in finding *integer* optimal solutions.

Packing problems

A generic packing problem has

- A finite set E of elements
- A family $\mathcal{D}$ of subsets of E, i.e., $D \in \mathcal{D} \implies D \subseteq E$.
- A vector $u \in \mathbb{Z}^E$ of capacities on elements.
- A vector $r \in \mathbb{Z}^{\mathcal{D}}$ of rewards on subsets.
- The decision is to choose a weight y_D to put on each $D \in \mathcal{D}$ such that the total weight packed into e is at most $u_e\ \forall\ e \in E$.
- And among such feasible packings, find one that maximizes $r^T y$.
- We are usually interested in finding *integer* optimal solutions.
- This generic problem has many applications, e.g., flow is packing paths into arcs, connectivity is packing trees into edges, etc.

Packing as an LP

- Now formulate a packing problem as an LP (it's more natural to make packing the dual):

Packing as an LP

- Now formulate a packing problem as an LP (it's more natural to make packing the dual):
  - put dual packing variable y_D on each $D \in \mathcal{D}$;

Packing as an LP

- Now formulate a packing problem as an LP (it's more natural to make packing the dual):
  - put dual packing variable y_D on each $D \in \mathcal{D}$;
  - put primal weight x_e on each element $e \in E$.

Packing as an LP

- Now formulate a packing problem as an LP (it's more natural to make packing the dual):
  - put dual packing variable y_D on each $D \in \mathcal{D}$;
  - put primal weight x_e on each element $e \in E$.
- The dual linear programs are:

$$
\text{(D)} \quad \max \sum_{D} r_D y_D \qquad\qquad \text{(P)} \quad \min \sum_{e} u_e x_e
$$

$$
\text{s.t.} \sum_{D \ni e} y_D \le u_e \quad \forall e \in E \qquad\qquad \text{s.t.} \sum_{e \in D} x_e \ge r_D \quad \forall D \in \mathcal{D}
$$

$$
y \ge 0 \qquad\qquad\qquad\qquad x \ge 0
$$

Packing as an LP

- Now formulate a packing problem as an LP (it's more natural to make packing the dual):
  - put dual packing variable y_D on each $D \in \mathcal{D}$;
  - put primal weight x_e on each element $e \in E$.
- The dual linear programs are:

$$
\begin{array}{llll}
\text{(D)} & \max & \displaystyle\sum_{D} r_D y_D & \\
& \text{s.t.} & \displaystyle\sum_{D \ni e} y_D \le u_e & \forall e \in E \\
& & y \ge 0 &
\end{array}
\qquad
\begin{array}{llll}
\text{(P)} & \min & \displaystyle\sum_{e} u_e x_e & \\
& \text{s.t.} & \displaystyle\sum_{e \in D} x_e \ge r_D & \forall D \in \mathcal{D} \\
& & x \ge 0 &
\end{array}
$$

"packing subsets into elements"

Packing as an LP

- Now formulate a packing problem as an LP (it's more natural to make packing the dual):
  - put dual packing variable y_D on each $D \in \mathcal{D}$;
  - put primal weight x_e on each element $e \in E$.
- The dual linear programs are:

$$\text{(D)} \quad \max \sum_{D} r_D y_D \qquad \text{s.t.} \sum_{D \ni e} y_D \leq u_e \quad \forall e \in E \qquad y \geq 0$$

$$\text{(P)} \quad \min \sum_{e} u_e x_e \qquad \text{s.t.} \sum_{e \in D} x_e \geq r_D \quad \forall D \in \mathcal{D} \qquad x \geq 0$$

"packing subsets into elements" "covering subsets by elements"

Packing as an LP

- Now formulate a packing problem as an LP (it's more natural to make packing the dual):
  - put dual packing variable y_D on each $D \in \mathcal{D}$;
  - put primal weight x_e on each element $e \in E$.
- The dual linear programs are:

$$\text{(D)} \quad \max \sum_D r_D y_D \quad \text{s.t.} \sum_{D \ni e} y_D \le u_e \quad \forall e \in E, \quad y \ge 0$$

$$\text{(P)} \quad \min \sum_e u_e x_e \quad \text{s.t.} \sum_{e \in D} x_e \ge r_D \quad \forall D \in \mathcal{D}, \quad x \ge 0$$

Big Question: When do these LPs have guaranteed integer optimal solutions?

An example packing LP

- Consider:

$$
\max \mathbb{1}^T y
$$

$$
\text{s.t.} \quad \begin{pmatrix}
1 & & 1 & & & 1 & & & \\
1 & 1 & & & 1 & & & & \\
 & 1 & 1 & & 1 & 1 & & & \\
 & 1 & & 1 & & & & 1 & \\
 & & 1 & 1 & 1 & 1 & & 1 & \\
 & & 1 & 1 & & & 1 & & \\
 & & & & 1 & 1 & 1 & 1 & \\
 & & & & 1 & & 1 & & 1 \\
 & & & & & 1 & & 1 & 1
\end{pmatrix} y \le \begin{pmatrix} 1 \\ 5 \\ 5 \\ 8 \\ 4 \\ 7 \\ 9 \\ 3 \\ 6 \end{pmatrix}
$$

$$
y \ge 0.
$$

An example packing LP

- Consider:

$$\max \mathbb{1}^T y$$

$$\text{s.t.} \quad \begin{pmatrix} 1 & & 1 & & & 1 & & & \\ 1 & 1 & & & 1 & & & & \\ & 1 & 1 & & 1 & 1 & & & \\ & 1 & & 1 & & & & 1 & \\ & & 1 & 1 & 1 & 1 & & 1 & \\ & & 1 & 1 & & & 1 & & \\ & & & & 1 & 1 & 1 & 1 & \\ & & & & 1 & & 1 & & 1 \\ & & & & & 1 & & 1 & 1 \end{pmatrix} y \leq \begin{pmatrix} 1 \\ 5 \\ 5 \\ 8 \\ 4 \\ 7 \\ 9 \\ 3 \\ 6 \end{pmatrix}$$

$$y \geq 0.$$

- Does this LP have an integer optimal solution?

An example packing LP

- Consider:

$$\max \mathbb{1}^T y$$

$$\text{s.t.} \quad \begin{pmatrix} 1 & & 1 & & & 1 & & & \\ 1 & 1 & & & 1 & & & & \\ & 1 & 1 & & 1 & 1 & & & \\ & 1 & & 1 & & & & 1 & \\ & & 1 & 1 & 1 & 1 & & 1 & \\ & & 1 & 1 & & & 1 & & \\ & & & & 1 & 1 & 1 & 1 & \\ & & & & 1 & & 1 & & 1 \\ & & & & & 1 & & 1 & 1 \end{pmatrix} y \leq \begin{pmatrix} 1 \\ 5 \\ 5 \\ 8 \\ 4 \\ 7 \\ 9 \\ 3 \\ 6 \end{pmatrix}$$

$$y \geq 0.$$

- Does this LP have an integer optimal solution?
- What if we change the RHS u? The objective r?

More on the example

- This LP has an integer optimal solution: $y^* = (1\ 4\ 0\ 4\ 0\ 0\ 3\ 0\ 0)$ of value 12.

More on the example

- This LP has an integer optimal solution: $y^* = (1\ 4\ 0\ 4\ 0\ 0\ 3\ 0\ 0)$ of value 12.
- In fact, it can be shown that this LP has integer optimal solutions for *any* RHS u.

More on the example

- This LP has an integer optimal solution: $y^* = (1\ 4\ 0\ 4\ 0\ 0\ 3\ 0\ 0)$ of value 12.
- In fact, it can be shown that this LP has integer optimal solutions for *any* RHS u.
- The same holds true for some objectives r:

More on the example

- This LP has an integer optimal solution: $y^* = (1\ 4\ 0\ 4\ 0\ 0\ 3\ 0\ 0)$ of value 12.
- In fact, it can be shown that this LP has integer optimal solutions for *any* RHS u.
- The same holds true for some objectives r:
  - E.g., $r = (4\ 3\ 2\ 3\ 1\ 1\ 3\ 2\ 4)$ has integer optimal solution $y^* = (1\ 4\ 0\ 4\ 0\ 0\ 0\ 0\ 3)$ of value 40 for the given RHS u, and this is true for any integral u.

