LINEAR PROGRAMMING

[V. CH10]: THE TRAVELING SALESMAN PROBLEM

Phillip Keldenich   Ahmad Moradi

Department of Computer Science
Algorithms Department
TU Braunschweig

January 31, 2023

WHAT? WHY? HOW?

CUTTING PLANES

EXCLUDING EDGES

NUMERICAL ISSUES

WHAT (PRECISELY) IS THE TSP?

In the TSP, we are given a set of n *cities* V with distances $d(i, j)$ between each other.
Our goal is to find a shortest tour that visits each city and returns to the starting point.
In our (actually, the most common) version:

WHAT (PRECISELY) IS THE TSP?

In the TSP, we are given a set of n *cities* V with distances $d(i, j)$ between each other.
Our goal is to find a shortest tour that visits each city and returns to the starting point.
In our (actually, the most common) version:

- the underlying graph $G = (V, E)$ is complete, i.e., we can directly go from any v to any w,

WHAT (PRECISELY) IS THE TSP?

In the TSP, we are given a set of n *cities* V with distances $d(i,j)$ between each other.
Our goal is to find a shortest tour that visits each city and returns to the starting point.
In our (actually, the most common) version:

- the underlying graph $G = (V, E)$ is complete, i.e., we can directly go from any v to any w,
- the distances are symmetric, i.e., $d(i,j) = d(j,i)$, i.e., the graph is undirected,

WHAT (PRECISELY) IS THE TSP?

In the TSP, we are given a set of n *cities* V with distances $d(i, j)$ between each other.
Our goal is to find a shortest tour that visits each city and returns to the starting point.
In our (actually, the most common) version:

- the underlying graph $G = (V, E)$ is complete, i.e., we can directly go from any v to any w,
- the distances are symmetric, i.e., $d(i, j) = d(j, i)$, i.e., the graph is undirected,
- we often assume the triangle inequality, i.e., $d(i, k) \leq d(i, j) + d(j, k)$.

Why the TSP?

The TSP is a very natural, fundamental NP-complete problem.
Direct practical applications are rather rare, but they do occur.
Closely related to very relevant problems, e.g., vehicle routing with various extensions.

WHY THE TSP?

The TSP is a very natural, fundamental NP-complete problem.
Direct practical applications are rather rare, but they do occur.
Closely related to very relevant problems, e.g., vehicle routing with various extensions.

Most important reason for studying it here:
It is one of the main drivers behind the development and improvement of IP solving, and IP solving is exceptionally successful for the TSP. Many ideas that improved (M)IP solvers over the years have been first developed and applied in the context of the TSP.

TSP Model

There are multiple TSP models. The practical one is:

TSP Model

There are multiple TSP models. The practical one is:

$$\min \sum_{vw \in E} d(v,w) x_{vw}, \text{ s.t.}$$

TSP MODEL

There are multiple TSP models. The practical one is:

$$\min \sum_{vw \in E} d(v,w) x_{vw}, \text{ s.t.}$$

$$\forall v \in V : \sum_{e \in \delta(\{v\})} x_e = 2,$$

$\delta(A)$ contains exactly all edges between a vertex $v \in A$ and $w \notin A$.

TSP MODEL

There are multiple TSP models. The practical one is:

$$\min \sum_{vw \in E} d(v,w) x_{vw}, \text{ s.t.}$$

$$\forall v \in V : \sum_{e \in \delta(\{v\})} x_e = 2,$$

$$\forall S \neq \emptyset, S \subsetneq V : \sum_{e \in \delta(S)} x_e \geq 2,$$

$\delta(A)$ contains exactly all edges between a vertex $v \in A$ and $w \notin A$.

TSP MODEL

There are multiple TSP models. The practical one is:

$$\min \sum_{vw \in E} d(v,w) x_{vw}, \text{ s.t.}$$

$$\forall v \in V : \sum_{e \in \delta(\{v\})} x_e = 2,$$

$$\forall S \neq \emptyset, S \subsetneq V : \sum_{e \in \delta(S)} x_e \geq 2,$$

$$\forall e : x_e \in \{0,1\}.$$

$\delta(A)$ contains exactly all edges between a vertex $v \in A$ and $w \notin A$.

TSP MODEL

There are multiple TSP models. The practical one is:

$$\min \sum_{vw \in E} d(v,w)x_{vw}, \text{ s.t.}$$

$$\forall v \in V : \sum_{e \in \delta(\{v\})} x_e = 2,$$

$$\forall S \neq \emptyset, S \subsetneq V : \sum_{e \in \delta(S)} x_e \geq 2,$$

$$\forall e : x_e \in \{0,1\}.$$

$\delta(A)$ contains exactly all edges between a vertex $v \in A$ and $w \notin A$.

Notes:

- The model has $\Omega(2^n)$ constraints (because of the last type of constraint).

TSP MODEL

There are multiple TSP models. The practical one is:

$$\min \sum_{vw \in E} d(v,w) x_{vw}, \text{ s.t.}$$

$$\forall v \in V : \sum_{e \in \delta(\{v\})} x_e = 2,$$

$$\forall S \neq \emptyset, S \subsetneq V : \sum_{e \in \delta(S)} x_e \geq 2,$$

$$\forall e : x_e \in \{0,1\}.$$

$\delta(A)$ contains exactly all edges between a vertex $v \in A$ and $w \notin A$.

Notes:

- The model has $\Omega(2^n)$ constraints (because of the last type of constraint).
- These constraints are called *Subtour Elimination Constraints*.

TSP MODEL

There are multiple TSP models. The practical one is:

$$\min \sum_{vw \in E} d(v,w) x_{vw}, \text{ s.t.}$$

$$\forall v \in V : \sum_{e \in \delta(\{v\})} x_e = 2,$$

$$\forall S \neq \emptyset, S \subsetneq V : \sum_{e \in \delta(S)} x_e \geq 2,$$

$$\forall e : x_e \in \{0,1\}.$$

$\delta(A)$ contains exactly all edges between a vertex $v \in A$ and $w \notin A$.

Notes:

- The model has $\Omega(2^n)$ constraints (because of the last type of constraint).
- These constraints are called *Subtour Elimination Constraints*.
- They are not normally added in the beginning; instead, one handles them *lazily*.

TSP MODEL

There are multiple TSP models. The practical one is:

$$\min \sum_{vw \in E} d(v,w) x_{vw}, \text{ s.t.}$$

$$\forall v \in V: \sum_{e \in \delta(\{v\})} x_e = 2,$$

$$\forall S \neq \emptyset, S \subsetneq V: \sum_{e \in \delta(S)} x_e \geq 2,$$

$$\forall e: x_e \in \{0,1\}.$$

$\delta(A)$ contains exactly all edges between a vertex $v \in A$ and $w \notin A$.

Notes:

- The model has $\Omega(2^n)$ constraints (because of the last type of constraint).
- These constraints are called *Subtour Elimination Constraints*.
- They are not normally added in the beginning; instead, one handles them *lazily*.
- That means we add *violated* subtour elimination constraints as needed while solving.

OBSTACLES

A rather simple solver based on this model (seen in the exercise) is enough to solve most instances with around 500 cities.

OBSTACLES

A rather simple solver based on this model (seen in the exercise) is enough to solve most instances with around 500 cities. The best TSP solver Concorde is based on this model as well.

OBSTACLES

A rather simple solver based on this model (seen in the exercise) is enough to solve most instances with around 500 cities. The best TSP solver Concorde is based on this model as well. However, it could deal (with lots of CPU time) with at least 85900 cities. We have the following obstacles to deal with when we increase the number of cities much further.

OBSTACLES

A rather simple solver based on this model (seen in the exercise) is enough to solve most instances with around 500 cities. The best TSP solver Concorde is based on this model as well. However, it could deal (with lots of CPU time) with at least 85900 cities. We have the following obstacles to deal with when we increase the number of cities much further.

- We have to deal with lazily finding violated subtour elimination constraints.

Obstacles

A rather simple solver based on this model (seen in the exercise) is enough to solve most instances with around 500 cities. The best TSP solver Concorde is based on this model as well. However, it could deal (with lots of CPU time) with at least 85900 cities. We have the following obstacles to deal with when we increase the number of cities much further.

- We have to deal with lazily finding violated subtour elimination constraints.
- The LP relaxation of our formulation is usually quite strong. However, we must improve it with cutting planes to tackle larger and more complicated instances.

OBSTACLES

A rather simple solver based on this model (seen in the exercise) is enough to solve most instances with around 500 cities. The best TSP solver Concorde is based on this model as well. However, it could deal (with lots of CPU time) with at least 85900 cities. We have the following obstacles to deal with when we increase the number of cities much further.

- We have to deal with lazily finding violated subtour elimination constraints.
- The LP relaxation of our formulation is usually quite strong. However, we must improve it with cutting planes to tackle larger and more complicated instances.
- The number of edges is $\Omega(n^2)$; the number of non-zeros in our matrix can be even larger.

OBSTACLES

A rather simple solver based on this model (seen in the exercise) is enough to solve most instances with around 500 cities. The best TSP solver Concorde is based on this model as well. However, it could deal (with lots of CPU time) with at least 85900 cities. We have the following obstacles to deal with when we increase the number of cities much further.

- We have to deal with lazily finding violated subtour elimination constraints.
- The LP relaxation of our formulation is usually quite strong. However, we must improve it with cutting planes to tackle larger and more complicated instances.
- The number of edges is $\Omega(n^2)$; the number of non-zeros in our matrix can be even larger.
  - This means we need too much memory!

OBSTACLES

A rather simple solver based on this model (seen in the exercise) is enough to solve most instances with around 500 cities. The best TSP solver Concorde is based on this model as well. However, it could deal (with lots of CPU time) with at least 85900 cities. We have the following obstacles to deal with when we increase the number of cities much further.

- We have to deal with lazily finding violated subtour elimination constraints.
- The LP relaxation of our formulation is usually quite strong. However, we must improve it with cutting planes to tackle larger and more complicated instances.
- The number of edges is $\Omega(n^2)$; the number of non-zeros in our matrix can be even larger.
  - This means we need too much memory!
  - Solving the LPs becomes time-consuming!

OBSTACLES

A rather simple solver based on this model (seen in the exercise) is enough to solve most instances with around 500 cities. The best TSP solver Concorde is based on this model as well. However, it could deal (with lots of CPU time) with at least 85900 cities. We have the following obstacles to deal with when we increase the number of cities much further.

- We have to deal with lazily finding violated subtour elimination constraints.
- The LP relaxation of our formulation is usually quite strong. However, we must improve it with cutting planes to tackle larger and more complicated instances.
- The number of edges is $\Omega(n^2)$; the number of non-zeros in our matrix can be even larger.
  - This means we need too much memory!
  - Solving the LPs becomes time-consuming!
  - Intuitively, most edges seem useless!

