CS498: Algorithmic Engineering

Lecture 7: TSP, MINLP, and Spatial-Branch and Bound.

Elfarouk Harb

University of Illinois Urbana-Champaign

Week 04 – 02/10/2026

Outline

1. Case Study: Travelling Salesman Problem (TSP).
2. Mixed Integer Non Linear Programming (MINLP)
3. Limits of Uniform Linearization and Spatial Branch and Bound
4. Summary and Outlook

1. Case Study: Travelling Salesman Problem (TSP).
2. Mixed Integer Non Linear Programming (MINLP)
3. Limits of Uniform Linearization and Spatial Branch and Bound
4. Summary and Outlook

Motivation: The Traveling Salesman Problem (TSP)

The **Traveling Salesman Problem** (TSP) is a classic example that combines binary decisions and ordering.

Motivation: The Traveling Salesman Problem (TSP)

The **Traveling Salesman Problem** (TSP) is a classic example that combines binary decisions and ordering.

Setup:

- A set of n cities with pairwise travel costs c_{ij}.

Motivation: The Traveling Salesman Problem (TSP)

The **Traveling Salesman Problem** (TSP) is a classic example that combines binary decisions and ordering.

Setup:

- A set of n cities with pairwise travel costs c_{ij}.
- A salesman must start at one city (say city 1), visit every other city exactly once, and return to the start.

Motivation: The Traveling Salesman Problem (TSP)

The **Traveling Salesman Problem** (TSP) is a classic example that combines binary decisions and ordering.

Setup:

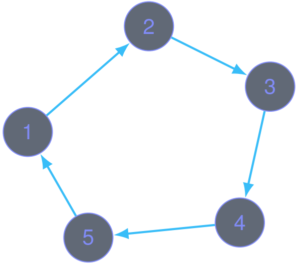
- A set of n cities with pairwise travel costs c_{ij}.
- A salesman must start at one city (say city 1), visit every other city exactly once, and return to the start.
- Goal: minimize total travel cost.

Motivation: The Traveling Salesman Problem (TSP)

The **Traveling Salesman Problem** (TSP) is a classic example that combines binary decisions and ordering.

Setup:

- A set of n cities with pairwise travel costs c_{ij}.
- A salesman must start at one city (say city 1), visit every other city exactly once, and return to the start.
- Goal: minimize total travel cost.



TSP with Arc Variables

Traveling Salesman Problem (TSP):

- n cities, travel costs c_{ij}.

TSP with Arc Variables

Traveling Salesman Problem (TSP):

- n cities, travel costs c_{ij}.
- Binary arc variables $x_{ij} \in \{0, 1\}$:

$$x_{ij} = 1 \;\Leftrightarrow\; \text{tour goes directly from } i \text{ to } j.$$

TSP with Arc Variables

Traveling Salesman Problem (TSP):

- n cities, travel costs c_{ij}.
- Binary arc variables $x_{ij} \in \{0, 1\}$:

$$x_{ij} = 1 \;\Leftrightarrow\; \text{tour goes directly from } i \text{ to } j.$$

Degree constraints:

TSP with Arc Variables

Traveling Salesman Problem (TSP):

- n cities, travel costs c_{ij}.
- Binary arc variables $x_{ij} \in \{0, 1\}$:

$$x_{ij} = 1 \;\Leftrightarrow\; \text{tour goes directly from } i \text{ to } j.$$

Degree constraints:

$$\sum_{j \neq i} x_{ij} = 1, \qquad \sum_{j \neq i} x_{ji} = 1 \quad \text{for all } i.$$

TSP with Arc Variables

Traveling Salesman Problem (TSP):

- n cities, travel costs c_{ij}.
- Binary arc variables $x_{ij} \in \{0, 1\}$:

$$x_{ij} = 1 \iff \text{tour goes directly from } i \text{ to } j.$$

Degree constraints:

$$\sum_{j \neq i} x_{ij} = 1, \qquad \sum_{j \neq i} x_{ji} = 1 \quad \text{for all } i.$$

These ensure each city has exactly one predecessor and one successor.

TSP Without Subtour Elimination

The degree constraints ensure:

$$\sum_{j \neq i} x_{ij} = 1, \quad \sum_{j \neq i} x_{ji} = 1$$

for every city i.