More on the example

- This LP has an integer optimal solution: $y^* = (1\ 4\ 0\ 4\ 0\ 0\ 3\ 0\ 0)$ of value 12.
- In fact, it can be shown that this LP has integer optimal solutions for *any* RHS u.
- The same holds true for some objectives r:
  - E.g., $r = (4\ 3\ 2\ 3\ 1\ 1\ 3\ 2\ 4)$ has integer optimal solution $y^* = (1\ 4\ 0\ 4\ 0\ 0\ 0\ 0\ 3)$ of value 40 for the given RHS u, and this is true for any integral u.
- But not all objectives r:

More on the example

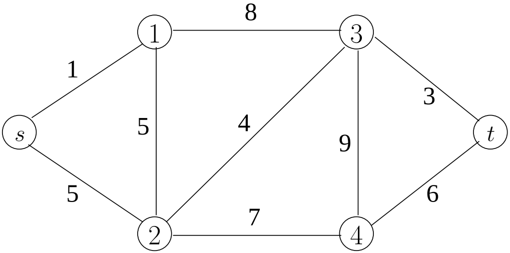
- This LP has an integer optimal solution: $y^* = (1\ 4\ 0\ 4\ 0\ 0\ 3\ 0\ 0)$ of value 12.
- In fact, it can be shown that this LP has integer optimal solutions for *any* RHS u.
- The same holds true for some objectives r:
  - E.g., $r = (4\ 3\ 2\ 3\ 1\ 1\ 3\ 2\ 4)$ has integer optimal solution $y^* = (1\ 4\ 0\ 4\ 0\ 0\ 0\ 0\ 3)$ of value 40 for the given RHS u, and this is true for any integral u.
- But not all objectives r:
  - E.g., $r = (0\ 9\ 0\ 0\ 9\ 0\ 0\ 9\ 0)$ has fractional optimal solution $y^* = (0\ 4.5\ 0\ 0\ 0.5\ 0\ 0\ 3.5\ 2.5)$ with value 76.5 for the given RHS u.

More on the example

- This LP has an integer optimal solution: $y^* = (1\ 4\ 0\ 4\ 0\ 0\ 3\ 0\ 0)$ of value 12.
- In fact, it can be shown that this LP has integer optimal solutions for *any* RHS u.
- The same holds true for some objectives r:
  - E.g., $r = (4\ 3\ 2\ 3\ 1\ 1\ 3\ 2\ 4)$ has integer optimal solution $y^* = (1\ 4\ 0\ 4\ 0\ 0\ 0\ 0\ 3)$ of value 40 for the given RHS u, and this is true for any integral u.
- But not all objectives r:
  - E.g., $r = (0\ 9\ 0\ 0\ 9\ 0\ 0\ 9\ 0)$ has fractional optimal solution $y^* = (0\ 4.5\ 0\ 0\ 0.5\ 0\ 0\ 3.5\ 2.5)$ with value 76.5 for the given RHS u.
- How do I know that the first two objectives are "good" for all RHS?

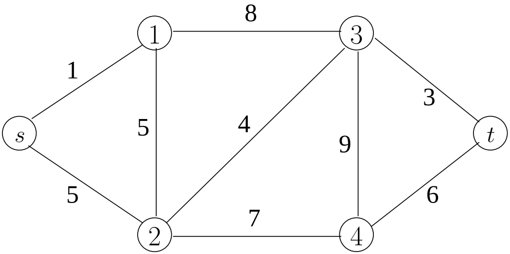
How the example was constructed

- Consider the following graph:



How the example was constructed

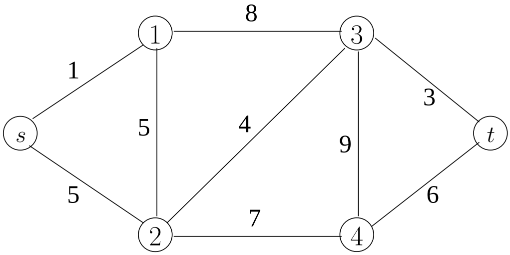
- Consider the following graph:



- There is a 1–1 correspondence between E and the nine edges of this graph.

How the example was constructed

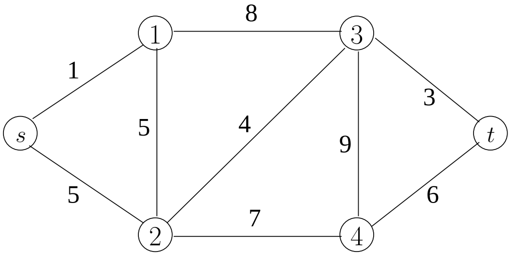
- Consider the following graph:



- There is a 1–1 correspondence between E and the nine edges of this graph.
- There is a 1–1 correspondence between the 9 interesting s–t cuts in this graph and the columns of the constraint matrix.

How the example was constructed

- Consider the following graph:



- There is a 1–1 correspondence between E and the nine edges of this graph.
- There is a 1–1 correspondence between the 9 interesting s–t cuts in this graph and the columns of the constraint matrix.
- Why does this lead to integer optimal LP solutions?

The primal covering LP

- Recall that the primal covering LP has variables x_e ...

The primal covering LP

- Recall that the primal covering LP has variables x_e ...
- ...and constraints $\sum_{e \in D} x_e \geq 1$ for all $D \in \mathcal{D}$.

The primal covering LP

- Recall that the primal covering LP has variables x_e ...
- ... and constraints $\sum_{e \in D} x_e \geq 1$ for all $D \in \mathcal{D}$.
- Imagine that x is 0–1, so that it picks out a subset of edges.

The primal covering LP

- Recall that the primal covering LP has variables x_e ...
- ... and constraints $\sum_{e \in D} x_e \geq 1$ for all $D \in \mathcal{D}$.
- Imagine that x is 0–1, so that it picks out a subset of edges.
- What subsets of edges hit every s–t cut?

The primal covering LP

- Recall that the primal covering LP has variables x_e ...
- ...and constraints $\sum_{e \in D} x_e \geq 1$ for all $D \in \mathcal{D}$.
- Imagine that x is 0–1, so that it picks out a subset of edges.
- What subsets of edges hit every s–t cut?
- The s–t paths are the minimal edge subsets hitting every s–t cut.

The primal covering LP

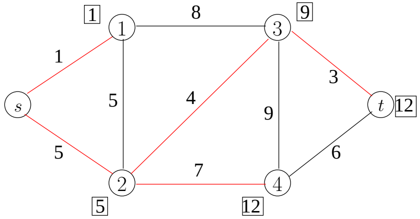
- Recall that the primal covering LP has variables x_e ...
- ...and constraints $\sum_{e \in D} x_e \geq 1$ for all $D \in \mathcal{D}$.
- Imagine that x is 0–1, so that it picks out a subset of edges.
- What subsets of edges hit every s–t cut?
- The s–t paths are the minimal edge subsets hitting every s–t cut.
- Therefore the primal LP is just Shortest Path.

The primal covering LP

- Recall that the primal covering LP has variables x_e ...
- ... and constraints $\sum_{e \in D} x_e \geq 1$ for all $D \in \mathcal{D}$.
- Imagine that x is 0–1, so that it picks out a subset of edges.
- What subsets of edges hit every s–t cut?
- The s–t paths are the minimal edge subsets hitting every s–t cut.
- Therefore the primal LP is just Shortest Path.
- And in fact Dijkstra's Algorithm gives an integer optimal solution to this form of Shortest Path.