OBSTACLES

A rather simple solver based on this model (seen in the exercise) is enough to solve most instances with around 500 cities. The best TSP solver Concorde is based on this model as well. However, it could deal (with lots of CPU time) with at least 85900 cities. We have the following obstacles to deal with when we increase the number of cities much further.

- We have to deal with lazily finding violated subtour elimination constraints.
- The LP relaxation of our formulation is usually quite strong. However, we must improve it with cutting planes to tackle larger and more complicated instances.
- The number of edges is $\Omega(n^2)$; the number of non-zeros in our matrix can be even larger.
  - This means we need too much memory!
  - Solving the LPs becomes time-consuming!
  - Intuitively, most edges seem useless!
  - We need to be able to dynamically add and remove edges (i.e., variables!) from consideration.

OBSTACLES

A rather simple solver based on this model (seen in the exercise) is enough to solve most instances with around 500 cities. The best TSP solver Concorde is based on this model as well. However, it could deal (with lots of CPU time) with at least 85900 cities. We have the following obstacles to deal with when we increase the number of cities much further.

- We have to deal with lazily finding violated subtour elimination constraints.
- The LP relaxation of our formulation is usually quite strong. However, we must improve it with cutting planes to tackle larger and more complicated instances.
- The number of edges is $\Omega(n^2)$; the number of non-zeros in our matrix can be even larger.
  - This means we need too much memory!
  - Solving the LPs becomes time-consuming!
  - Intuitively, most edges seem useless!
  - We need to be able to dynamically add and remove edges (i.e., variables!) from consideration.
  - Without losing provable optimality!

Obstacles

A rather simple solver based on this model (seen in the exercise) is enough to solve most instances with around 500 cities. The best TSP solver Concorde is based on this model as well. However, it could deal (with lots of CPU time) with at least 85900 cities. We have the following obstacles to deal with when we increase the number of cities much further.

- We have to deal with lazily finding violated subtour elimination constraints.
- The LP relaxation of our formulation is usually quite strong. However, we must improve it with cutting planes to tackle larger and more complicated instances.
- The number of edges is $\Omega(n^2)$; the number of non-zeros in our matrix can be even larger.
  - This means we need too much memory!
  - Solving the LPs becomes time-consuming!
  - Intuitively, most edges seem useless!
  - We need to be able to dynamically add and remove edges (i.e., variables!) from consideration.
  - Without losing provable optimality!
- Numerical issues can cause solutions obtained from our LP solver to not be strictly feasible/optimal. If we really want provable optimality, we have to deal with this (often ignored) problem.

Subtour Elimination Constraints

We can think about subtour elimination constraints just like cutting planes:

Subtour Elimination Constraints

We can think about subtour elimination constraints just like cutting planes:

- They are added in the same place in the search — after solving a linear relaxation.

Subtour Elimination Constraints

We can think about subtour elimination constraints just like cutting planes:

- They are added in the same place in the search — after solving a linear relaxation.
- They are both constraints satisfied by all (integer) tours.

Subtour Elimination Constraints

We can think about subtour elimination constraints just like cutting planes:

- They are added in the same place in the search — after solving a linear relaxation.
- They are both constraints satisfied by all (integer) tours.
- After adding constraints in a node, the linear relaxation is re-solved.

Subtour Elimination Constraints

We can think about subtour elimination constraints just like cutting planes:

- They are added in the same place in the search — after solving a linear relaxation.
- They are both constraints satisfied by all (integer) tours.
- After adding constraints in a node, the linear relaxation is re-solved.
- After re-solving, if we are not integral, we either continue adding constraints, or we branch.

Subtour Elimination Constraints

We can think about subtour elimination constraints just like cutting planes:

- They are added in the same place in the search — after solving a linear relaxation.
- They are both constraints satisfied by all (integer) tours.
- After adding constraints in a node, the linear relaxation is re-solved.
- After re-solving, if we are not integral, we either continue adding constraints, or we branch.

The *separation problem* for subtour elimination problems asks:
Given a solution to our model, is there a subtour constraint violated by the solution?

Subtour Elimination Constraints

We can think about subtour elimination constraints just like cutting planes:

- They are added in the same place in the search — after solving a linear relaxation.
- They are both constraints satisfied by all (integer) tours.
- After adding constraints in a node, the linear relaxation is re-solved.
- After re-solving, if we are not integral, we either continue adding constraints, or we branch.

The *separation problem* for subtour elimination problems asks:
Given a solution to our model, is there a subtour constraint violated by the solution?

For integer solutions, this is easy: Just check if there is more than one component!

Subtour Elimination Constraints

We can think about subtour elimination constraints just like cutting planes:

- They are added in the same place in the search — after solving a linear relaxation.
- They are both constraints satisfied by all (integer) tours.
- After adding constraints in a node, the linear relaxation is re-solved.
- After re-solving, if we are not integral, we either continue adding constraints, or we branch.

The *separation problem* for subtour elimination problems asks:
Given a solution to our model, is there a subtour constraint violated by the solution?

For integer solutions, this is easy: Just check if there is more than one component!

What about non-integer solutions?

SEPARATION PROBLEMS

In general, a *separation problem* consists of the following task.

Separation Problems

In general, a *separation problem* consists of the following task.

- We are given a solution to some *relaxation*:
  a solution to a problem with missing constraints.

SEPARATION PROBLEMS

In general, a *separation problem* consists of the following task.

- We are given a solution to some *relaxation*:
  a solution to a problem with missing constraints.
- We want to find out if there is a missing constraint violated by the solution.

SEPARATION PROBLEMS

In general, a *separation problem* consists of the following task.

- We are given a solution to some *relaxation*:
  a solution to a problem with missing constraints.
- We want to find out if there is a missing constraint violated by the solution.
- If there is, we also want to find a suitable violated constraint that we can add to our relaxation.

Separation Problems

In general, a *separation problem* consists of the following task.

- We are given a solution to some *relaxation*:
  a solution to a problem with missing constraints.
- We want to find out if there is a missing constraint violated by the solution.
- If there is, we also want to find a suitable violated constraint that we can add to our relaxation.

One important separation problem is the general separation problem for (M)IPs.

SEPARATION PROBLEMS

In general, a *separation problem* consists of the following task.

- We are given a solution to some *relaxation*:
  a solution to a problem with missing constraints.
- We want to find out if there is a missing constraint violated by the solution.
- If there is, we also want to find a suitable violated constraint that we can add to our relaxation.

One important separation problem is the general separation problem for (M)IPs.
In it, the missing constraints are the integrality constraints, the given solution is the solution to the linear relaxation, and the violated constraint must be a linear inequality.

SEPARATION PROBLEMS

In general, a *separation problem* consists of the following task.

- We are given a solution to some *relaxation*:
  a solution to a problem with missing constraints.
- We want to find out if there is a missing constraint violated by the solution.
- If there is, we also want to find a suitable violated constraint that we can add to our relaxation.

One important separation problem is the general separation problem for (M)IPs.
In it, the missing constraints are the integrality constraints, the given solution is the solution to the linear relaxation, and the violated constraint must be a linear inequality.

Separation Oracle Method:
If we can solve this problem in polynomial time for *any* given solution, we can solve the corresponding (M)IP in polynomial time.

SEPARATION PROBLEMS

In general, a *separation problem* consists of the following task.

- We are given a solution to some *relaxation*:
  a solution to a problem with missing constraints.
- We want to find out if there is a missing constraint violated by the solution.
- If there is, we also want to find a suitable violated constraint that we can add to our relaxation.

One important separation problem is the general separation problem for (M)IPs.
In it, the missing constraints are the integrality constraints, the given solution is the solution to the linear relaxation, and the violated constraint must be a linear inequality.

Separation Oracle Method:
If we can solve this problem in polynomial time for *any* given solution, we can solve the corresponding (M)IP in polynomial time.

There are also separation problems for specific types of cutting planes, e.g., for Gomory Cuts, Subtour Elimination Constraints, . . . ; those can sometimes be solved efficiently (and are often only solved heuristically).

Subtour Elimination Constraints — Separation Problem

The separation problem can be interpreted as follows:
Given a weighted graph $H = (V, E', w)$ with $w(e) = x_e^*$ and $E' = \{e \in E : w(e) > 0\}$, find some $\emptyset \neq S \subsetneq V$ such that $\sum_{e \in \delta(S)} w(e) < 2$, or find out no such set exists.

Subtour Elimination Constraints — Separation Problem

The separation problem can be interpreted as follows:
Given a weighted graph $H = (V, E', w)$ with $w(e) = x_e^*$ and $E' = \{e \in E : w(e) > 0\}$, find some $\emptyset \neq S \subsetneq V$ such that $\sum_{e \in \delta(S)} w(e) < 2$, or find out no such set exists.

Example in an interactive tool: https://www.math.uwaterloo.ca/tsp/app/diy.html

Subtour Elimination Constraints — Options for Separation

To detect violated subtour constraints, we have the following options:

Subtour Elimination Constraints — Options for Separation

To detect violated subtour constraints, we have the following options:

- Look for connected components (linear time).

Subtour Elimination Constraints — Options for Separation

To detect violated subtour constraints, we have the following options:

- Look for connected components (linear time).
- Look for biconnected components (linear time).

SUBTOUR ELIMINATION CONSTRAINTS — OPTIONS FOR SEPARATION

To detect violated subtour constraints, we have the following options:

- Look for connected components (linear time).
- Look for biconnected components (linear time).
- Using some source s and some sink t, find a minimum s-t-cut (almost linear time).

Subtour Elimination Constraints — Options for Separation

To detect violated subtour constraints, we have the following options:

- Look for connected components (linear time).
- Look for biconnected components (linear time).
- Using some source s and some sink t, find a minimum s-t-cut (almost linear time).
- Find a global minimum graph cut using, e.g., the Stoer-Wagner algorithm ($O(mn + n^2 \log n)$).

SUBTOUR ELIMINATION CONSTRAINTS — OPTIONS FOR SEPARATION

To detect violated subtour constraints, we have the following options:

- Look for connected components (linear time).
- Look for biconnected components (linear time).
- Using some source s and some sink t, find a minimum s-t-cut (almost linear time).
- Find a global minimum graph cut using, e.g., the Stoer-Wagner algorithm ($O(mn + n^2 \log n)$).

A global minimum graph cut below 2 corresponds to a violated subtour constraint and vice versa; we can solve the separation problem exactly in polynomial time!