TSP Without Subtour Elimination

The degree constraints ensure:

$$\sum_{j \neq i} x_{ij} = 1, \quad \sum_{j \neq i} x_{ji} = 1$$

for every city i.

But these only make sure every node has one incoming and one outgoing edge, they do **not** guarantee all cities belong to a single tour.

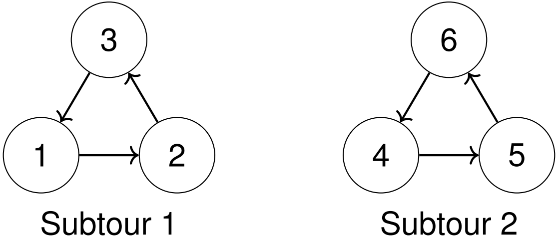
TSP Without Subtour Elimination

The degree constraints ensure:

$$\sum_{j \neq i} x_{ij} = 1, \quad \sum_{j \neq i} x_{ji} = 1$$

for every city i.

But these only make sure every node has one incoming and one outgoing edge, they do **not** guarantee all cities belong to a single tour.



Idea of MTZ (Miller-Tucker-Zemlin): Learn the Visit Order

Goal: forbid subtours by giving each city an *order* in the tour.

Idea of MTZ (Miller-Tucker-Zemlin): Learn the Visit Order

Goal: forbid subtours by giving each city an *order* in the tour.
Introduce **continuous variables**:

$$u_i \in [1, n] \quad \text{for } i = 1, \ldots, n,$$

where u_i is the position of city i in the tour.

Idea of MTZ (Miller-Tucker-Zemlin): Learn the Visit Order

Goal: forbid subtours by giving each city an *order* in the tour.
Introduce **continuous variables**:

$$u_i \in [1, n] \quad \text{for } i = 1, \ldots, n,$$

where u_i is the position of city i in the tour.

- If the tour goes from i to j ($x_{ij} = 1$), then city j must be visited *later* than city i.

Idea of MTZ (Miller-Tucker-Zemlin): Learn the Visit Order

Goal: forbid subtours by giving each city an *order* in the tour.
Introduce **continuous variables**:

$$u_i \in [1, n] \quad \text{for } i = 1, \ldots, n,$$

where u_i is the position of city i in the tour.

- If the tour goes from i to j ($x_{ij} = 1$), then city j must be visited *later* than city i.

$$x_{ij} = 1 \quad \Rightarrow \quad u_j \geq u_i + 1 \quad \forall i \neq j, i, j \in \{2, ..., n\}.$$

Idea of MTZ (Miller-Tucker-Zemlin): Learn the Visit Order

Goal: forbid subtours by giving each city an *order* in the tour.
Introduce **continuous variables**:

$$u_i \in [1, n] \quad \text{for } i = 1, \dots, n,$$

where u_i is the position of city i in the tour.

- If the tour goes from i to j ($x_{ij} = 1$), then city j must be visited *later* than city i.

$$x_{ij} = 1 \quad \Rightarrow \quad u_j \geq u_i + 1 \quad \forall i \neq j, i, j \in \{2, ..., n\}.$$

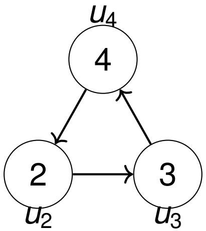
This is exactly a **logical implication** like our earlier patterns.

MTZ Eliminates Subtours (Not Involving City 1)

MTZ enforces a strictly increasing visit order along selected arcs *among cities* $\{2, \ldots, n\}$.

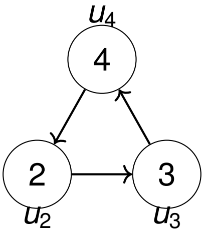
MTZ Eliminates Subtours (Not Involving City 1)

MTZ enforces a strictly increasing visit order along selected arcs *among cities* $\{2, \ldots, n\}$.



MTZ Eliminates Subtours (Not Involving City 1)

MTZ enforces a strictly increasing visit order along selected arcs *among cities* $\{2, \ldots, n\}$.