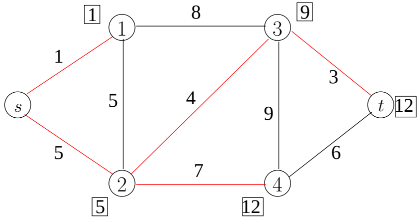
Going back to the dual packing LP

- Here is the Dijkstra solution with its shortest path tree:

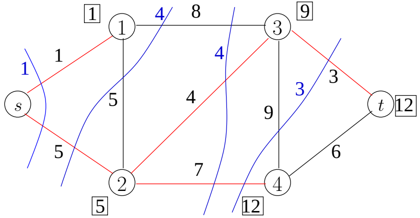


Going back to the dual packing LP

- Here is the Dijkstra solution with its shortest path tree:



- Recall that we can greedily construct a tight cut packing that proves that this shortest path tree is optimal:



Generalizing this behavior

- Since we know that Dijkstra, and this greedy cut packing, work for any non-negative capacities u, we know that we get integer optimal solutions for all RHS u.

Generalizing this behavior

- Since we know that Dijkstra, and this greedy cut packing, work for any non-negative capacities u, we know that we get integer optimal solutions for all RHS u.
- It is very cool that this random-looking constraint matrix always has an integer optimal solution *with the special objective vector* $\mathbb{1}$.

Generalizing this behavior

- Since we know that Dijkstra, and this greedy cut packing, work for any non-negative capacities u, we know that we get integer optimal solutions for all RHS u.
- It is very cool that this random-looking constraint matrix always has an integer optimal solution *with the special objective vector* $\mathbb{1}$.
- LPs such as this where you get guaranteed integer optimal solutions for all RHSs, but only for some special objective vectors, are called Totally Dual Integral, or TDI.

Generalizing this behavior

- Since we know that Dijkstra, and this greedy cut packing, work for any non-negative capacities u, we know that we get integer optimal solutions for all RHS u.
- It is very cool that this random-looking constraint matrix always has an integer optimal solution *with the special objective vector* $\mathbb{1}$.
- LPs such as this where you get guaranteed integer optimal solutions for all RHSs, but only for some special objective vectors, are called Totally Dual Integral, or TDI.
- A natural question here is whether we can generalize this sort of example to a broader class of packing LPs with 0–1 constraint matrices.

Generalizing this behavior

- Since we know that Dijkstra, and this greedy cut packing, work for any non-negative capacities u, we know that we get integer optimal solutions for all RHS u.
- It is very cool that this random-looking constraint matrix always has an integer optimal solution *with the special objective vector* $\mathbb{1}$.
- LPs such as this where you get guaranteed integer optimal solutions for all RHSs, but only for some special objective vectors, are called Totally Dual Integral, or TDI.
- A natural question here is whether we can generalize this sort of example to a broader class of packing LPs with 0–1 constraint matrices.
- Hoffman did it . . .

Outline

1. Combinatorial Optimization
   - Packing problems
2. Hoffman's Models
   - Lattice Polyhedra
   - Blocking
3. Algorithms
   - Primal-Dual Algorithm
   - P-D for WACP
4. Conclusion
   - Open questions

Alan Hoffman's lattice polyhedron model

- We are given a finite set of elements E (nodes/arcs/mixed)

Alan Hoffman's lattice polyhedron model

- We are given a finite set of elements E (nodes/arcs/mixed)
  - Each $e \in E$ has capacity u_e

Alan Hoffman's lattice polyhedron model

- We are given a finite set of elements E (nodes/arcs/mixed)
  - Each $e \in E$ has capacity u_e
- And a family $\mathcal{L}$ of cuts, where

Alan Hoffman's lattice polyhedron model

- We are given a finite set of elements E (nodes/arcs/mixed)
  - Each $e \in E$ has capacity u_e
- And a family $\mathcal{L}$ of cuts, where
  - $D \in \mathcal{L}$ means that $D \subseteq E$

Alan Hoffman's lattice polyhedron model

- We are given a finite set of elements E (nodes/arcs/mixed)
  - Each $e \in E$ has capacity u_e
- And a family $\mathcal{L}$ of cuts, where
  - $D \in \mathcal{L}$ means that $D \subseteq E$
  - $\mathcal{L}$ is a lattice with partial order $\preceq$ and operations $\wedge$ and $\vee$ satisfying

Alan Hoffman's lattice polyhedron model

- We are given a finite set of elements E (nodes/arcs/mixed)
  - Each $e \in E$ has capacity u_e
- And a family $\mathcal{L}$ of cuts, where
  - $D \in \mathcal{L}$ means that $D \subseteq E$
  - $\mathcal{L}$ is a lattice with partial order $\preceq$ and operations $\wedge$ and $\vee$ satisfying
    - $D_i \prec D_j \prec D_k \implies D_i \cap D_k \subseteq D_j$ (consecutive), and

Alan Hoffman's lattice polyhedron model

- We are given a finite set of elements E (nodes/arcs/mixed)
  - Each $e \in E$ has capacity u_e
- And a family $\mathcal{L}$ of cuts, where
  - $D \in \mathcal{L}$ means that $D \subseteq E$
  - $\mathcal{L}$ is a lattice with partial order $\preceq$ and operations $\wedge$ and $\vee$ satisfying
    - $D_i \prec D_j \prec D_k \implies D_i \cap D_k \subseteq D_j$ (consecutive), and
    - $(D_i \wedge D_j) \cup (D_i \vee D_j) \subseteq D_i \cup D_j$ (submodular).

Alan Hoffman's lattice polyhedron model

- We are given a finite set of elements E (nodes/arcs/mixed)
  - Each $e \in E$ has capacity u_e
- And a family $\mathcal{L}$ of cuts, where
  - $D \in \mathcal{L}$ means that $D \subseteq E$
  - $\mathcal{L}$ is a lattice with partial order $\preceq$ and operations $\wedge$ and $\vee$ satisfying
    - $D_i \prec D_j \prec D_k \implies D_i \cap D_k \subseteq D_j$ (consecutive), and
    - $(D_i \wedge D_j) \cup (D_i \vee D_j) \subseteq D_i \cup D_j$ (submodular).
  - each $D \in \mathcal{L}$ has a per unit reward r_D (the *weight* of D)

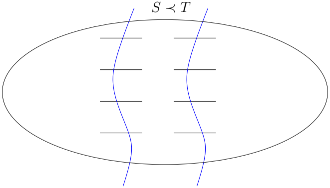
Alan Hoffman's lattice polyhedron model

- We are given a finite set of elements E (nodes/arcs/mixed)
  - Each $e \in E$ has capacity u_e
- And a family $\mathcal{L}$ of cuts, where
  - $D \in \mathcal{L}$ means that $D \subseteq E$
  - $\mathcal{L}$ is a lattice with partial order $\preceq$ and operations $\wedge$ and $\vee$ satisfying
    - $D_i \prec D_j \prec D_k \implies D_i \cap D_k \subseteq D_j$ (consecutive), and
    - $(D_i \wedge D_j) \cup (D_i \vee D_j) \subseteq D_i \cup D_j$ (submodular).
  - each $D \in \mathcal{L}$ has a per unit reward r_D (the *weight* of D)
- r satisfies a kind of supermodularity:

$$r_{D_i \wedge D_j} + r_{D_i \vee D_j} \geq r_{D_i} + r_{D_j}.$$

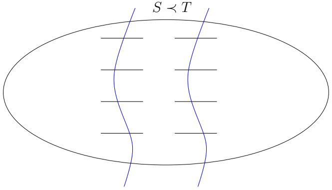
Understanding the lattice axioms

Ordinary cuts are partially ordered:

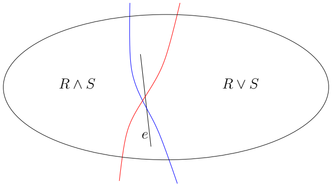


Understanding the lattice axioms

Ordinary cuts are partially ordered:

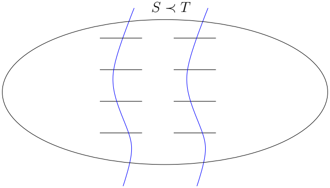


Ordinary cuts have meet and join, submodularity:

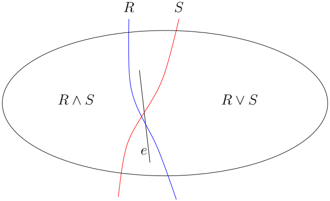


Understanding the lattice axioms

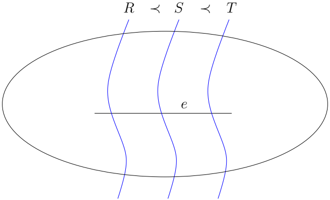
Ordinary cuts are partially ordered:



Ordinary cuts have meet and join, submodularity:



Ordinary cuts are consecutive ($e \in R \cap T \implies e \in S$):



The lattice polyhedron (Weighted Abstract Cut Packing) linear programs

- The lattice polyhedron *Weighted Abstract Cut Packing (WACP)* problem associated with E and $\mathcal{L}$ puts

The lattice polyhedron (Weighted Abstract Cut Packing) linear programs

- The lattice polyhedron *Weighted Abstract Cut Packing (WACP)* problem associated with E and $\mathcal{L}$ puts
  - packing variable y_D on each $D \in \mathcal{L}$;

The lattice polyhedron (Weighted Abstract Cut Packing) linear programs

- The lattice polyhedron *Weighted Abstract Cut Packing (WACP)* problem associated with E and $\mathcal{L}$ puts
  - packing variable y_D on each $D \in \mathcal{L}$;
  - weight x_e on each element $e \in E$.

The lattice polyhedron (Weighted Abstract Cut Packing) linear programs

- The lattice polyhedron *Weighted Abstract Cut Packing (WACP)* problem associated with E and $\mathcal{L}$ puts
  - packing variable y_D on each $D \in \mathcal{L}$;
  - weight x_e on each element $e \in E$.
- The dual linear programs are:

$$\text{(D)} \quad \max \sum_{D} r_D y_D \qquad \text{s.t.} \sum_{D \ni e} y_D \le u_e \quad \forall e \in E, \qquad y \ge 0$$

$$\text{(P)} \quad \min \sum_{e} u_e x_e \qquad \text{s.t.} \sum_{e \in D} x_e \ge r_D \quad \forall D \in \mathcal{L}, \qquad x \ge 0$$

The lattice polyhedron (Weighted Abstract Cut Packing) linear programs

- The lattice polyhedron *Weighted Abstract Cut Packing (WACP)* problem associated with E and $\mathcal{L}$ puts
  - packing variable y_D on each $D \in \mathcal{L}$;
  - weight x_e on each element $e \in E$.
- The dual linear programs are:

$$\text{(D)} \quad \max \sum_{D} r_D y_D \qquad \text{s.t.} \sum_{D \ni e} y_D \le u_e \quad \forall e \in E \qquad y \ge 0$$

"packing cuts into elements"

$$\text{(P)} \quad \min \sum_{e} u_e x_e \qquad \text{s.t.} \sum_{e \in D} x_e \ge r_D \quad \forall D \in \mathcal{L} \qquad x \ge 0$$

The lattice polyhedron (Weighted Abstract Cut Packing) linear programs

- The lattice polyhedron *Weighted Abstract Cut Packing (WACP)* problem associated with E and $\mathcal{L}$ puts
  - packing variable y_D on each $D \in \mathcal{L}$;
  - weight x_e on each element $e \in E$.
- The dual linear programs are:

$$\text{(D)} \quad \max \sum_{D} r_D y_D \qquad \text{s.t.} \sum_{D \ni e} y_D \le u_e \quad \forall e \in E \qquad y \ge 0$$

"packing cuts into elements"

$$\text{(P)} \quad \min \sum_{e} u_e x_e \qquad \text{s.t.} \sum_{e \in D} x_e \ge r_D \quad \forall D \in \mathcal{L} \qquad x \ge 0$$

"covering cuts by elements"

The lattice polyhedron (Weighted Abstract Cut Packing) linear programs

- The lattice polyhedron *Weighted Abstract Cut Packing (WACP)* problem associated with E and $\mathcal{L}$ puts
  - packing variable y_D on each $D \in \mathcal{L}$;
  - weight x_e on each element $e \in E$.
- The dual linear programs are:

$$
\begin{array}{llll}
\text{(D)} & \max & \displaystyle\sum_D r_D y_D & \\
 & \text{s.t.} & \displaystyle\sum_{D \ni e} y_D \le u_e & \forall e \in E \\
 & & y \ge 0 &
\end{array}
\qquad
\begin{array}{llll}
\text{(P)} & \min & \displaystyle\sum_e u_e x_e & \\
 & \text{s.t.} & \displaystyle\sum_{e \in D} x_e \ge r_D & \forall D \in \mathcal{L} \\
 & & x \ge 0 &
\end{array}
$$

If $\mathcal{L}$ is just s–t cuts in a max flow network, and $r \equiv 1$, then this is just the usual blocking dual formulation of Dijkstra shortest path.

The lattice polyhedron (Weighted Abstract Cut Packing) linear programs

- The lattice polyhedron *Weighted Abstract Cut Packing (WACP)* problem associated with E and $\mathcal{L}$ puts
  - packing variable y_D on each $D \in \mathcal{L}$;
  - weight x_e on each element $e \in E$.
- The dual linear programs are:

$$\text{(D)} \quad \max \sum_D r_D y_D \qquad \text{s.t.} \sum_{D \ni e} y_D \le u_e \quad \forall e \in E, \qquad y \ge 0$$

$$\text{(P)} \quad \min \sum_e u_e x_e \qquad \text{s.t.} \sum_{e \in D} x_e \ge r_D \quad \forall D \in \mathcal{L}, \qquad x \ge 0$$

Theorem (Hoffman & Schwartz '76)

When r and u are integral, (P) and (D) have integral optimal solutions.

Other applications

Lattice polyhedra would not be so interesting unless they included interesting applications other than Shortest Path:

- Dilworth's Theorem (chains and antichains in posets) and various Greene-Kleitman generalizations.

Other applications

Lattice polyhedra would not be so interesting unless they included interesting applications other than Shortest Path:

- Dilworth's Theorem (chains and antichains in posets) and various Greene-Kleitman generalizations.
- Shortest Path in hypergraphs.

Other applications

Lattice polyhedra would not be so interesting unless they included interesting applications other than Shortest Path:

- Dilworth's Theorem (chains and antichains in posets) and various Greene-Kleitman generalizations.
- Shortest Path in hypergraphs.
- Polymatroids and intersections of polymatroids.

Other applications

Lattice polyhedra would not be so interesting unless they included interesting applications other than Shortest Path:

- Dilworth's Theorem (chains and antichains in posets) and various Greene-Kleitman generalizations.
- Shortest Path in hypergraphs.
- Polymatroids and intersections of polymatroids.
- Min-cost arborescence.

Other applications

Lattice polyhedra would not be so interesting unless they included interesting applications other than Shortest Path:

- Dilworth's Theorem (chains and antichains in posets) and various Greene-Kleitman generalizations.
- Shortest Path in hypergraphs.
- Polymatroids and intersections of polymatroids.
- Min-cost arborescence.
- Our example with $r = (4\ 3\ 2\ 3\ 1\ 1\ 3\ 2\ 4)$ has integer optimal solutions for all RHS u because this r is supermodular: each $r_D = 6 - (\#$ edges crossing $D)$.

Other applications

Lattice polyhedra would not be so interesting unless they included interesting applications other than Shortest Path:

- Dilworth's Theorem (chains and antichains in posets) and various Greene-Kleitman generalizations.
- Shortest Path in hypergraphs.
- Polymatroids and intersections of polymatroids.
- Min-cost arborescence.
- Our example with $r = (4\ 3\ 2\ 3\ 1\ 1\ 3\ 2\ 4)$ has integer optimal solutions for all RHS u because this r is supermodular: each $r_D = 6 - (\#$ edges crossing $D)$.
- Our example with $r = (0\ 9\ 0\ 0\ 9\ 0\ 0\ 9\ 0)$ can have a fractional solution because this r is not supermodular.