Subtour Elimination Constraints — Options for Separation

To detect violated subtour constraints, we have the following options:

- Look for connected components (linear time).
- Look for biconnected components (linear time).
- Using some source s and some sink t, find a minimum s-t-cut (almost linear time).
- Find a global minimum graph cut using, e.g., the Stoer-Wagner algorithm ($O(mn + n^2 \log n)$).

A global minimum graph cut below 2 corresponds to a violated subtour constraint and vice versa; we can solve the separation problem exactly in polynomial time!

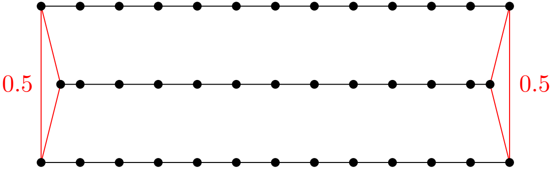
Usually, we cannot afford to do minimum graph cut computations all the time; we go through the list from the top and use the more expensive methods sparingly.

Further Cutting Planes

This is not all — there are more cutting plane families for tours!

Further Cutting Planes

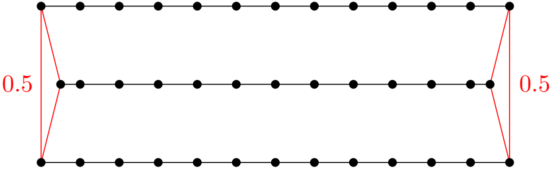
This is not all — there are more cutting plane families for tours!
Consider this example:



Are there any violated subtour constraints?

Further Cutting Planes

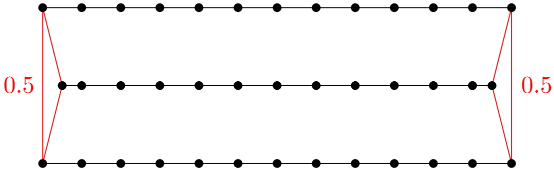
This is not all — there are more cutting plane families for tours!
Consider this example:



Are there any violated subtour constraints? No!

FURTHER CUTTING PLANES

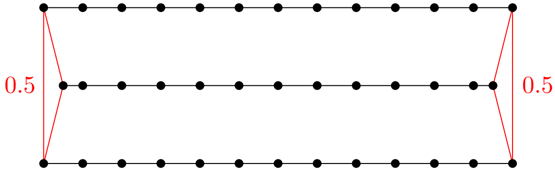
This is not all — there are more cutting plane families for tours!
Consider this example:



Are there any violated subtour constraints? No!
How can we cut off solutions such as this by cutting planes?

Further Cutting Planes

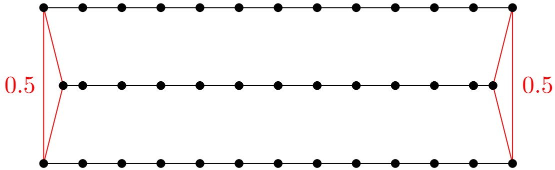
This is not all — there are more cutting plane families for tours!
Consider this example:



Are there any violated subtour constraints? No!
How can we cut off solutions such as this by cutting planes?
There is an important cutting plane family called *combs*.

Further Cutting Planes

This is not all — there are more cutting plane families for tours!
Consider this example:

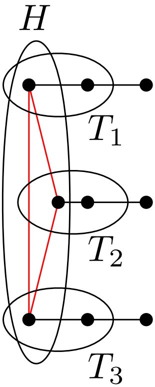


Are there any violated subtour constraints? No!
How can we cut off solutions such as this by cutting planes?
There is an important cutting plane family called *combs*.

For generalized versions of the above example, the length of the optimal tour is about $4/3$ of the optimal value of the LP relaxation including all subtours. The $4/3$ conjecture states that this is the worst case, i.e., for all sets of cities, the factor between the optimal integer solution and the optimal fractional solution is at most $4/3$. In general, this factor is called *integrality gap*.

COMB INEQUALITIES

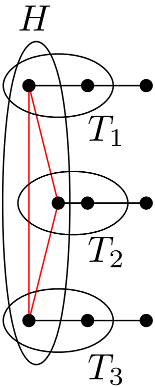
Consider a vertex set H (*handle*) and k sets $T_1, \ldots, T_k$ (*teeth*) with:



Comb Inequalities

Consider a vertex set H (*handle*) and k sets $T_1, \ldots, T_k$ (*teeth*) with:

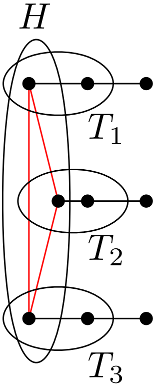
- $H \cap T_i \neq \emptyset$ for all i, i.e., every tooth has a vertex in the handle,



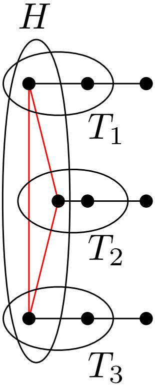
COMB INEQUALITIES

Consider a vertex set H (*handle*) and k sets $T_1, \ldots, T_k$ (*teeth*) with:

- $H \cap T_i \neq \emptyset$ for all i, i.e., every tooth has a vertex in the handle,
- $T_i \setminus H \neq \emptyset$ for all i, i.e., every tooth has a vertex outset the handle,



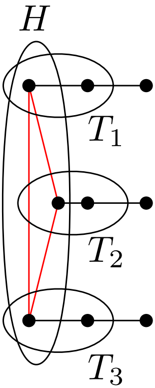
COMB INEQUALITIES



Consider a vertex set H (*handle*) and k sets $T_1, \dots, T_k$ (*teeth*) with:

- $H \cap T_i \neq \emptyset$ for all i, i.e., every tooth has a vertex in the handle,
- $T_i \setminus H \neq \emptyset$ for all i, i.e., every tooth has a vertex outset the handle,
- $T_i \cap T_j = \emptyset$ for all $i \neq j$, i.e., the teeth are disjoint,

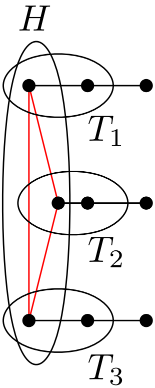
COMB INEQUALITIES



Consider a vertex set H (*handle*) and k sets $T_1, \dots, T_k$ (*teeth*) with:

- $H \cap T_i \neq \emptyset$ for all i, i.e., every tooth has a vertex in the handle,
- $T_i \setminus H \neq \emptyset$ for all i, i.e., every tooth has a vertex outset the handle,
- $T_i \cap T_j = \emptyset$ for all $i \neq j$, i.e., the teeth are disjoint,
- $k \geq 3$, and k is odd.

COMB INEQUALITIES

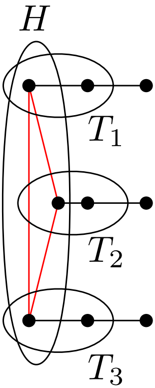


Consider a vertex set H (*handle*) and k sets $T_1, \dots, T_k$ (*teeth*) with:

- $H \cap T_i \neq \emptyset$ for all i, i.e., every tooth has a vertex in the handle,
- $T_i \setminus H \neq \emptyset$ for all i, i.e., every tooth has a vertex outset the handle,
- $T_i \cap T_j = \emptyset$ for all $i \neq j$, i.e., the teeth are disjoint,
- $k \geq 3$, and k is odd.

We analyze $$\underbrace{\sum_{e \in \delta(H)} x_e}_{P_1} + \underbrace{\sum_{i=1}^{k} \sum_{e \in \delta(T_i)} x_e}_{P_2}.$$

COMB INEQUALITIES



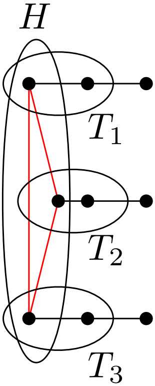
Consider a vertex set H (*handle*) and k sets $T_1, \ldots, T_k$ (*teeth*) with:

- $H \cap T_i \neq \emptyset$ for all i, i.e., every tooth has a vertex in the handle,
- $T_i \setminus H \neq \emptyset$ for all i, i.e., every tooth has a vertex outset the handle,
- $T_i \cap T_j = \emptyset$ for all $i \neq j$, i.e., the teeth are disjoint,
- $k \geq 3$, and k is odd.

We analyze $\underbrace{\sum_{e \in \delta(H)} x_e}_{P_1} + \underbrace{\sum_{i=1}^{k} \sum_{e \in \delta(T_i)} x_e}_{P_2}.$

- Let τ be any tour.

COMB INEQUALITIES



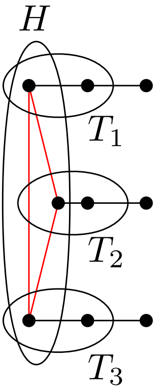
Consider a vertex set H (*handle*) and k sets $T_1, \ldots, T_k$ (*teeth*) with:

- $H \cap T_i \neq \emptyset$ for all i, i.e., every tooth has a vertex in the handle,
- $T_i \setminus H \neq \emptyset$ for all i, i.e., every tooth has a vertex outset the handle,
- $T_i \cap T_j = \emptyset$ for all $i \neq j$, i.e., the teeth are disjoint,
- $k \geq 3$, and k is odd.

We analyze $\underbrace{\sum_{e \in \delta(H)} x_e}_{P_1} + \underbrace{\sum_{i=1}^{k} \sum_{e \in \delta(T_i)} x_e}_{P_2}.$

- Let τ be any tour.
- For each tooth T_i, we have two cases: (1) τ has an edge going from $T_i \cap H$ to $T_i \setminus H$, or (2) it has no such edge.

COMB INEQUALITIES



Consider a vertex set H (*handle*) and k sets $T_1, \dots, T_k$ (*teeth*) with:

- $H \cap T_i \neq \emptyset$ for all i, i.e., every tooth has a vertex in the handle,
- $T_i \setminus H \neq \emptyset$ for all i, i.e., every tooth has a vertex outset the handle,
- $T_i \cap T_j = \emptyset$ for all $i \neq j$, i.e., the teeth are disjoint,
- $k \geq 3$, and k is odd.

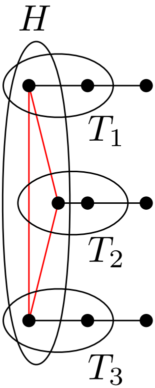
We analyze $\underbrace{\sum_{e \in \delta(H)} x_e}_{P_1} + \underbrace{\sum_{i=1}^{k} \sum_{e \in \delta(T_i)} x_e}_{P_2}.$

- Let τ be any tour.
- For each tooth T_i, we have two cases: (1) τ has an edge going from $T_i \cap H$ to $T_i \setminus H$, or (2) it has no such edge.
- In case (1), T_i contributes one edge to P_1 and two edges to P_2, i.e., at least three edges in total.