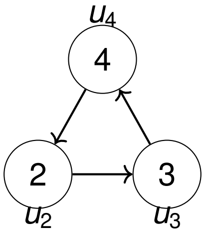


Along the subtour, MTZ requires:

$$u_3 \geq u_2 + 1, \quad u_2 \geq u_4 + 1, \quad u_4 \geq u_3 + 1,$$

MTZ Eliminates Subtours (Not Involving City 1)

MTZ enforces a strictly increasing visit order along selected arcs *among cities* $\{2, \ldots, n\}$.



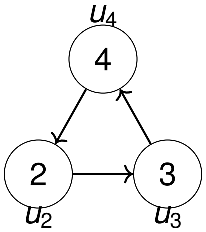
Along the subtour, MTZ requires:

$$u_3 \geq u_2 + 1, \quad u_2 \geq u_4 + 1, \quad u_4 \geq u_3 + 1,$$

which is impossible.

MTZ Eliminates Subtours (Not Involving City 1)

MTZ enforces a strictly increasing visit order along selected arcs *among cities* $\{2, \ldots, n\}$.



Along the subtour, MTZ requires:

$$u_3 \geq u_2 + 1, \quad u_2 \geq u_4 + 1, \quad u_4 \geq u_3 + 1,$$

which is impossible.

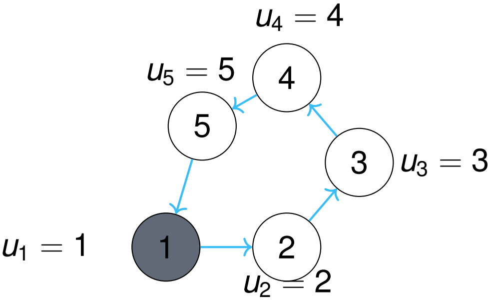
Conclusion: MTZ forbids every cycle that does not include city 1.

Why the One Big Tour Is Still Allowed

City 1 is treated as a special anchor: no MTZ constraints on arcs involving city 1.

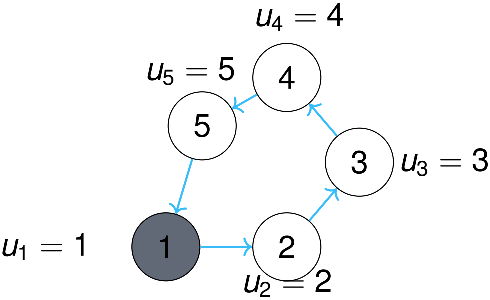
Why the One Big Tour Is Still Allowed

City 1 is treated as a special anchor: no MTZ constraints on arcs involving city 1.



Why the One Big Tour Is Still Allowed

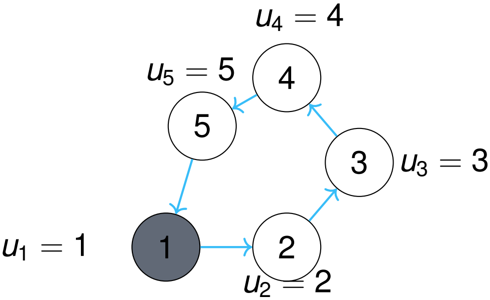
City 1 is treated as a special anchor: no MTZ constraints on arcs involving city 1.



The increasing order holds along the tour, and the final arc back to city 1 is allowed because it is *not constrained by MTZ*.

Why the One Big Tour Is Still Allowed

City 1 is treated as a special anchor: no MTZ constraints on arcs involving city 1.



The increasing order holds along the tour, and the final arc back to city 1 is allowed because it is *not constrained by MTZ*.

Key point: MTZ eliminates all subtours, while deliberately allowing TSP cycles that pass through city 1.

From Implication to Big-M

Desired logic for each pair $i \neq j, i, j \in \{2, \ldots, n\}$:

$$x_{ij} = 1 \quad \Rightarrow \quad u_j \geq u_i + 1.$$

From Implication to Big-M

Desired logic for each pair $i \neq j, i, j \in \{2, ..., n\}$:

$$x_{ij} = 1 \quad \Rightarrow \quad u_j \geq u_i + 1.$$

Rewrite as an inequality:

$$u_j - u_i - 1 \;\geq\; 0.$$

From Implication to Big-M

Desired logic for each pair $i \neq j, i, j \in \{2, ..., n\}$:

$$x_{ij} = 1 \quad \Rightarrow \quad u_j \geq u_i + 1.$$

Rewrite as an inequality:

$$u_j - u_i - 1 \ \geq \ 0.$$

Now use the same Big-M pattern as before:

> *When the binary is 1, the inequality is enforced; when it is 0, it can be relaxed.*

From Implication to Big-M

Desired logic for each pair $i \neq j, i, j \in \{2, ..., n\}$:

$$x_{ij} = 1 \quad \Rightarrow \quad u_j \geq u_i + 1.$$

Rewrite as an inequality:

$$u_j - u_i - 1 \ \geq \ 0.$$

Now use the same Big-M pattern as before:

> *When the binary is 1, the inequality is enforced; when it is 0, it can be relaxed.*

So we **gate** the inequality with x_{ij}:

$$u_j - u_i - 1 \ \geq \ -M(1 - x_{ij}).$$

From Implication to Big-M

Desired logic for each pair $i \neq j, i, j \in \{2, ..., n\}$:

$$x_{ij} = 1 \quad \Rightarrow \quad u_j \geq u_i + 1.$$

Rewrite as an inequality:

$$u_j - u_i - 1 \ \geq \ 0.$$

Now use the same Big-M pattern as before:

> *When the binary is 1, the inequality is enforced; when it is 0, it can be relaxed.*

So we **gate** the inequality with x_{ij}:

$$u_j - u_i - 1 \ \geq \ -M(1 - x_{ij}).$$

- If $x_{ij} = 1$: we get $u_j - u_i - 1 \geq 0 \iff u_j \geq u_i + 1$.

From Implication to Big-M

Desired logic for each pair $i \neq j, i, j \in \{2, ..., n\}$:

$$x_{ij} = 1 \quad \Rightarrow \quad u_j \geq u_i + 1.$$

Rewrite as an inequality:

$$u_j - u_i - 1 \ \geq \ 0.$$

Now use the same Big-M pattern as before:

> *When the binary is 1, the inequality is enforced; when it is 0, it can be relaxed.*

So we **gate** the inequality with x_{ij}:

$$u_j - u_i - 1 \ \geq \ -M(1 - x_{ij}).$$

- If $x_{ij} = 1$: we get $u_j - u_i - 1 \geq 0 \iff u_j \geq u_i + 1$.
- If $x_{ij} = 0$: $u_j - u_i - 1 \geq -M$, which has to be true if M is large enough.

Choosing M from Variable Bounds

We know $u_i \in [1, n]$ for $i = 1, \ldots, n$.

Choosing M from Variable Bounds

We know $u_i \in [1, n]$ for $i = 1, \ldots, n$.
In the *relaxed* case $x_{ij} = 0$, the constraint is

$$u_j - u_i - 1 \geq -M.$$

Choosing M from Variable Bounds

We know $u_i \in [1, n]$ for $i = 1, \dots, n$.
In the *relaxed* case $x_{ij} = 0$, the constraint is

$$u_j - u_i - 1 \geq -M.$$

Worst case (largest LHS violation) is when u_j is as small and u_i as large as possible:

$$u_j = 1, \quad u_i = n \quad \Rightarrow \quad u_j - u_i - 1 = 1 - n - 1 = -n.$$

Choosing M from Variable Bounds

We know $u_i \in [1, n]$ for $i = 1, \dots, n$.
In the *relaxed* case $x_{ij} = 0$, the constraint is

$$u_j - u_i - 1 \geq -M.$$

Worst case (largest LHS violation) is when u_j is as small and u_i as large as possible:

$$u_j = 1, \quad u_i = n \quad \Rightarrow \quad u_j - u_i - 1 = 1 - n - 1 = -n.$$

To ensure this is always allowed when $x_{ij} = 0$, we need

$$-n \geq -M \quad \Rightarrow \quad M \geq n.$$

Final ILP Formulation: TSP with MTZ

Decision variables

$$x_{ij} \in \{0, 1\} \quad (i \neq j) \qquad u_i \in [1, n] \quad (i = 1, \dots, n)$$

Final ILP Formulation: TSP with MTZ

Decision variables

$$x_{ij} \in \{0, 1\} \quad (i \neq j) \qquad u_i \in [1, n] \quad (i = 1, \ldots, n)$$

Objective: $\min \sum_{i \in V} \sum_{\substack{j \in V \\ j \neq i}} c_{ij} \, x_{ij}$.