Other applications

Lattice polyhedra would not be so interesting unless they included interesting applications other than Shortest Path:

- Dilworth's Theorem (chains and antichains in posets) and various Greene-Kleitman generalizations.
- Shortest Path in hypergraphs.
- Polymatroids and intersections of polymatroids.
- Min-cost arborescence.
- Our example with $r = (4\ 3\ 2\ 3\ 1\ 1\ 3\ 2\ 4)$ has integer optimal solutions for all RHS u because this r is supermodular: each $r_D = 6 - (\#$ edges crossing $D)$.
- Our example with $r = (0\ 9\ 0\ 0\ 9\ 0\ 0\ 9\ 0)$ can have a fractional solution because this r is not supermodular.
- Etc, etc ...

Paths and cuts block each other

- Set family $\mathcal{D}$ is a clutter if $R, S \in \mathcal{D}$, then $R \not\subset S$ and $S \not\subset R$ (edge sets of s–t cuts are a clutter).

Paths and cuts block each other

- Set family $\mathcal{D}$ is a clutter if $R, S \in \mathcal{D}$, then $R \not\subset S$ and $S \not\subset R$ (edge sets of s–t cuts are a clutter).
- Define the blocker of $\mathcal{D}$, $B(\mathcal{D})$, to be the set of minimal subsets Q of E such that $Q \cap D \neq \emptyset \ \forall \ D \in \mathcal{D}$; thus $B(\mathcal{D})$ is also a clutter.

Paths and cuts block each other

- Set family $\mathcal{D}$ is a clutter if $R, S \in \mathcal{D}$, then $R \not\subset S$ and $S \not\subset R$ (edge sets of s–t cuts are a clutter).
- Define the blocker of $\mathcal{D}$, $B(\mathcal{D})$, to be the set of minimal subsets Q of E such that $Q \cap D \neq \emptyset \ \forall\ D \in \mathcal{D}$; thus $B(\mathcal{D})$ is also a clutter.
- Fact: $B(B(\mathcal{D})) = \mathcal{D}$, and so blockers come in dual pairs.

Paths and cuts block each other

- Set family $\mathcal{D}$ is a clutter if $R, S \in \mathcal{D}$, then $R \not\subset S$ and $S \not\subset R$ (edge sets of s–t cuts are a clutter).
- Define the blocker of $\mathcal{D}$, $B(\mathcal{D})$, to be the set of minimal subsets Q of E such that $Q \cap D \neq \emptyset \ \forall \ D \in \mathcal{D}$; thus $B(\mathcal{D})$ is also a clutter.
- Fact: $B(B(\mathcal{D})) = \mathcal{D}$, and so blockers come in dual pairs.
- Easy to see that the families of s–t paths and s–t cuts are a blocking pair.

Paths and cuts block each other

- Set family $\mathcal{D}$ is a clutter if $R, S \in \mathcal{D}$, then $R \not\subset S$ and $S \not\subset R$ (edge sets of s–t cuts are a clutter).
- Define the blocker of $\mathcal{D}$, $B(\mathcal{D})$, to be the set of minimal subsets Q of E such that $Q \cap D \neq \emptyset \ \forall\ D \in \mathcal{D}$; thus $B(\mathcal{D})$ is also a clutter.
- Fact: $B(B(\mathcal{D})) = \mathcal{D}$, and so blockers come in dual pairs.
- Easy to see that the families of s–t paths and s–t cuts are a blocking pair.
  - WACP generalizes s–t cuts.

Paths and cuts block each other

- Set family $\mathcal{D}$ is a clutter if $R, S \in \mathcal{D}$, then $R \not\subset S$ and $S \not\subset R$ (edge sets of s–t cuts are a clutter).
- Define the blocker of $\mathcal{D}$, $B(\mathcal{D})$, to be the set of minimal subsets Q of E such that $Q \cap D \neq \emptyset \ \forall \ D \in \mathcal{D}$; thus $B(\mathcal{D})$ is also a clutter.
- Fact: $B(B(\mathcal{D})) = \mathcal{D}$, and so blockers come in dual pairs.
- Easy to see that the families of s–t paths and s–t cuts are a blocking pair.
  - WACP generalizes s–t cuts.
  - Hoffman also generalized packing of s–t paths (i.e., Max Flow) to *Weighted Abstract Flow* (WAF).

Paths and cuts block each other

- Set family $\mathcal{D}$ is a clutter if $R, S \in \mathcal{D}$, then $R \not\subset S$ and $S \not\subset R$ (edge sets of s–t cuts are a clutter).
- Define the blocker of $\mathcal{D}$, $B(\mathcal{D})$, to be the set of minimal subsets Q of E such that $Q \cap D \neq \emptyset \ \forall \ D \in \mathcal{D}$; thus $B(\mathcal{D})$ is also a clutter.
- Fact: $B(B(\mathcal{D})) = \mathcal{D}$, and so blockers come in dual pairs.
- Easy to see that the families of s–t paths and s–t cuts are a blocking pair.
  - WACP generalizes s–t cuts.
  - Hoffman also generalized packing of s–t paths (i.e., Max Flow) to *Weighted Abstract Flow* (WAF).

Theorem (Hoffman '78)

If $\mathcal{L}$ is a submodular clutter, then the blocker of $\mathcal{L}$ is an abstract path system.

Outline

1. Combinatorial Optimization
   - Packing problems
2. Hoffman's Models
   - Lattice Polyhedra
   - Blocking
3. Algorithms
   - Primal-Dual Algorithm
   - P-D for WACP
4. Conclusion
   - Open questions

How to compute integer optimal solutions?

- Max Flow and Shortest Path are important because we have efficient algorithms that compute integer optimal solutions.

How to compute integer optimal solutions?

- Max Flow and Shortest Path are important because we have efficient algorithms that compute integer optimal solutions.
- So, it's not enough to just know that integer optimal solutions exist (TDI), but we also need algorithms to compute them.

How to compute integer optimal solutions?

- Max Flow and Shortest Path are important because we have efficient algorithms that compute integer optimal solutions.
- So, it's not enough to just know that integer optimal solutions exist (TDI), but we also need algorithms to compute them.
- WAF: A weakly polynomial combinatorial algorithm was developed by Martens and Mc.

How to compute integer optimal solutions?

- Max Flow and Shortest Path are important because we have efficient algorithms that compute integer optimal solutions.
- So, it's not enough to just know that integer optimal solutions exist (TDI), but we also need algorithms to compute them.
- WAF: A weakly polynomial combinatorial algorithm was developed by Martens and Mc.
- WACP: The result here is a weakly polynomial combinatorial algorithm.

How to compute integer optimal solutions?

- Max Flow and Shortest Path are important because we have efficient algorithms that compute integer optimal solutions.
- So, it's not enough to just know that integer optimal solutions exist (TDI), but we also need algorithms to compute them.
- WAF: A weakly polynomial combinatorial algorithm was developed by Martens and Mc.
- WACP: The result here is a weakly polynomial combinatorial algorithm.
  - There was a previous algorithm for the case where r is monotone (i.e., $D \preceq Q \implies r_D \leq r_Q$) by Frank, but this does not cover important applications such as polymatroid intersection.

The Primal-Dual Algorithm

- Recall the Primal-Dual (Successive Shortest Path, SSP) Algorithm for max flow at min cost.

The Primal-Dual Algorithm

- Recall the Primal-Dual (Successive Shortest Path, SSP) Algorithm for max flow at min cost.
- It greedily pushes flow on the cheapest (shortest) augmenting path.

Primal-Dual Algorithm:

The Primal-Dual Algorithm

- Recall the Primal-Dual (Successive Shortest Path, SSP) Algorithm for max flow at min cost.
- It greedily pushes flow on the cheapest (shortest) augmenting path.

Primal-Dual Algorithm:

Set $x = 0$, $\pi = 0$.

The Primal-Dual Algorithm

- Recall the Primal-Dual (Successive Shortest Path, SSP) Algorithm for max flow at min cost.
- It greedily pushes flow on the cheapest (shortest) augmenting path.

Primal-Dual Algorithm:

Set $x = 0$, $\pi = 0$.
While augmenting paths remain do

End

The Primal-Dual Algorithm

- Recall the Primal-Dual (Successive Shortest Path, SSP) Algorithm for max flow at min cost.
- It greedily pushes flow on the cheapest (shortest) augmenting path.