COMB INEQUALITIES

Consider a vertex set H (*handle*) and k sets $T_1, \dots, T_k$ (*teeth*) with:

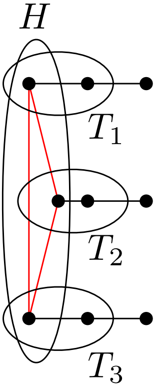
- $H \cap T_i \neq \emptyset$ for all i, i.e., every tooth has a vertex in the handle,
- $T_i \setminus H \neq \emptyset$ for all i, i.e., every tooth has a vertex outset the handle,
- $T_i \cap T_j = \emptyset$ for all $i \neq j$, i.e., the teeth are disjoint,
- $k \geq 3$, and k is odd.



We analyze $\underbrace{\sum_{e \in \delta(H)} x_e}_{P_1} + \underbrace{\sum_{i=1}^{k} \sum_{e \in \delta(T_i)} x_e}_{P_2}.$

- Let τ be any tour.
- For each tooth T_i, we have two cases: (1) τ has an edge going from $T_i \cap H$ to $T_i \setminus H$, or (2) it has no such edge.
- In case (1), T_i contributes one edge to P_1 and two edges to P_2, i.e., at least three edges in total.
- In case (2), T_i has to be entered and left at least twice, once for $T_i \cap H$ and once for $T_i \setminus H$. Each visit contributes two edges to P_2, i.e., at least four edges in total.

COMB INEQUALITIES



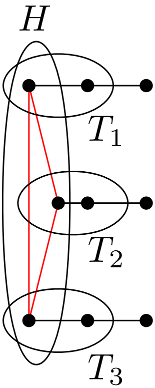
Consider a vertex set H (*handle*) and k sets $T_1, \dots, T_k$ (*teeth*) with:

- $H \cap T_i \neq \emptyset$ for all i, i.e., every tooth has a vertex in the handle,
- $T_i \setminus H \neq \emptyset$ for all i, i.e., every tooth has a vertex outset the handle,
- $T_i \cap T_j = \emptyset$ for all $i \neq j$, i.e., the teeth are disjoint,
- $k \geq 3$, and k is odd.

We analyze $\underbrace{\sum_{e \in \delta(H)} x_e}_{P_1} + \underbrace{\sum_{i=1}^{k} \sum_{e \in \delta(T_i)} x_e}_{P_2}.$

- Let τ be any tour.
- For each tooth T_i, we have two cases: (1) τ has an edge going from $T_i \cap H$ to $T_i \setminus H$, or (2) it has no such edge.
- In case (1), T_i contributes one edge to P_1 and two edges to P_2, i.e., at least three edges in total.
- In case (2), T_i has to be entered and left at least twice, once for $T_i \cap H$ and once for $T_i \setminus H$. Each visit contributes two edges to P_2, i.e., at least four edges in total.
- We have $P_1 + P_2 \geq 3k$. What parities do the sides of this inequality have?

COMB INEQUALITIES



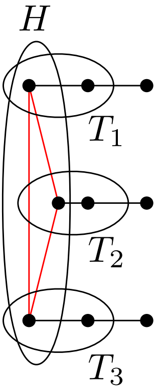
Consider a vertex set H (*handle*) and k sets $T_1, \dots, T_k$ (*teeth*) with:

- $H \cap T_i \neq \emptyset$ for all i, i.e., every tooth has a vertex in the handle,
- $T_i \setminus H \neq \emptyset$ for all i, i.e., every tooth has a vertex outset the handle,
- $T_i \cap T_j = \emptyset$ for all $i \neq j$, i.e., the teeth are disjoint,
- $k \geq 3$, and k is odd.

We analyze $\underbrace{\sum_{e \in \delta(H)} x_e}_{P_1} + \underbrace{\sum_{i=1}^{k} \sum_{e \in \delta(T_i)} x_e}_{P_2}$.

- Let τ be any tour.
- For each tooth T_i, we have two cases: (1) τ has an edge going from $T_i \cap H$ to $T_i \setminus H$, or (2) it has no such edge.
- In case (1), T_i contributes one edge to P_1 and two edges to P_2, i.e., at least three edges in total.
- In case (2), T_i has to be entered and left at least twice, once for $T_i \cap H$ and once for $T_i \setminus H$. Each visit contributes two edges to P_2, i.e., at least four edges in total.
- We have $P_1 + P_2 \geq 3k$. What parities do the sides of this inequality have?
- For any $S \subseteq V$ and any tour, the number of edges $\delta(S)$ crossing the boundary of S is even. The right-hand side is odd because k is odd.

COMB INEQUALITIES



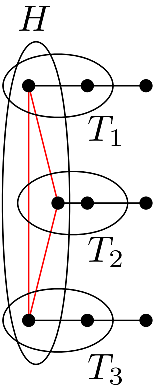
Consider a vertex set H (*handle*) and k sets $T_1, \ldots, T_k$ (*teeth*) with:

- $H \cap T_i \neq \emptyset$ for all i, i.e., every tooth has a vertex in the handle,
- $T_i \setminus H \neq \emptyset$ for all i, i.e., every tooth has a vertex outset the handle,
- $T_i \cap T_j = \emptyset$ for all $i \neq j$, i.e., the teeth are disjoint,
- $k \geq 3$, and k is odd.

We analyze $\underbrace{\sum_{e \in \delta(H)} x_e}_{P_1} + \underbrace{\sum_{i=1}^{k} \sum_{e \in \delta(T_i)} x_e}_{P_2}.$

- Let τ be any tour.
- For each tooth T_i, we have two cases: (1) τ has an edge going from $T_i \cap H$ to $T_i \setminus H$, or (2) it has no such edge.
- In case (1), T_i contributes one edge to P_1 and two edges to P_2, i.e., at least three edges in total.
- In case (2), T_i has to be entered and left at least twice, once for $T_i \cap H$ and once for $T_i \setminus H$. Each visit contributes two edges to P_2, i.e., at least four edges in total.
- We have $P_1 + P_2 \geq 3k$. What parities do the sides of this inequality have?
- For any $S \subseteq V$ and any tour, the number of edges $\delta(S)$ crossing the boundary of S is even. The right-hand side is odd because k is odd.
- We thus obtain the *comb inequality* $P_1 + P_2 \geq 3k + 1$.

COMB INEQUALITIES



Consider a vertex set H (*handle*) and k sets $T_1, \ldots, T_k$ (*teeth*) with:

- $H \cap T_i \neq \emptyset$ for all i, i.e., every tooth has a vertex in the handle,
- $T_i \setminus H \neq \emptyset$ for all i, i.e., every tooth has a vertex outset the handle,
- $T_i \cap T_j = \emptyset$ for all $i \neq j$, i.e., the teeth are disjoint,
- $k \geq 3$, and k is odd.

We analyze $\underbrace{\sum_{e \in \delta(H)} x_e}_{P_1} + \underbrace{\sum_{i=1}^{k} \sum_{e \in \delta(T_i)} x_e}_{P_2}.$

- Let τ be any tour.
- For each tooth T_i, we have two cases: (1) τ has an edge going from $T_i \cap H$ to $T_i \setminus H$, or (2) it has no such edge.
- In case (1), T_i contributes one edge to P_1 and two edges to P_2, i.e., at least three edges in total.
- In case (2), T_i has to be entered and left at least twice, once for $T_i \cap H$ and once for $T_i \setminus H$. Each visit contributes two edges to P_2, i.e., at least four edges in total.
- We have $P_1 + P_2 \geq 3k$. What parities do the sides of this inequality have?
- For any $S \subseteq V$ and any tour, the number of edges $\delta(S)$ crossing the boundary of S is even. The right-hand side is odd because k is odd.
- We thus obtain the *comb inequality* $P_1 + P_2 \geq 3k + 1$.

How can we separate cutting planes of this type?

COMB INEQUALITIES — SEPARATION

For a given (fractional) solution to the linear relaxation of our IP, the separation problem for combs asks for finding a violated comb inequality, i.e., some H and T_i for which $P_1 + P_2 < 3k + 1$, or to find that no such sets exist.

COMB INEQUALITIES — SEPARATION

For a given (fractional) solution to the linear relaxation of our IP, the separation problem for combs asks for finding a violated comb inequality, i.e., some H and T_i for which $P_1 + P_2 < 3k + 1$, or to find that no such sets exist.

It is not known whether this problem is in P or NP-complete. There are, however, a number of good heuristics that often successfully identify violated combs.

COMB INEQUALITIES — SEPARATION

For a given (fractional) solution to the linear relaxation of our IP, the separation problem for combs asks for finding a violated comb inequality, i.e., some H and T_i for which $P_1 + P_2 < 3k + 1$, or to find that no such sets exist.

It is not known whether this problem is in P or NP-complete. There are, however, a number of good heuristics that often successfully identify violated combs.

In the following, we will present the so-called *odd-component heuristic*. It is based on the idea of identifying potential handles H by looking at connected components of the solution graph without edges with (too close to) integral weight.

Comb Inequalities — Separation

For a given (fractional) solution to the linear relaxation of our IP, the separation problem for combs asks for finding a violated comb inequality, i.e., some H and T_i for which $P_1 + P_2 < 3k + 1$, or to find that no such sets exist.

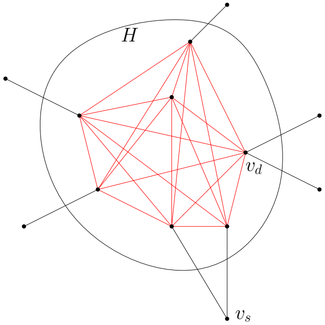
It is not known whether this problem is in P or NP-complete. There are, however, a number of good heuristics that often successfully identify violated combs.

In the following, we will present the so-called *odd-component heuristic*. It is based on the idea of identifying potential handles H by looking at connected components of the solution graph without edges with (too close to) integral weight.

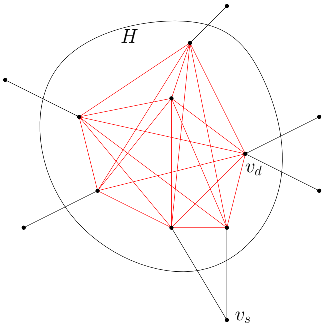
Example in interactive tool: https://www.math.uwaterloo.ca/tsp/app/diy.html

Comb Inequalities — Odd Component Heuristic

The odd component heuristic:



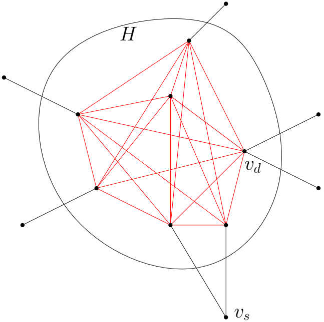
Comb Inequalities — Odd Component Heuristic



The odd component heuristic:

- Computes connected components of the fractional graph, i.e., the weighted solution graph with only edges that are more than some small $\varepsilon > 0$ away from integrality.