Final ILP Formulation: TSP with MTZ

Decision variables

$$x_{ij} \in \{0, 1\} \quad (i \neq j) \qquad u_i \in [1, n] \quad (i = 1, \ldots, n)$$

Objective: $\min \sum_{i \in V} \sum_{\substack{j \in V \\ j \neq i}} c_{ij}\, x_{ij}$.

Degree constraints

$$\sum_{j \neq i} x_{ij} = 1, \qquad \sum_{j \neq i} x_{ji} = 1 \quad \forall i \in V$$

Final ILP Formulation: TSP with MTZ

Decision variables

$$x_{ij} \in \{0,1\} \quad (i \neq j) \qquad u_i \in [1,n] \quad (i = 1,\dots,n)$$

Objective: $\min \sum_{i \in V} \sum_{\substack{j \in V \\ j \neq i}} c_{ij}\, x_{ij}$.

Degree constraints

$$\sum_{j \neq i} x_{ij} = 1, \qquad \sum_{j \neq i} x_{ji} = 1 \quad \forall i \in V$$

MTZ subtour elimination

$$u_j \;\geq\; u_i + 1 - n(1 - x_{ij}) \quad \forall i \neq j,\; i,j \in \{2,\dots,n\} \qquad u_1 = 1$$

Gurobi: TSP with MTZ Constraints

```python
import gurobipy as gp
import numpy as np
n = 1000
V = range(n)                    # cities 0,...,n-1
c = np.random.uniform(size=(n, n)) #random costs
m = gp.Model("TSP_MTZ")
x = m.addVars(V, V, vtype=gp.GRB.BINARY, name="x")
u = m.addVars(V, lb=1, ub=n, name="u")

# Objective
m.setObjective(gp.quicksum(c[i,j] * x[i,j] for i in V for j in V if i != j), gp.GRB.MINIMIZE)

# Degree constraints
for i in V:
    m.addConstr(gp.quicksum(x[i,j] for j in V if j != i) == 1)
    m.addConstr(gp.quicksum(x[j,i] for j in V if j != i) == 1)

# MTZ subtour elimination
for i in range(1, n):
    for j in range(1, n):
        if i != j: m.addConstr(u[j] >= u[i] + 1 - n * (1 - x[i,j]))

m.optimize()
```

A Note on Big-M vs Implication

I replaced these lines:

```python
# MTZ subtour elimination
for i in range(1, n):
    for j in range(1, n):
        if i != j: m.addConstr(u[j] >= u[i] + 1 - n * (1 - x[i,j]))
```

A Note on Big-M vs Implication

I replaced these lines:

```
# MTZ subtour elimination
for i in range(1, n):
    for j in range(1, n):
        if i != j: m.addConstr(u[j] >= u[i] + 1 - n * (1 - x[i,j]))
```

With:

```
# MTZ subtour elimination
for i in range(1, n):
    for j in range(1, n):
        if i!=j: m.addConstr( (x[i,j]==1) >> (u[j]>=u[i]+1))
```

A Note on Big-M vs Implication

I replaced these lines:

```python
# MTZ subtour elimination
for i in range(1, n):
    for j in range(1, n):
        if i != j: m.addConstr(u[j] >= u[i] + 1 - n * (1 - x[i,j]))
```

With:

```python
# MTZ subtour elimination
for i in range(1, n):
    for j in range(1, n):
        if i!=j: m.addConstr( (x[i,j]==1) >> (u[j]>=u[i]+1))
```

Same runtime! So why select M?

A Note on Big-*M* vs Implication

I replaced this line:

```
u = m.addVars(V, lb=1, ub=n, name="u")
```

A Note on Big-*M* vs Implication

I replaced this line:

```
u = m.addVars(V, lb=1, ub=n, name="u")
```

With:

```
u = m.addVars(V, lb=1, ub=2*n, name="u")
```

A Note on Big-M vs Implication

I replaced this line:

```
u = m.addVars(V, lb=1, ub=n, name="u")
```

With:

```
u = m.addVars(V, lb=1, ub=2*n, name="u")
```

Runtime went from 48 seconds (< 1 minute) for carefully chosen $M = n$, to more than 10 minutes...