Primal-Dual Algorithm:

Set $x = 0$, $\pi = 0$.
While augmenting paths remain do
  Use Shortest Path to compute the subnetwork $\mathcal{S}$
    of min-cost augmenting paths (dual change).

End

The Primal-Dual Algorithm

- Recall the Primal-Dual (Successive Shortest Path, SSP) Algorithm for max flow at min cost.
- It greedily pushes flow on the cheapest (shortest) augmenting path.

Primal-Dual Algorithm:

Set $x = 0$, $\pi = 0$.
While augmenting paths remain do
    Use Shortest Path to compute the subnetwork $\mathcal{S}$
        of min-cost augmenting paths (dual change).
    Use Max Flow to augment all paths in $\mathcal{S}$ (primal change).
End

The Primal-Dual Algorithm

- Recall the Primal-Dual (Successive Shortest Path, SSP) Algorithm for max flow at min cost.
- It greedily pushes flow on the cheapest (shortest) augmenting path.

Primal-Dual Algorithm:

Set $x = 0$, $\pi = 0$.
While augmenting paths remain do
    Use Shortest Path to compute the subnetwork $\mathcal{S}$
        of min-cost augmenting paths (dual change).
    Use Max Flow to augment all paths in $\mathcal{S}$ (primal change).
End

- Each iteration maintains that x and π are optimal for current flow value, so when x becomes a max flow, it is optimal.

A Technical Detail

- Complementary slackness $\implies$ if a primal variable > 0, the dual constraint must stay tight.

A Technical Detail

- Complementary slackness $\implies$ if a primal variable > 0, the dual constraint must stay tight.
- Thus P-D solves a restricted problem in inner iterations where some elements in R must stay tight.

A Technical Detail

- Complementary slackness $\implies$ if a primal variable > 0, the dual constraint must stay tight.
- Thus P-D solves a restricted problem in inner iterations where some elements in R must stay tight.
- But otherwise, the advantage of P-D is that it replaces the complicated objective $r^T y$ with a simple objective $\mathbb{1}^T y$.

A Technical Detail

- Complementary slackness $\implies$ if a primal variable > 0, the dual constraint must stay tight.
- Thus P-D solves a restricted problem in inner iterations where some elements in R must stay tight.
- But otherwise, the advantage of P-D is that it replaces the complicated objective $r^T y$ with a simple objective $\mathbb{1}^T y$.
- Due to R, the solution to the restricted dual could have -1 values in it, so the dual update need not be monotone.

P-D, TDI, and CPlex

Theorem (Applegate, Cook, Mc '91)

If a problem class is TDI, then P-D can be used to solve it while always maintaining integral solutions.

P-D, TDI, and CPlex

Theorem (Applegate, Cook, Mc '91)

If a problem class is TDI, then P-D can be used to solve it while always maintaining integral solutions.

Corollary

A conjecture of Barahona & Mahjoub on the TDI-ness of a feedback arc set formulation for K_5.

P-D, TDI, and CPlex

Theorem (Applegate, Cook, Mc '91)

If a problem class is TDI, then P-D can be used to solve it while always maintaining integral solutions.

Corollary

A conjecture of Barahona & Mahjoub on the TDI-ness of a feedback arc set formulation for K_5.

Proof.

Via LOPT 3.0, an early precursor to CPlex. □

P-D, TDI, and CPlex

Theorem (Applegate, Cook, Mc '91)

If a problem class is TDI, then P-D can be used to solve it while always maintaining integral solutions.

Corollary

A conjecture of Barahona & Mahjoub on the TDI-ness of a feedback arc set formulation for K_5.

Proof.

Via LOPT 3.0, an early precursor to CPlex. □

(This is the earliest paper I know using CPlex as a solver)

Overview

- $\max$ instead of $\min$ $\implies$ must start with max weight cuts.

Overview

- $\max$ instead of $\min$ $\implies$ must start with max weight cuts.
- Define λ as the weight of the current highest-reward cut; initially $\lambda = \max_D r_D = r_{\max}$.

Overview

- $\max$ instead of $\min \implies$ must start with max weight cuts.
- Define λ as the weight of the current highest-reward cut; initially $\lambda = \max_D r_D = r_{\max}$.
- Relax $x(D) \geq r_D$ to $x(D) \geq r_D - \lambda$.

Overview

- $\max$ instead of $\min$ $\implies$ must start with max weight cuts.
- Define λ as the weight of the current highest-reward cut; initially $\lambda = \max_D r_D = r_{\max}$.
- Relax $x(D) \geq r_D$ to $x(D) \geq r_D - \lambda$.
- [When $\lambda = r_{\max} + 1$, $x = y = 0$ is optimal.]

Overview

- $\max$ instead of $\min \implies$ must start with max weight cuts.
- Define λ as the weight of the current highest-reward cut; initially $\lambda = \max_D r_D = r_{\max}$.
- Relax $x(D) \geq r_D$ to $x(D) \geq r_D - \lambda$.
- [When $\lambda = r_{\max} + 1$, $x = y = 0$ is optimal.]
- Now decrease λ to 0, keeping optimality $\implies$ when $\lambda = 0$ we are optimal.

Overview

- $\max$ instead of $\min$ $\implies$ must start with max weight cuts.
- Define λ as the weight of the current highest-reward cut; initially $\lambda = \max_D r_D = r_{\max}$.
- Relax $x(D) \geq r_D$ to $x(D) \geq r_D - \lambda$.
- [When $\lambda = r_{\max} + 1$, $x = y = 0$ is optimal.]
- Now decrease λ to 0, keeping optimality $\implies$ when $\lambda = 0$ we are optimal.
- For fixed λ, focus on subnetwork of cuts with $\mathrm{gap}(D) = x(D) - r_D + \lambda = 0$.

Overview

- $\max$ instead of $\min$ $\implies$ must start with max weight cuts.
- Define λ as the weight of the current highest-reward cut; initially $\lambda = \max_D r_D = r_{\max}$.
- Relax $x(D) \geq r_D$ to $x(D) \geq r_D - \lambda$.
- [When $\lambda = r_{\max} + 1$, $x = y = 0$ is optimal.]
- Now decrease λ to 0, keeping optimality $\implies$ when $\lambda = 0$ we are optimal.
- For fixed λ, focus on subnetwork of cuts with $\text{gap}(D) = x(D) - r_D + \lambda = 0$.
- (implicitly get subnetwork via an oracle that gives any violating cuts $\implies$ Ellipsoid-polynomial)

Overview

- $\max$ instead of $\min$ $\implies$ must start with max weight cuts.
- Define λ as the weight of the current highest-reward cut; initially $\lambda = \max_D r_D = r_{\max}$.
- Relax $x(D) \geq r_D$ to $x(D) \geq r_D - \lambda$.
- [When $\lambda = r_{\max} + 1$, $x = y = 0$ is optimal.]
- Now decrease λ to 0, keeping optimality $\implies$ when $\lambda = 0$ we are optimal.
- For fixed λ, focus on subnetwork of cuts with $\mathrm{gap}(D) = x(D) - r_D + \lambda = 0$.
- (implicitly get subnetwork via an oracle that gives any violating cuts $\implies$ Ellipsoid-polynomial)
- Lemma: this subnetwork still satisfies the axioms.

Overview

- $\max$ instead of $\min \implies$ must start with max weight cuts.
- Define λ as the weight of the current highest-reward cut; initially $\lambda = \max_D r_D = r_{\max}$.
- Relax $x(D) \geq r_D$ to $x(D) \geq r_D - \lambda$.
- [When $\lambda = r_{\max} + 1$, $x = y = 0$ is optimal.]
- Now decrease λ to 0, keeping optimality $\implies$ when $\lambda = 0$ we are optimal.
- For fixed λ, focus on subnetwork of cuts with $\mathrm{gap}(D) = x(D) - r_D + \lambda = 0$.
- (implicitly get subnetwork via an oracle that gives any violating cuts $\implies$ Ellipsoid-polynomial)
- Lemma: this subnetwork still satisfies the axioms.
- *But* $R = \{e \mid x_e > 0\}$ is restricted to be tight, i.e., $\sum_{D \ni e} y_D = u_e$.