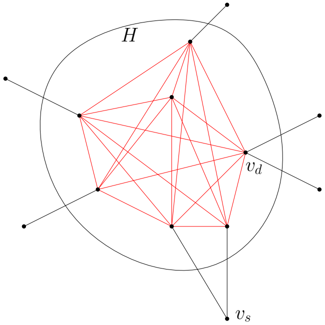
Comb Inequalities — Odd Component Heuristic



The odd component heuristic:

- Computes connected components of the fractional graph, i.e., the weighted solution graph with only edges that are more than some small $\varepsilon > 0$ away from integrality.
- For each component, it counts how many edges with weight close to 1 cross the boundary of the component.

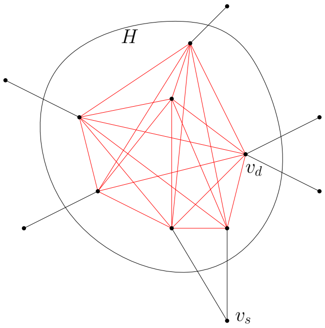
Comb Inequalities — Odd Component Heuristic



The odd component heuristic:

- Computes connected components of the fractional graph, i.e., the weighted solution graph with only edges that are more than some small $\varepsilon > 0$ away from integrality.
- For each component, it counts how many edges with weight close to 1 cross the boundary of the component.
- For any component where this count is odd, it constructs a comb as follows.

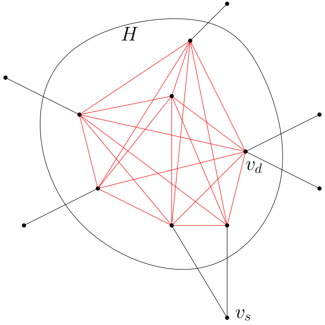
Comb Inequalities — Odd Component Heuristic



The odd component heuristic:

- Computes connected components of the fractional graph, i.e., the weighted solution graph with only edges that are more than some small $\varepsilon > 0$ away from integrality.
- For each component, it counts how many edges with weight close to 1 cross the boundary of the component.
- For any component where this count is odd, it constructs a comb as follows.
  - The handle H is initialized to the vertices in the fractional component.

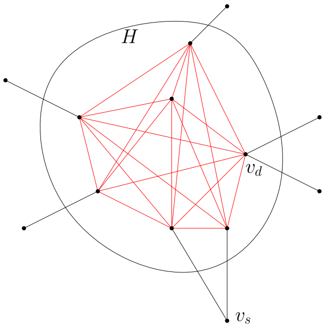
Comb Inequalities — Odd Component Heuristic



The odd component heuristic:

- Computes connected components of the fractional graph, i.e., the weighted solution graph with only edges that are more than some small $\varepsilon > 0$ away from integrality.
- For each component, it counts how many edges with weight close to 1 cross the boundary of the component.
- For any component where this count is odd, it constructs a comb as follows.
  - The handle H is initialized to the vertices in the fractional component.
  - There may be vertices v_s outside the component that are connected to two vertices on the inside. Such vertices are added to H.

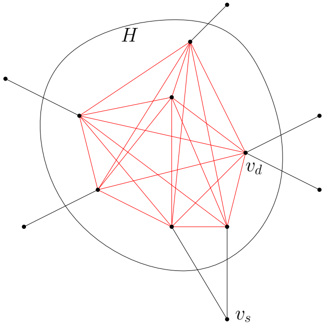
Comb Inequalities — Odd Component Heuristic



The odd component heuristic:

- Computes connected components of the fractional graph, i.e., the weighted solution graph with only edges that are more than some small $\varepsilon > 0$ away from integrality.
- For each component, it counts how many edges with weight close to 1 cross the boundary of the component.
- For any component where this count is odd, it constructs a comb as follows.
  - The handle H is initialized to the vertices in the fractional component.
  - There may be vertices v_s outside the component that are connected to two vertices on the inside. Such vertices are added to H.
  - There may be vertices v_d inside the component that are connected to two vertices on the outside. Such vertices are removed from H.

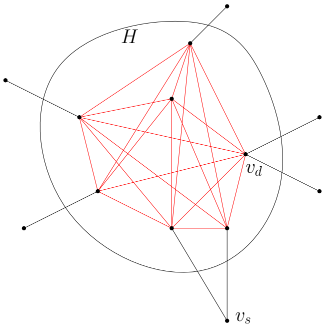
COMB INEQUALITIES — ODD COMPONENT HEURISTIC



The odd component heuristic:

- Computes connected components of the fractional graph, i.e., the weighted solution graph with only edges that are more than some small $\varepsilon > 0$ away from integrality.
- For each component, it counts how many edges with weight close to 1 cross the boundary of the component.
- For any component where this count is odd, it constructs a comb as follows.
  - The handle H is initialized to the vertices in the fractional component.
  - There may be vertices v_s outside the component that are connected to two vertices on the inside. Such vertices are added to H.
  - There may be vertices v_d inside the component that are connected to two vertices on the outside. Such vertices are removed from H.
  - In any case, an odd number of close-to-1 edges crossing H remains.

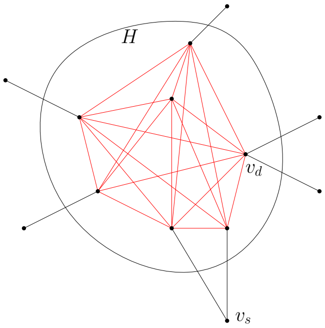
COMB INEQUALITIES — ODD COMPONENT HEURISTIC



The odd component heuristic:

- Computes connected components of the fractional graph, i.e., the weighted solution graph with only edges that are more than some small $\varepsilon > 0$ away from integrality.
- For each component, it counts how many edges with weight close to 1 cross the boundary of the component.
- For any component where this count is odd, it constructs a comb as follows.
  - The handle H is initialized to the vertices in the fractional component.
  - There may be vertices v_s outside the component that are connected to two vertices on the inside. Such vertices are added to H.
  - There may be vertices v_d inside the component that are connected to two vertices on the outside. Such vertices are removed from H.
  - In any case, an odd number of close-to-1 edges crossing H remains.
  - If that number is 1, we likely found a violated subtour constraint.

COMB INEQUALITIES — ODD COMPONENT HEURISTIC



The odd component heuristic:

- Computes connected components of the fractional graph, i.e., the weighted solution graph with only edges that are more than some small $\varepsilon > 0$ away from integrality.
- For each component, it counts how many edges with weight close to 1 cross the boundary of the component.
- For any component where this count is odd, it constructs a comb as follows.
  - The handle H is initialized to the vertices in the fractional component.
  - There may be vertices v_s outside the component that are connected to two vertices on the inside. Such vertices are added to H.
  - There may be vertices v_d inside the component that are connected to two vertices on the outside. Such vertices are removed from H.
  - In any case, an odd number of close-to-1 edges crossing H remains.
  - If that number is 1, we likely found a violated subtour constraint.
  - If that number is 3 or higher, we make the crossing edges into teeth T_i, and have likely found a violated comb inequality.

More Cutting Planes

There are more cutting planes that are used by powerful tools like Concorde, for which we do not have the time. A common theme among them is that they always can be translated into the following form:

$$\sum_{S \in \mathcal{F}} \sum_{e \in \delta(S)} x_e \geq \mu,$$

for some family of vertex sets $\mathcal{F}$ and some suitable integral constant μ. This allows efficient storage (only store vertex sets and μ) and also changes to the set of edges.

What? Why? How?

Cutting Planes

Excluding Edges

Numerical Issues

Dynamic Edge Set

Perhaps the biggest obstacle to increasing the number of cities is the large set of potential edges.

Dynamic Edge Set

Perhaps the biggest obstacle to increasing the number of cities is the large set of potential edges.

To deal with this, Concorde handles a dynamic set of possible edges (and a dynamic set of cutting planes). Only edges from this core set are added as variables to the LP relaxations solved during Branch & Bound.

DYNAMIC EDGE SET

Perhaps the biggest obstacle to increasing the number of cities is the large set of potential edges.

To deal with this, Concorde handles a dynamic set of possible edges (and a dynamic set of cutting planes). Only edges from this core set are added as variables to the LP relaxations solved during Branch & Bound. Typically, the core LP has only about $1.5n$–$3n$ variables.

DYNAMIC EDGE SET

Perhaps the biggest obstacle to increasing the number of cities is the large set of potential edges.

To deal with this, Concorde handles a dynamic set of possible edges (and a dynamic set of cutting planes). Only edges from this core set are added as variables to the LP relaxations solved during Branch & Bound. Typically, the core LP has only about $1.5n$–$3n$ variables.

- We can use our formulation to solve the TSP on a smaller edge set. As we discard some potential edges, the solution might be suboptimal if any edges from the optimal solution are removed.

DYNAMIC EDGE SET

Perhaps the biggest obstacle to increasing the number of cities is the large set of potential edges.

To deal with this, Concorde handles a dynamic set of possible edges (and a dynamic set of cutting planes). Only edges from this core set are added as variables to the LP relaxations solved during Branch & Bound. Typically, the core LP has only about $1.5n$–$3n$ variables.

- We can use our formulation to solve the TSP on a smaller edge set. As we discard some potential edges, the solution might be suboptimal if any edges from the optimal solution are removed.
- We have already seen how to store our cutting planes and subtour constraints without explicitly listing edges.

DYNAMIC EDGE SET

Perhaps the biggest obstacle to increasing the number of cities is the large set of potential edges.

To deal with this, Concorde handles a dynamic set of possible edges (and a dynamic set of cutting planes). Only edges from this core set are added as variables to the LP relaxations solved during Branch & Bound. Typically, the core LP has only about $1.5n$–$3n$ variables.

- We can use our formulation to solve the TSP on a smaller edge set. As we discard some potential edges, the solution might be suboptimal if any edges from the optimal solution are removed.
- We have already seen how to store our cutting planes and subtour constraints without explicitly listing edges.
- To make sure that we do not discard the optimal solution in our overall search, we consider the nodes of our Branch & Bound search tree where we do not continue branching:

DYNAMIC EDGE SET

Perhaps the biggest obstacle to increasing the number of cities is the large set of potential edges.

To deal with this, Concorde handles a dynamic set of possible edges (and a dynamic set of cutting planes). Only edges from this core set are added as variables to the LP relaxations solved during Branch & Bound. Typically, the core LP has only about $1.5n$–$3n$ variables.