The Mystery: Why was Gurobi 10x Slower?

The Setup:

- **Manual Big-M with correct bounds:** Runs in $\sim$ 1 minutes.
- **Indicator Constraint (‘$\gg$‘) with loose upper bound:** Runs in 10 minutes.

The Mystery: Why was Gurobi 10x Slower?

The Setup:

- **Manual Big-M with correct bounds:** Runs in $\sim$ 1 minutes.
- **Indicator Constraint (‘$\gg$‘) with loose upper bound:** Runs in 10 minutes.

The Assumption: “Gurobi converts ‘$\gg$‘ to Big-M internally with the best M value, so they should be the same.”

The Mystery: Why was Gurobi 10x Slower?

The Setup:

- **Manual Big-M with correct bounds:** Runs in ~ 1 minutes.
- **Indicator Constraint ('$\gg$') with loose upper bound:** Runs in 10 minutes.

The Assumption: "Gurobi converts '$\gg$' to Big-M internally with the best M value, so they should be the same."

The Reality

The assumption is wrong. Gurobi gets **Weak Bounds (The M Value)** and is conservative with how it chooses M.

Automatic M Selection

Scenario: Three resource variables with a budget constraint.

$$x_1, x_2, x_3 \in [0, 100]$$

Automatic M Selection

Scenario: Three resource variables with a budget constraint.

$$x_1, x_2, x_3 \in [0, 100]$$

Global Constraint: The sum cannot exceed 10.

$$x_1 + x_2 + x_3 \leq 10$$

Automatic M Selection

Scenario: Three resource variables with a budget constraint.

$$x_1, x_2, x_3 \in [0, 100]$$

Global Constraint: The sum cannot exceed 10.

$$x_1 + x_2 + x_3 \leq 10$$

Logical Requirement: If binary $z = 0$, then x_1 must be 0.

$$z = 0 \implies x_1 = 0 \quad (\text{modeled as } x_1 \leq M \cdot z)$$

Automatic M Selection

Scenario: Three resource variables with a budget constraint.

$$x_1, x_2, x_3 \in [0, 100]$$

Global Constraint: The sum cannot exceed 10.

$$x_1 + x_2 + x_3 \leq 10$$

Logical Requirement: If binary $z = 0$, then x_1 must be 0.

$$z = 0 \implies x_1 = 0 \quad (\text{modeled as } x_1 \leq M \cdot z)$$

1. Solver's View (Local) Gurobi looks at the variable object `x[1]`. Declared Upper Bound: 100. "Safe" $M = 100$.

2. Mathematician View (Global) You know $x_2, x_3 \geq 0$. Implied Upper Bound: 10. Tight $M = 10$.

Automatic M Selection

The Relaxation Gap (Suppose we want $x_1 = 5$)

- **Solver's** $M = 100$**:** $5 \leq 100z \implies z \geq$ **0.05**.
- **Engineer's** $M = 10$**:** $5 \leq 10z \implies z \geq$ **0.50**.

The solver allows z to be 10x smaller (weaker), creating a massive search tree.

Automatic M Selection

The Relaxation Gap (Suppose we want $x_1 = 5$)

- **Solver's** $M = 100$**:** $5 \leq 100z \implies z \geq \mathbf{0.05}$.
- **Engineer's** $M = 10$**:** $5 \leq 10z \implies z \geq \mathbf{0.50}$.

The solver allows z to be 10x smaller (weaker), creating a massive search tree.

In MTZ TSP. If we set $lb = 1, ub = n$, then using bound propagation gives $M = n$.

Automatic M Selection

The Relaxation Gap (Suppose we want $x_1 = 5$)

- **Solver's** $M = 100$**:** $5 \leq 100z \implies z \geq$ **0.05**.
- **Engineer's** $M = 10$**:** $5 \leq 10z \implies z \geq$ **0.50**.

The solver allows z to be 10x smaller (weaker), creating a massive search tree.

In MTZ TSP. If we set $lb = 1, ub = n$, then using bound propagation gives $M = n$.