Restricted subnetwork

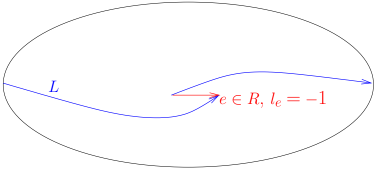
- Solve $\mathrm{gap}(D) = 0$ subnetwork using extension of A. Frank '99 Abstract SP.

Restricted subnetwork

- Solve $\mathrm{gap}(D) = 0$ subnetwork using extension of A. Frank '99 Abstract SP.
- Since restr. subnetwork is cut packing, it's blocked by a SP l.

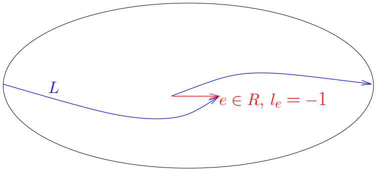
Restricted subnetwork

- Solve $\mathrm{gap}(D) = 0$ subnetwork using extension of A. Frank '99 Abstract SP.
- Since restr. subnetwork is cut packing, it's blocked by a SP l.
- Here l is 0, ± 1:



Restricted subnetwork

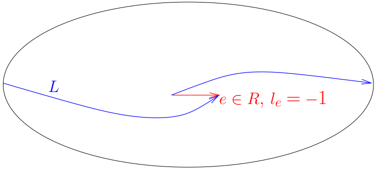
- Solve $\mathrm{gap}(D) = 0$ subnetwork using extension of A. Frank '99 Abstract SP.
- Since restr. subnetwork is cut packing, it's blocked by a SP l.
- Here l is 0, ± 1:



- Restricted subnetwork uses original y, auxiliary dual l.

Restricted subnetwork

- Solve $\mathrm{gap}(D) = 0$ subnetwork using extension of A. Frank '99 Abstract SP.
- Since restr. subnetwork is cut packing, it's blocked by a SP l.
- Here l is 0, ± 1:



- Restricted subnetwork uses original y, auxiliary dual l.
- Thus y is automatically updated.

Solving the restricted problem

1. Ensure that the cut packing y is a chain.

Solving the restricted problem

1. Ensure that the cut packing y is a chain.
2. Build an auxiliary (real) digraph G based on this chain. The restricted abstract shortest path problem turns out to be equivalent to a generalized shortest path problem on G.

Solving the restricted problem

1. Ensure that the cut packing y is a chain.
2. Build an auxiliary (real) digraph G based on this chain. The restricted abstract shortest path problem turns out to be equivalent to a generalized shortest path problem on G.
3. Find a generalized s-t path in G with incidence vector l using only elements of R.

Solving the restricted problem

1. Ensure that the cut packing y is a chain.
2. Build an auxiliary (real) digraph G based on this chain. The restricted abstract shortest path problem turns out to be equivalent to a generalized shortest path problem on G.
3. Find a generalized s-t path in G with incidence vector l using only elements of R.
4. If y and l are not complementary slack (i.e., if l is not tight on y's chain), update y along path l and return to Step 1.

Some details

What's going on here?

- Trying to get complementary slackness between y and l:

Some details

What's going on here?

- Trying to get complementary slackness between y and l:
  - If $y_D > 0$ (use cut D), $\sum_{e \in D} l_e = 1$ ("path" l crosses D only once), and conversely.

Some details

What's going on here?

- Trying to get complementary slackness between y and l:
  - If $y_D > 0$ (use cut D), $\sum_{e \in D} l_e = 1$ ("path" l crosses D only once), and conversely.
  - If $\sum_{D \ni e} y_D < u_e$ (cut D not tight), $l_e = 0$ (path l does not use e), and conversely.

Some details

What's going on here?

- Trying to get complementary slackness between y and l:
  - If $y_D > 0$ (use cut D), $\sum_{e \in D} l_e = 1$ ("path" l crosses D only once), and conversely.
  - If $\sum_{D \ni e} y_D < u_e$ (cut D not tight), $l_e = 0$ (path l does not use e), and conversely.
- $\mathcal{L}(\lambda)$ is modular and consecutive $\implies$ there is a sort of concrete s–t network underlying every y that's a chain.

Some details

What's going on here?

- Trying to get complementary slackness between y and l:
  - If $y_D > 0$ (use cut D), $\sum_{e \in D} l_e = 1$ ("path" l crosses D only once), and conversely.
  - If $\sum_{D \ni e} y_D < u_e$ (cut D not tight), $l_e = 0$ (path l does not use e), and conversely.
- $\mathcal{L}(\lambda)$ is modular and consecutive $\implies$ there is a sort of concrete s–t network underlying every y that's a chain.
- Try to find an s–t path in this network that is complementary slack with y via breadth-first search.

Some details

What's going on here?

- Trying to get complementary slackness between y and l:
  - If $y_D > 0$ (use cut D), $\sum_{e \in D} l_e = 1$ ("path" l crosses D only once), and conversely.
  - If $\sum_{D \ni e} y_D < u_e$ (cut D not tight), $l_e = 0$ (path l does not use e), and conversely.
- $\mathcal{L}(\lambda)$ is modular and consecutive $\implies$ there is a sort of concrete s–t network underlying every y that's a chain.
- Try to find an s–t path in this network that is complementary slack with y via breadth-first search.
- If the BFS is blocked, this tells you how to change y so it can advance.

Some details

What's going on here?

- Trying to get complementary slackness between y and l:
  - If $y_D > 0$ (use cut D), $\sum_{e \in D} l_e = 1$ ("path" l crosses D only once), and conversely.
  - If $\sum_{D \ni e} y_D < u_e$ (cut D not tight), $l_e = 0$ (path l does not use e), and conversely.
- $\mathcal{L}(\lambda)$ is modular and consecutive $\implies$ there is a sort of concrete s–t network underlying every y that's a chain.
- Try to find an s–t path in this network that is complementary slack with y via breadth-first search.
- If the BFS is blocked, this tells you how to change y so it can advance.
- This process is monotone, and so terminates in strongly polynomial time with CS solutions.

Updating θ

- Update $x' \longleftarrow x + \theta l$,
  $\lambda' \longleftarrow \lambda - \theta$
  $\implies \text{gap}'(D) \longleftarrow \text{gap}(D) + \theta(l(D) - 1)$.

Updating θ

- Update $x' \longleftarrow x + \theta l$,
  $\lambda' \longleftarrow \lambda - \theta$
  $\implies \text{gap}'(D) \longleftarrow \text{gap}(D) + \theta(l(D) - 1)$.
- Lemma: θ is always an integer.

Updating θ

- Update $x' \longleftarrow x + \theta l$,
  $\lambda' \longleftarrow \lambda - \theta$
  $\implies \text{gap}'(D) \longleftarrow \text{gap}(D) + \theta(l(D) - 1)$.
- Lemma: θ is always an integer.
- Knowing this, we can use binary search plus the oracle to find new value of θ s.t. $\text{gap}'(D) \geq 0 \ \forall \ D \in \mathcal{D}$.

Updating θ

- Update $x' \longleftarrow x + \theta l$,
  $\lambda' \longleftarrow \lambda - \theta$
  $\Longrightarrow \ \mathrm{gap}'(D) \longleftarrow \mathrm{gap}(D) + \theta(l(D) - 1)$.
- Lemma: θ is always an integer.
- Knowing this, we can use binary search plus the oracle to find new value of θ s.t. $\mathrm{gap}'(D) \geq 0 \ \forall \ D \in \mathcal{D}$.
- Also need to use oracle to “uncross” the new y.

Updating θ

- Update $x' \longleftarrow x + \theta l$,
  $\lambda' \longleftarrow \lambda - \theta$
  $\implies \text{gap}'(D) \longleftarrow \text{gap}(D) + \theta(l(D) - 1)$.
- Lemma: θ is always an integer.
- Knowing this, we can use binary search plus the oracle to find new value of θ s.t. $\text{gap}'(D) \geq 0 \ \forall \ D \in \mathcal{D}$.
- Also need to use oracle to "uncross" the new y.
- Corollary: new x and y are optimal for the new λ.