- We can use our formulation to solve the TSP on a smaller edge set. As we discard some potential edges, the solution might be suboptimal if any edges from the optimal solution are removed.
- We have already seen how to store our cutting planes and subtour constraints without explicitly listing edges.
- To make sure that we do not discard the optimal solution in our overall search, we consider the nodes of our Branch & Bound search tree where we do not continue branching:
  - LP infeasibility: We must make sure that there are no variables we could add to restore feasibility (normally, this does not happen).

Dynamic Edge Set

Perhaps the biggest obstacle to increasing the number of cities is the large set of potential edges.

To deal with this, Concorde handles a dynamic set of possible edges (and a dynamic set of cutting planes). Only edges from this core set are added as variables to the LP relaxations solved during Branch & Bound. Typically, the core LP has only about $1.5n$–$3n$ variables.

- We can use our formulation to solve the TSP on a smaller edge set. As we discard some potential edges, the solution might be suboptimal if any edges from the optimal solution are removed.
- We have already seen how to store our cutting planes and subtour constraints without explicitly listing edges.
- To make sure that we do not discard the optimal solution in our overall search, we consider the nodes of our Branch & Bound search tree where we do not continue branching:
  - LP infeasibility: We must make sure that there are no variables we could add to restore feasibility (normally, this does not happen).
  - Integrality or Pruning: we need to make sure that there is no better LP solution when we add variables.

DYNAMIC EDGE SET

Perhaps the biggest obstacle to increasing the number of cities is the large set of potential edges.

To deal with this, Concorde handles a dynamic set of possible edges (and a dynamic set of cutting planes). Only edges from this core set are added as variables to the LP relaxations solved during Branch & Bound. Typically, the core LP has only about $1.5n$–$3n$ variables.

- We can use our formulation to solve the TSP on a smaller edge set. As we discard some potential edges, the solution might be suboptimal if any edges from the optimal solution are removed.
- We have already seen how to store our cutting planes and subtour constraints without explicitly listing edges.
- To make sure that we do not discard the optimal solution in our overall search, we consider the nodes of our Branch & Bound search tree where we do not continue branching:
  - LP infeasibility: We must make sure that there are no variables we could add to restore feasibility (normally, this does not happen).
  - Integrality or Pruning: we need to make sure that there is no better LP solution when we add variables.
- How can we make sure (in a single LP) that we did not miss an important edge variable?

DYNAMIC EDGE SET

Perhaps the biggest obstacle to increasing the number of cities is the large set of potential edges.

To deal with this, Concorde handles a dynamic set of possible edges (and a dynamic set of cutting planes). Only edges from this core set are added as variables to the LP relaxations solved during Branch & Bound. Typically, the core LP has only about $1.5n$–$3n$ variables.

- We can use our formulation to solve the TSP on a smaller edge set. As we discard some potential edges, the solution might be suboptimal if any edges from the optimal solution are removed.
- We have already seen how to store our cutting planes and subtour constraints without explicitly listing edges.
- To make sure that we do not discard the optimal solution in our overall search, we consider the nodes of our Branch & Bound search tree where we do not continue branching:
    - LP infeasibility: We must make sure that there are no variables we could add to restore feasibility (normally, this does not happen).
    - Integrality or Pruning: we need to make sure that there is no better LP solution when we add variables.
- How can we make sure (in a single LP) that we did not miss an important edge variable?
- We need to find edges that are not dually feasible!

Dynamic Edge Set

Perhaps the biggest obstacle to increasing the number of cities is the large set of potential edges.

To deal with this, Concorde handles a dynamic set of possible edges (and a dynamic set of cutting planes). Only edges from this core set are added as variables to the LP relaxations solved during Branch & Bound. Typically, the core LP has only about $1.5n$–$3n$ variables.

- We can use our formulation to solve the TSP on a smaller edge set. As we discard some potential edges, the solution might be suboptimal if any edges from the optimal solution are removed.
- We have already seen how to store our cutting planes and subtour constraints without explicitly listing edges.
- To make sure that we do not discard the optimal solution in our overall search, we consider the nodes of our Branch & Bound search tree where we do not continue branching:
  - LP infeasibility: We must make sure that there are no variables we could add to restore feasibility (normally, this does not happen).
  - Integrality or Pruning: we need to make sure that there is no better LP solution when we add variables.
- How can we make sure (in a single LP) that we did not miss an important edge variable?
- We need to find edges that are not dually feasible!
- Is there a way to exclude lots of edges at once and only consider fewer remaining ones?

DUAL

With cutting planes and subtour constraints $\mathcal{H}$, we have the following primal:

$$\min \sum_{e} c_e x_e \text{ s.t.}$$

$$\forall v \in V : \sum_{e \in \delta(\{v\})} x_e = 2,$$

$$\forall e \in E : x_e \leq 1,$$

$$\forall (\mathcal{F}, \mu_{\mathcal{F}}) \in \mathcal{H} : \sum_{S \in \mathcal{F}} \sum_{e \in \delta(S)} x_e \geq \mu.$$

The dual of our problem is thus

DUAL

With cutting planes and subtour constraints $\mathcal{H}$, we have the following primal:

$$\min \sum_{e} c_e x_e \text{ s.t.}$$

$$\forall v \in V : \sum_{e \in \delta(\{v\})} x_e = 2,$$

$$\forall e \in E : x_e \leq 1,$$

$$\forall (\mathcal{F}, \mu_{\mathcal{F}}) \in \mathcal{H} : \sum_{S \in \mathcal{F}} \sum_{e \in \delta(S)} x_e \geq \mu.$$

The dual of our problem is thus

$$\max 2 \sum_{v \in V} y_v - \sum_{e \in E} y_e + \sum_{(\mathcal{F}, \mu_{\mathcal{F}}) \in H} \mu_{\mathcal{F}} \cdot y_{\mathcal{F}} \text{ s.t.}$$

DUAL

With cutting planes and subtour constraints $\mathcal{H}$, we have the following primal:

$$\min \sum_e c_e x_e \text{ s.t.}$$

$$\forall v \in V : \sum_{e \in \delta(\{v\})} x_e = 2,$$

$$\forall e \in E : x_e \leq 1,$$

$$\forall (\mathcal{F}, \mu_{\mathcal{F}}) \in \mathcal{H} : \sum_{S \in \mathcal{F}} \sum_{e \in \delta(S)} x_e \geq \mu.$$

The dual of our problem is thus

$$\max 2 \sum_{v \in V} y_v - \sum_{e \in E} y_e + \sum_{(\mathcal{F}, \mu_{\mathcal{F}}) \in H} \mu_{\mathcal{F}} \cdot y_{\mathcal{F}} \text{ s.t.}$$

$$\forall e = vw \in E : y_v + y_w - y_e + \sum_{(\mathcal{F}, \mu_{\mathcal{F}}) \in \mathcal{H}} \pi(e, \mathcal{F}) y_{\mathcal{F}} \leq c_e$$

where $\pi(e, \mathcal{F})$ denotes for how many sets in $\mathcal{F}$ edge e crosses the boundary.

DUAL

With cutting planes and subtour constraints $\mathcal{H}$, we have the following primal:

$$\min \sum_{e} c_e x_e \text{ s.t.}$$

$$\forall v \in V : \sum_{e \in \delta(\{v\})} x_e = 2,$$

$$\forall e \in E : x_e \leq 1,$$

$$\forall (\mathcal{F}, \mu_{\mathcal{F}}) \in \mathcal{H} : \sum_{S \in \mathcal{F}} \sum_{e \in \delta(S)} x_e \geq \mu.$$

The dual of our problem is thus

$$\max 2 \sum_{v \in V} y_v - \sum_{e \in E} y_e + \sum_{(\mathcal{F}, \mu_{\mathcal{F}}) \in H} \mu_{\mathcal{F}} \cdot y_{\mathcal{F}} \text{ s.t.}$$

$$\forall e = vw \in E : y_v + y_w - y_e + \sum_{(\mathcal{F}, \mu_{\mathcal{F}}) \in \mathcal{H}} \pi(e, \mathcal{F}) y_{\mathcal{F}} \leq c_e$$

$$y_v \text{ free}, y_e, y_{\mathcal{F}} \geq 0,$$

where $\pi(e, \mathcal{F})$ denotes for how many sets in $\mathcal{F}$ edge e crosses the boundary.

Pricing

We are looking for edges e for which $\alpha_e = c_e - y_v - y_w + y_e - \sum_{(\mathcal{F},\mu_{\mathcal{F}})\in\mathcal{H}} \pi(e,\mathcal{F}) y_{\mathcal{F}} < 0.$

Pricing

We are looking for edges e for which $\alpha_e = c_e - y_v - y_w + y_e - \sum_{(\mathcal{F},\mu_{\mathcal{F}})\in\mathcal{H}} \pi(e,\mathcal{F}) y_{\mathcal{F}} < 0.$

- We want to replace this term for α_e by a good estimate $\bar{\alpha}_e \leq \alpha_e$.

PRICING

We are looking for edges e for which $\alpha_e = c_e - y_v - y_w + y_e - \sum_{(\mathcal{F},\mu_{\mathcal{F}})\in\mathcal{H}} \pi(e,\mathcal{F})y_{\mathcal{F}} < 0.$

- We want to replace this term for α_e by a good estimate $\bar{\alpha}_e \leq \alpha_e$.
- $\bar{\alpha}_e$ should be simpler and faster to compute.

Pricing

We are looking for edges e for which $\alpha_e = c_e - y_v - y_w + y_e - \sum_{(\mathcal{F},\mu_\mathcal{F}) \in \mathcal{H}} \pi(e,\mathcal{F}) y_\mathcal{F} < 0.$

- We want to replace this term for α_e by a good estimate $\bar{\alpha}_e \leq \alpha_e$.
- $\bar{\alpha}_e$ should be simpler and faster to compute.
- We then only have to compute α_e explicitly for edges with $\bar{\alpha}_e < 0$.

PRICING

We are looking for edges e for which $\alpha_e = c_e - y_v - y_w + y_e - \sum_{(\mathcal{F},\mu_\mathcal{F})\in\mathcal{H}} \pi(e,\mathcal{F}) y_\mathcal{F} < 0.$

- We want to replace this term for α_e by a good estimate $\bar{\alpha}_e \leq \alpha_e$.
- $\bar{\alpha}_e$ should be simpler and faster to compute.
- We then only have to compute α_e explicitly for edges with $\bar{\alpha}_e < 0$.
- For $\bar{\alpha}_e$, we simply ignore y_e (complementarity dictates $y_e = 0$ anyways).