One way to check that is to run (before m.optimize()):

```
p = m.presolve()
p.write("presolved.lp") #This is a text file
```

Automatic M Selection

The Relaxation Gap (Suppose we want $x_1 = 5$)

- **Solver's $M = 100$: $5 \leq 100z \implies z \geq$ 0.05.**
- **Engineer's $M = 10$: $5 \leq 10z \implies z \geq$ 0.50.**

The solver allows z to be 10x smaller (weaker), creating a massive search tree.

In MTZ TSP. If we set $lb = 1$, $ub = n$, then using bound propagation gives $M = n$.

One way to check that is to run (before m.optimize()):

```
p = m.presolve()
p.write("presolved.lp") #This is a text file
```

```
R966526: 1000 x[967,459] - u[459] + u[967] <= 999 //ub=n
```

Automatic M Selection

The Relaxation Gap (Suppose we want $x_1 = 5$)

- **Solver's** $M = 100$: $5 \leq 100z \implies z \geq \mathbf{0.05}$.
- **Engineer's** $M = 10$: $5 \leq 10z \implies z \geq \mathbf{0.50}$.

The solver allows z to be 10x smaller (weaker), creating a massive search tree.

In MTZ TSP. If we set $lb = 1, ub = n$, then using bound propagation gives $M = n$.

One way to check that is to run (before m.optimize()):

```
p = m.presolve()
p.write("presolved.lp") #This is a text file
R966526: 1000 x[967,459] - u[459] + u[967] <= 999 //ub=n
R966526: 2000 x[967,459] - u[459] + u[967] <= 1999 //ub=2n
```

What if we don't set *ub*?

```
u = m.addVars(V, lb=1, name="u")
```

What if we don't set *ub*?

```
u = m.addVars(V, lb=1, name="u")
```

Took longer than 12 hours and did not converge...

What if we don't set *ub*?

```
u = m.addVars(V, lb=1, name="u")
```

Took longer than 12 hours and did not converge...

Gurobi cannot do bound propagation.

What if we don't set *ub*?

```
u = m.addVars(V, lb=1, name="u")
```

Took longer than 12 hours and did not converge...

Gurobi cannot do bound propagation.

The hard-coded M converged, but after $\approx$ 1 hour (compare to 1 minute with properly set $ub = n$).

Engineering Takeaway: Explicit vs. Implicit

Rule of Thumb

- **Use Indicators ('$\gg$')** for:
  - "One-off" logical conditions (e.g., if factory opens, MinProduction $\geq$ 50).
  - Constraints that do not heavily impact the core combinatorial structure.

Engineering Takeaway: Explicit vs. Implicit

Rule of Thumb

- **Use Indicators ('$\gg$')** for:
  - "One-off" logical conditions (e.g., if factory opens, MinProduction $\geq$ 50).
  - Constraints that do not heavily impact the core combinatorial structure.
- **Use Manual Big-M** for:
- Core structural constraints (like TSP subtours, Network Flow).

Engineering Takeaway: Explicit vs. Implicit

Rule of Thumb

- **Use Indicators (‘$\gg$‘)** for:
  - “One-off” logical conditions (e.g., if factory opens, MinProduction $\geq$ 50).
  - Constraints that do not heavily impact the core combinatorial structure.
- **Use Manual Big-M** for:
- Core structural constraints (like TSP subtours, Network Flow).
- When you can derive a **tight** theoretical bound for M.

Engineering Takeaway: Explicit vs. Implicit

Rule of Thumb

- **Use Indicators (‘$\gg$‘)** for:
  - “One-off” logical conditions (e.g., if factory opens, MinProduction $\geq$ 50).
  - Constraints that do not heavily impact the core combinatorial structure.
- **Use Manual Big-M** for:
- Core structural constraints (like TSP subtours, Network Flow).
- When you can derive a **tight** theoretical bound for M.
- When you need the solver to “see” the geometry for cuts (MIR, Gomory).

“Don’t hide the geometry inside a logical wrapper.”

Engineering Takeaway: Explicit vs. Implicit

Rule of Thumb

- **Use Indicators ('$\gg$')** for:
  - "One-off" logical conditions (e.g., if factory opens, MinProduction $\geq$ 50).
  - Constraints that do not heavily impact the core combinatorial structure.
- **Use Manual Big-M** for:
- Core structural constraints (like TSP subtours, Network Flow).
- When you can derive a **tight** theoretical bound for M.
- When you need the solver to "see" the geometry for cuts (MIR, Gomory).