Complexity

- Each solve of Restr. Abstract Cut Pack is polynomial.

Complexity

- Each solve of Restr. Abstract Cut Pack is polynomial.
- x stays same at most n consecutive solves $\implies O(nr_{\max})$ solves.

Complexity

- Each solve of Restr. Abstract Cut Pack is polynomial.
- x stays same at most n consecutive solves $\implies O(nr_{\max})$ solves.
- This gives a *pseudo-polynomial* bound.

Complexity

- Each solve of Restr. Abstract Cut Pack is polynomial.
- x stays same at most n consecutive solves $\implies O(nr_{\max})$ solves.
- This gives a *pseudo-polynomial* bound.
- Make weakly polynomial via bit scaling.

Complexity

- Each solve of Restr. Abstract Cut Pack is polynomial.
- x stays same at most n consecutive solves $\implies O(nr_{\max})$ solves.
- This gives a *pseudo-polynomial* bound.
- Make weakly polynomial via bit scaling.
  - Not clear how to scale supermodular $r \implies$ scale u.

Complexity

- Each solve of Restr. Abstract Cut Pack is polynomial.
- x stays same at most n consecutive solves $\implies O(nr_{\max})$ solves.
- This gives a *pseudo-polynomial* bound.
- Make weakly polynomial via bit scaling.
  - Not clear how to scale supermodular $r \implies$ scale u.
  - Use standard trick of using one more bit of precision at each phase; doubling previous phase's y gives a good initial solution.

Complexity

- Each solve of Restr. Abstract Cut Pack is polynomial.
- x stays same at most n consecutive solves $\Longrightarrow$ $O(nr_{\max})$ solves.
- This gives a *pseudo-polynomial* bound.
- Make weakly polynomial via bit scaling.
  - Not clear how to scale supermodular r $\Longrightarrow$ scale u.
  - Use standard trick of using one more bit of precision at each phase; doubling previous phase's y gives a good initial solution.
  - Introduce new "1" bits one-by-one $\Longrightarrow$ need only to solve subproblems with $u_e \longleftarrow u_e + 1$ $\Longrightarrow$ computational sensitivity analysis.

Complexity

- Each solve of Restr. Abstract Cut Pack is polynomial.
- x stays same at most n consecutive solves $\Longrightarrow$ $O(nr_{\max})$ solves.
- This gives a *pseudo-polynomial* bound.
- Make weakly polynomial via bit scaling.
  - Not clear how to scale supermodular r $\Longrightarrow$ scale u.
  - Use standard trick of using one more bit of precision at each phase; doubling previous phase's y gives a good initial solution.
  - Introduce new "1" bits one-by-one $\Longrightarrow$ need only to solve subproblems with $u_e \longleftarrow u_e + 1$ $\Longrightarrow$ computational sensitivity analysis.
  - Same tools apply, but now are strongly polynomial.

Complexity

- Each solve of Restr. Abstract Cut Pack is polynomial.
- x stays same at most n consecutive solves $\implies O(nr_{\max})$ solves.
- This gives a *pseudo-polynomial* bound.
- Make weakly polynomial via bit scaling.
  - Not clear how to scale supermodular $r \implies$ scale u.
  - Use standard trick of using one more bit of precision at each phase; doubling previous phase's y gives a good initial solution.
  - Introduce new "1" bits one-by-one $\implies$ need only to solve subproblems with $u_e \longleftarrow u_e + 1 \implies$ computational sensitivity analysis.
  - Same tools apply, but now are strongly polynomial.
- Theorem: This algorithm solves Weighted Abstract Cut Packing in $O((m \log C + m^2 \log r_{\max}(m + \mathrm{CO}))(m \cdot \mathrm{CO} + h(m + h)))$ (weakly polynomial) time ("CO" is # oracle calls; h is height of lattice, C is max u_e).

Outline

1. Combinatorial Optimization
   - Packing problems
2. Hoffman's Models
   - Lattice Polyhedra
   - Blocking
3. Algorithms
   - Primal-Dual Algorithm
   - P-D for WACP
4. Conclusion
   - Open questions

Conclusion

1. Much the same P-D framework was used for the WAF algorithm.

Conclusion

1. Much the same P-D framework was used for the WAF algorithm.
2. If you are interested in algorithms for combinatorial optimization problems, a good place to look is at problems that have Ellipsoid but not (yet) combinatorial algorithms

Conclusion

1. Much the same P-D framework was used for the WAF algorithm.
2. If you are interested in algorithms for combinatorial optimization problems, a good place to look is at problems that have Ellipsoid but not (yet) combinatorial algorithms
   - ...such as optimizing over the Subtour Elimination Polytope for TSP

Conclusion

1. Much the same P-D framework was used for the WAF algorithm.
2. If you are interested in algorithms for combinatorial optimization problems, a good place to look is at problems that have Ellipsoid but not (yet) combinatorial algorithms
   - …such as optimizing over the Subtour Elimination Polytope for TSP
3. Can we get a combinatorial faster, or even strongly polynomial algorithm? Maybe some version of *Min Mean Cycle*?

Conclusion

1. Much the same P-D framework was used for the WAF algorithm.
2. If you are interested in algorithms for combinatorial optimization problems, a good place to look is at problems that have Ellipsoid but not (yet) combinatorial algorithms
   - . . . such as optimizing over the Subtour Elimination Polytope for TSP
3. Can we get a combinatorial faster, or even strongly polynomial algorithm? Maybe some version of *Min Mean Cycle*?
4. Typically for such problems, figuring out how to represent the problem is a big hurdle; here we suppressed details of the oracles we are using.

Conclusion

1. Much the same P-D framework was used for the WAF algorithm.
2. If you are interested in algorithms for combinatorial optimization problems, a good place to look is at problems that have Ellipsoid but not (yet) combinatorial algorithms
   - ...such as optimizing over the Subtour Elimination Polytope for TSP
3. Can we get a combinatorial faster, or even strongly polynomial algorithm? Maybe some version of *Min Mean Cycle*?
4. Typically for such problems, figuring out how to represent the problem is a big hurdle; here we suppressed details of the oracles we are using.
5. Gröflin and Hoffman extended lattice polyhedra to 0, ± 1 matrices and to a version with sub- and super-modular interchanged; can we adapt our algorithm for these?

Conclusion

1. Much the same P-D framework was used for the WAF algorithm.
2. If you are interested in algorithms for combinatorial optimization problems, a good place to look is at problems that have Ellipsoid but not (yet) combinatorial algorithms
   - ...such as optimizing over the Subtour Elimination Polytope for TSP
3. Can we get a combinatorial faster, or even strongly polynomial algorithm? Maybe some version of *Min Mean Cycle*?
4. Typically for such problems, figuring out how to represent the problem is a big hurdle; here we suppressed details of the oracles we are using.
5. Gröflin and Hoffman extended lattice polyhedra to 0, ± 1 matrices and to a version with sub- and super-modular interchanged; can we adapt our algorithm for these?
6. Could we further extend this idea to solve, e.g., Schrijver's general framework for TDI problems?

Conclusion

1. Much the same P-D framework was used for the WAF algorithm.
2. If you are interested in algorithms for combinatorial optimization problems, a good place to look is at problems that have Ellipsoid but not (yet) combinatorial algorithms
   - …such as optimizing over the Subtour Elimination Polytope for TSP
3. Can we get a combinatorial faster, or even strongly polynomial algorithm? Maybe some version of *Min Mean Cycle*?
4. Typically for such problems, figuring out how to represent the problem is a big hurdle; here we suppressed details of the oracles we are using.
5. Gröflin and Hoffman extended lattice polyhedra to 0, ± 1 matrices and to a version with sub- and super-modular interchanged; can we adapt our algorithm for these?
6. Could we further extend this idea to solve, e.g., Schrijver's general framework for TDI problems?
7. Is there a good blocking dual to Schrijver's framework?

Any questions?

Questions?

Comments?