PRICING

We are looking for edges e for which $\alpha_e = c_e - y_v - y_w + y_e - \sum_{(\mathcal{F},\mu_\mathcal{F})\in\mathcal{H}} \pi(e,\mathcal{F}) y_\mathcal{F} < 0.$

- We want to replace this term for α_e by a good estimate $\bar{\alpha}_e \leq \alpha_e$.
- $\bar{\alpha}_e$ should be simpler and faster to compute.
- We then only have to compute α_e explicitly for edges with $\bar{\alpha}_e < 0$.
- For $\bar{\alpha}_e$, we simply ignore y_e (complementarity dictates $y_e = 0$ anyways).
- The most complex term is related to the cutting planes.
  Essentially, we have to figure out for what sets S from our cuts e crosses the boundary.

Pricing

We are looking for edges e for which $\alpha_e = c_e - y_v - y_w + y_e - \sum_{(\mathcal{F},\mu_{\mathcal{F}})\in\mathcal{H}} \pi(e,\mathcal{F})y_{\mathcal{F}} < 0.$

- We want to replace this term for α_e by a good estimate $\bar{\alpha}_e \leq \alpha_e$.
- $\bar{\alpha}_e$ should be simpler and faster to compute.
- We then only have to compute α_e explicitly for edges with $\bar{\alpha}_e < 0$.
- For $\bar{\alpha}_e$, we simply ignore y_e (complementarity dictates $y_e = 0$ anyways).
- The most complex term is related to the cutting planes.
  Essentially, we have to figure out for what sets S from our cuts e crosses the boundary.
- However, we note that in order for $e = vw$ to cross the boundary of S, v or w must be in S.

PRICING

We are looking for edges e for which $\alpha_e = c_e - y_v - y_w + y_e - \sum_{(\mathcal{F},\mu_\mathcal{F})\in\mathcal{H}} \pi(e,\mathcal{F})y_\mathcal{F} < 0.$

- We want to replace this term for α_e by a good estimate $\bar{\alpha}_e \leq \alpha_e$.
- $\bar{\alpha}_e$ should be simpler and faster to compute.
- We then only have to compute α_e explicitly for edges with $\bar{\alpha}_e < 0$.
- For $\bar{\alpha}_e$, we simply ignore y_e (complementarity dictates $y_e = 0$ anyways).
- The most complex term is related to the cutting planes.
  Essentially, we have to figure out for what sets S from our cuts e crosses the boundary.
- However, we note that in order for $e = vw$ to cross the boundary of S, v or w must be in S.
- So let us overestimate the cut term by including all sets containing v or w.

PRICING

We are looking for edges e for which $\alpha_e = c_e - y_v - y_w + y_e - \sum_{(\mathcal{F},\mu_\mathcal{F})\in\mathcal{H}} \pi(e,\mathcal{F})y_\mathcal{F} < 0$.

- We want to replace this term for α_e by a good estimate $\bar{\alpha}_e \leq \alpha_e$.
- $\bar{\alpha}_e$ should be simpler and faster to compute.
- We then only have to compute α_e explicitly for edges with $\bar{\alpha}_e < 0$.
- For $\bar{\alpha}_e$, we simply ignore y_e (complementarity dictates $y_e = 0$ anyways).
- The most complex term is related to the cutting planes.
  Essentially, we have to figure out for what sets S from our cuts e crosses the boundary.
- However, we note that in order for $e = vw$ to cross the boundary of S, v or w must be in S.
- So let us overestimate the cut term by including all sets containing v or w.
- We can then move these terms into y_v and y_w:

$$\bar{y}_v = y_v + \sum_{(\mu_\mathcal{F},\mathcal{F})} \pi(v,\mathcal{F})y_\mathcal{F},$$

  where $\pi(v,\mathcal{F})$ describes how many sets in $\mathcal{F}$ contain v.

PRICING

We are looking for edges e for which $\alpha_e = c_e - y_v - y_w + y_e - \sum_{(\mathcal{F},\mu_\mathcal{F}) \in \mathcal{H}} \pi(e, \mathcal{F}) y_\mathcal{F} < 0$.

- We want to replace this term for α_e by a good estimate $\bar{\alpha}_e \leq \alpha_e$.
- $\bar{\alpha}_e$ should be simpler and faster to compute.
- We then only have to compute α_e explicitly for edges with $\bar{\alpha}_e < 0$.
- For $\bar{\alpha}_e$, we simply ignore y_e (complementarity dictates $y_e = 0$ anyways).
- The most complex term is related to the cutting planes.
  Essentially, we have to figure out for what sets S from our cuts e crosses the boundary.
- However, we note that in order for $e = vw$ to cross the boundary of S, v or w must be in S.
- So let us overestimate the cut term by including all sets containing v or w.
- We can then move these terms into y_v and y_w:

  $$\bar{y}_v = y_v + \sum_{(\mu_\mathcal{F}, \mathcal{F})} \pi(v, \mathcal{F}) y_\mathcal{F},$$

  where $\pi(v, \mathcal{F})$ describes how many sets in $\mathcal{F}$ contain v.
- We then set $\bar{\alpha}_e = c_e - \bar{y}_v - \bar{y}_w$.

PRICING

We are looking for edges e for which $\alpha_e = c_e - y_v - y_w + y_e - \sum_{(\mathcal{F},\mu_{\mathcal{F}})\in\mathcal{H}} \pi(e,\mathcal{F})y_{\mathcal{F}} < 0$.

- We want to replace this term for α_e by a good estimate $\bar{\alpha}_e \leq \alpha_e$.
- $\bar{\alpha}_e$ should be simpler and faster to compute.
- We then only have to compute α_e explicitly for edges with $\bar{\alpha}_e < 0$.
- For $\bar{\alpha}_e$, we simply ignore y_e (complementarity dictates $y_e = 0$ anyways).
- The most complex term is related to the cutting planes.
  Essentially, we have to figure out for what sets S from our cuts e crosses the boundary.
- However, we note that in order for $e = vw$ to cross the boundary of S, v or w must be in S.
- So let us overestimate the cut term by including all sets containing v or w.
- We can then move these terms into y_v and y_w:

  $$\bar{y}_v = y_v + \sum_{(\mu_{\mathcal{F}},\mathcal{F})} \pi(v,\mathcal{F})y_{\mathcal{F}},$$

  where $\pi(v,\mathcal{F})$ describes how many sets in $\mathcal{F}$ contain v.
- We then set $\bar{\alpha}_e = c_e - \bar{y}_v - \bar{y}_w$.
- With careful management of variables, for instances up to tens of thousands of cities, this suffices to bring down the time needed for scanning for dual infeasible edges into the practical range.

PRICING

We are looking for edges e for which $\alpha_e = c_e - y_v - y_w + y_e - \sum_{(\mathcal{F},\mu_{\mathcal{F}})\in\mathcal{H}} \pi(e,\mathcal{F}) y_{\mathcal{F}} < 0$.

- We want to replace this term for α_e by a good estimate $\bar{\alpha}_e \leq \alpha_e$.
- $\bar{\alpha}_e$ should be simpler and faster to compute.
- We then only have to compute α_e explicitly for edges with $\bar{\alpha}_e < 0$.
- For $\bar{\alpha}_e$, we simply ignore y_e (complementarity dictates $y_e = 0$ anyways).
- The most complex term is related to the cutting planes.
  Essentially, we have to figure out for what sets S from our cuts e crosses the boundary.
- However, we note that in order for $e = vw$ to cross the boundary of S, v or w must be in S.
- So let us overestimate the cut term by including all sets containing v or w.
- We can then move these terms into y_v and y_w:

  $$\bar{y}_v = y_v + \sum_{(\mu_{\mathcal{F}},\mathcal{F})} \pi(v,\mathcal{F}) y_{\mathcal{F}},$$

  where $\pi(v,\mathcal{F})$ describes how many sets in $\mathcal{F}$ contain v.
- We then set $\bar{\alpha}_e = c_e - \bar{y}_v - \bar{y}_w$.
- With careful management of variables, for instances up to tens of thousands of cities, this suffices to bring down the time needed for scanning for dual infeasible edges into the practical range.
- We can do even better, e.g., for geometric instances.

PRICING

We are looking for edges e for which $\alpha_e = c_e - y_v - y_w + y_e - \sum_{(\mathcal{F},\mu_{\mathcal{F}})\in\mathcal{H}} \pi(e,\mathcal{F})y_{\mathcal{F}} < 0$.

- We want to replace this term for α_e by a good estimate $\bar{\alpha}_e \leq \alpha_e$.
- $\bar{\alpha}_e$ should be simpler and faster to compute.
- We then only have to compute α_e explicitly for edges with $\bar{\alpha}_e < 0$.
- For $\bar{\alpha}_e$, we simply ignore y_e (complementarity dictates $y_e = 0$ anyways).
- The most complex term is related to the cutting planes.
  Essentially, we have to figure out for what sets S from our cuts e crosses the boundary.
- However, we note that in order for $e = vw$ to cross the boundary of S, v or w must be in S.
- So let us overestimate the cut term by including all sets containing v or w.
- We can then move these terms into y_v and y_w:

  $$\bar{y}_v = y_v + \sum_{(\mu_{\mathcal{F}},\mathcal{F})} \pi(v,\mathcal{F})y_{\mathcal{F}},$$

  where $\pi(v,\mathcal{F})$ describes how many sets in $\mathcal{F}$ contain v.
- We then set $\bar{\alpha}_e = c_e - \bar{y}_v - \bar{y}_w$.
- With careful management of variables, for instances up to tens of thousands of cities, this suffices to bring down the time needed for scanning for dual infeasible edges into the practical range.
- We can do even better, e.g., for geometric instances.

The process of looking for dual infeasible variables is also called *pricing*; this is because $c - A^T y$ is also called the vector of *reduced cost*.

PRICING

We are looking for edges e for which $\alpha_e = c_e - y_v - y_w + y_e - \sum_{(\mathcal{F},\mu_\mathcal{F})\in\mathcal{H}} \pi(e,\mathcal{F}) y_\mathcal{F} < 0$.

- We want to replace this term for α_e by a good estimate $\bar{\alpha}_e \leq \alpha_e$.
- $\bar{\alpha}_e$ should be simpler and faster to compute.
- We then only have to compute α_e explicitly for edges with $\bar{\alpha}_e < 0$.
- For $\bar{\alpha}_e$, we simply ignore y_e (complementarity dictates $y_e = 0$ anyways).
- The most complex term is related to the cutting planes.
  Essentially, we have to figure out for what sets S from our cuts e crosses the boundary.
- However, we note that in order for $e = vw$ to cross the boundary of S, v or w must be in S.
- So let us overestimate the cut term by including all sets containing v or w.
- We can then move these terms into y_v and y_w:

  $$\bar{y}_v = y_v + \sum_{(\mu_\mathcal{F},\mathcal{F})} \pi(v,\mathcal{F}) y_\mathcal{F},$$

  where $\pi(v,\mathcal{F})$ describes how many sets in $\mathcal{F}$ contain v.
- We then set $\bar{\alpha}_e = c_e - \bar{y}_v - \bar{y}_w$.
- With careful management of variables, for instances up to tens of thousands of cities, this suffices to bring down the time needed for scanning for dual infeasible edges into the practical range.
- We can do even better, e.g., for geometric instances.

The process of looking for dual infeasible variables is also called *pricing*; this is because $c - A^T y$ is also called the vector of *reduced cost*.

If we can price many variables without explicitly considering each individually, like in the case of geometric instances, the term *batch pricing* is sometimes used.

WHAT? WHY? HOW?

CUTTING PLANES

EXCLUDING EDGES

NUMERICAL ISSUES

Numerical Issues

There are multiple numerical issues.

How can we overcome these issues? Simple ideas:

Numerical Issues

There are multiple numerical issues.

- The solution by the LP solver may not be primally feasible in the strict sense.

How can we overcome these issues? Simple ideas:

Numerical Issues

There are multiple numerical issues.

- The solution by the LP solver may not be primally feasible in the strict sense.
- The solution by the LP solver may not be dually feasible in the strict sense.

How can we overcome these issues? Simple ideas:

Numerical Issues

There are multiple numerical issues.

- The solution by the LP solver may not be primally feasible in the strict sense.
- The solution by the LP solver may not be dually feasible in the strict sense.
- Solutions produced by the LP solver are only rarely integral, even though they should be, e.g., because in exact arithmetic, the basis chosen by the solver produces an integral solution.

How can we overcome these issues? Simple ideas:

NUMERICAL ISSUES

There are multiple numerical issues.

- The solution by the LP solver may not be primally feasible in the strict sense.
- The solution by the LP solver may not be dually feasible in the strict sense.
- Solutions produced by the LP solver are only rarely integral, even though they should be, e.g., because in exact arithmetic, the basis chosen by the solver produces an integral solution.
- Cutting planes that should be violated are considered non-violated by the LP solver.

How can we overcome these issues? Simple ideas:

NUMERICAL ISSUES

There are multiple numerical issues.

- The solution by the LP solver may not be primally feasible in the strict sense.
- The solution by the LP solver may not be dually feasible in the strict sense.
- Solutions produced by the LP solver are only rarely integral, even though they should be, e.g., because in exact arithmetic, the basis chosen by the solver produces an integral solution.
- Cutting planes that should be violated are considered non-violated by the LP solver.

How can we overcome these issues? Simple ideas:

- Run LP with exact rational arithmetic — way too expensive!

Numerical Issues

There are multiple numerical issues.

- The solution by the LP solver may not be primally feasible in the strict sense.
- The solution by the LP solver may not be dually feasible in the strict sense.
- Solutions produced by the LP solver are only rarely integral, even though they should be, e.g., because in exact arithmetic, the basis chosen by the solver produces an integral solution.
- Cutting planes that should be violated are considered non-violated by the LP solver.

How can we overcome these issues? Simple ideas:

- Run LP with exact rational arithmetic — way too expensive!
- Run regular LP solver, then take the optimal basis and use it as starting point for an exact rational Simplex — still quite expensive!

Numerical Issues — Severity

- Integer feasibility

NUMERICAL ISSUES — SEVERITY

- Integer feasibility can simply be checked with some $\varepsilon > 0$ tolerance value, considering values below ε as 0 and values above $1 - \varepsilon$ as 1.

NUMERICAL ISSUES — SEVERITY

- Integer feasibility can simply be checked with some $\varepsilon > 0$ tolerance value, considering values below ε as 0 and values above $1 - \varepsilon$ as 1.
- Accidental primal infeasibility

Numerical Issues — Severity

- Integer feasibility can simply be checked with some $\varepsilon > 0$ tolerance value, considering values below ε as 0 and values above $1 - \varepsilon$ as 1.
- Accidental primal infeasibility is only really problematic for integer solutions. Fortunately, it is easy to check if some integer solution induces a tour.

NUMERICAL ISSUES — SEVERITY

- Integer feasibility can simply be checked with some $\varepsilon > 0$ tolerance value, considering values below ε as 0 and values above $1 - \varepsilon$ as 1.
- Accidental primal infeasibility is only really problematic for integer solutions. Fortunately, it is easy to check if some integer solution induces a tour.
- Cutting planes are used purely to speed up the search. Cutting planes that are only violated by a tiny amount are unlikely to be good; if we avoid them, the solver should not consider our cutting planes as non-violated.

NUMERICAL ISSUES — SEVERITY

- Integer feasibility can simply be checked with some $\varepsilon > 0$ tolerance value, considering values below ε as 0 and values above $1 - \varepsilon$ as 1.
- Accidental primal infeasibility is only really problematic for integer solutions. Fortunately, it is easy to check if some integer solution induces a tour.
- Cutting planes are used purely to speed up the search. Cutting planes that are only violated by a tiny amount are unlikely to be good; if we avoid them, the solver should not consider our cutting planes as non-violated.
- The most problematic issue is accidental dual infeasibility (i.e., non-optimality): it means that our bounds used for pruning are potentially invalid. What can we do?

EXACT BOUNDS FROM INEXACT SOLUTIONS

How can we turn inexact bounds from our solver into exact bounds useful for pruning?

Exact Bounds from Inexact Solutions

How can we turn inexact bounds from our solver into exact bounds useful for pruning?

- Start with the dual solution y.

Exact Bounds from Inexact Solutions

How can we turn inexact bounds from our solver into exact bounds useful for pruning?

- Start with the dual solution y.
- Turn it into rational (more expensive, better bounds) or fixed-point (less expensive) numbers.

EXACT BOUNDS FROM INEXACT SOLUTIONS

How can we turn inexact bounds from our solver into exact bounds useful for pruning?

- Start with the dual solution y.
- Turn it into rational (more expensive, better bounds) or fixed-point (less expensive) numbers.
- For each dual constraint, check feasibility (in exact rational or fixed-point arithmetic).

Exact Bounds from Inexact Solutions

How can we turn inexact bounds from our solver into exact bounds useful for pruning?

- Start with the dual solution y.
- Turn it into rational (more expensive, better bounds) or fixed-point (less expensive) numbers.
- For each dual constraint, check feasibility (in exact rational or fixed-point arithmetic).
- For each dual infeasibility found, increase y_{vw}, the dual slack variable for $x_{vw} \leq 1$, to make the dual constraint

$$y_v + y_w - y_{vw} + \sum_{(\mathcal{F},\mu_{\mathcal{F}})\in\mathcal{H}} \pi(vw,\mathcal{F})y_{\mathcal{F}} \leq c_{vw}$$

feasible.

Exact Bounds from Inexact Solutions

How can we turn inexact bounds from our solver into exact bounds useful for pruning?

- Start with the dual solution y.
- Turn it into rational (more expensive, better bounds) or fixed-point (less expensive) numbers.
- For each dual constraint, check feasibility (in exact rational or fixed-point arithmetic).
- For each dual infeasibility found, increase y_{vw}, the dual slack variable for $x_{vw} \leq 1$, to make the dual constraint

$$y_v + y_w - y_{vw} + \sum_{(\mathcal{F},\mu_{\mathcal{F}})\in\mathcal{H}} \pi(vw,\mathcal{F}) y_{\mathcal{F}} \leq c_{vw}$$

  feasible.
- Subtract all the infeasibilities from the objective according to the term $-y_{vw}$ in the dual objective.

EXACT BOUNDS FROM INEXACT SOLUTIONS

How can we turn inexact bounds from our solver into exact bounds useful for pruning?

- Start with the dual solution y.
- Turn it into rational (more expensive, better bounds) or fixed-point (less expensive) numbers.
- For each dual constraint, check feasibility (in exact rational or fixed-point arithmetic).
- For each dual infeasibility found, increase y_{vw}, the dual slack variable for $x_{vw} \leq 1$, to make the dual constraint

$$y_v + y_w - y_{vw} + \sum_{(\mathcal{F},\mu_{\mathcal{F}}) \in \mathcal{H}} \pi(vw, \mathcal{F}) y_{\mathcal{F}} \leq c_{vw}$$

  feasible.
- Subtract all the infeasibilities from the objective according to the term $-y_{vw}$ in the dual objective.
- The result is a valid (possibly slightly super-optimal) bound.

Conclusion

The work on the TSP has contributed huge advances in MIP solving in general.
The approaches used by Concorde to tackle the TSP are not all simple to implement in a commercial MIP solver.

Conclusion

The work on the TSP has contributed huge advances in MIP solving in general.
The approaches used by Concorde to tackle the TSP are not all simple to implement in a commercial MIP solver.

- Custom cutting planes and lazy constraints can be easily added using a callback.

CONCLUSION

The work on the TSP has contributed huge advances in MIP solving in general.
The approaches used by Concorde to tackle the TSP are not all simple to implement in a commercial MIP solver.

- Custom cutting planes and lazy constraints can be easily added using a callback.
- Dynamic Removal of Cutting Planes is performed, but cannot really be controlled.

CONCLUSION

The work on the TSP has contributed huge advances in MIP solving in general.
The approaches used by Concorde to tackle the TSP are not all simple to implement in a commercial MIP solver.

- Custom cutting planes and lazy constraints can be easily added using a callback.
- Dynamic Removal of Cutting Planes is performed, but cannot really be controlled.
- Auto-generated cutting planes are usually not accessible.

CONCLUSION

The work on the TSP has contributed huge advances in MIP solving in general.
The approaches used by Concorde to tackle the TSP are not all simple to implement in a commercial MIP solver.

- Custom cutting planes and lazy constraints can be easily added using a callback.
- Dynamic Removal of Cutting Planes is performed, but cannot really be controlled.
- Auto-generated cutting planes are usually not accessible.
- Dynamic handling of variables is usually impossible in a (M)IP.

CONCLUSION

The work on the TSP has contributed huge advances in MIP solving in general.
The approaches used by Concorde to tackle the TSP are not all simple to implement in a commercial MIP solver.

- Custom cutting planes and lazy constraints can be easily added using a callback.
- Dynamic Removal of Cutting Planes is performed, but cannot really be controlled.
- Auto-generated cutting planes are usually not accessible.
- Dynamic handling of variables is usually impossible in a (M)IP.
- There are frameworks such as SCIP that allow more flexibility, but usually worse performance.