"Don't hide the geometry inside a logical wrapper."
"Always set lb and ub to be as tight as possible for your variables."

1. Case Study: Travelling Salesman Problem (TSP).
2. Mixed Integer Non Linear Programming (MINLP)
3. Limits of Uniform Linearization and Spatial Branch and Bound
4. Summary and Outlook

Why MINLP?

Many real constraints are not linear:

- Power / energy: cost $\sim x^2$.
- Congestion: latency $\sim (\sum_i x_i)^2$.
- Mixing / quality: averages and ratios.
- Physics / finance: products and quotients everywhere.

Warm-up: Approximate a Curve with Line Segments

We want to model $y \approx f(x)$ on an interval $x \in [L, U]$.

Warm-up: Approximate a Curve with Line Segments

We want to model $y \approx f(x)$ on an interval $x \in [L, U]$. Choose breakpoints:

$$(x_0, f_0), (x_1, f_1), \ldots, (x_K, f_K)$$

Then approximate using linear segments between adjacent points.

Convex Combination Form (Core Pattern)

Introduce weights $\lambda_0, \ldots, \lambda_K$:

$$\lambda_k \geq 0, \qquad \sum_{k=0}^{K} \lambda_k = 1.$$

Convex Combination Form (Core Pattern)

Introduce weights $\lambda_0, \ldots, \lambda_K$:

$$\lambda_k \geq 0, \qquad \sum_{k=0}^{K} \lambda_k = 1.$$

Enforce linear constraints:

$$x = \sum_{k=0}^{K} x_k \lambda_k, \qquad y = \sum_{k=0}^{K} f_k \lambda_k.$$

Convex Combination Form (Core Pattern)

Introduce weights $\lambda_0, \ldots, \lambda_K$:

$$\lambda_k \geq 0, \qquad \sum_{k=0}^{K} \lambda_k = 1.$$

Enforce linear constraints:

$$x = \sum_{k=0}^{K} x_k \lambda_k, \qquad y = \sum_{k=0}^{K} f_k \lambda_k.$$

Missing rule: x must lie on *one segment*, not a mixture of far-apart points.

Convex Combination Form (Core Pattern)

Introduce weights $\lambda_0, \ldots, \lambda_K$:

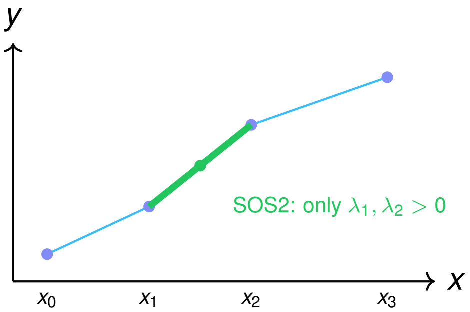
$$\lambda_k \geq 0, \qquad \sum_{k=0}^{K} \lambda_k = 1.$$

Enforce linear constraints:

$$x = \sum_{k=0}^{K} x_k \lambda_k, \qquad y = \sum_{k=0}^{K} f_k \lambda_k.$$

Missing rule: x must lie on *one segment*, not a mixture of far-apart points.
Enter SOS2: at most two adjacent λ_k are nonzero.

A Tiny Picture: PWL Approximation



Gurobi: Piecewise Linear via SOS2

```python
import gurobipy as gp
from gurobipy import GRB

# Breakpoints for y ~ x^2 on [0, 4]
xs = [0, 1, 2, 3, 4]
ys = [x*x for x in xs]
m = gp.Model("pwl_square")

lam = m.addVars(len(xs), lb=0.0, name="lam")
x   = m.addVar(lb=0.0, ub=4.0, name="x")
y   = m.addVar(lb=0.0, name="y")

m.addConstr(gp.quicksum(lam[i] for i in range(len(xs))) == 1)
m.addConstr(x == gp.quicksum(xs[i] * lam[i] for i in range(len(xs))))
m.addConstr(y == gp.quicksum(ys[i] * lam[i] for i in range(len(xs))))

# SOS2: only two adjacent lambdas can be nonzero (ordered by xs)
m.addSOS(GRB.SOS_TYPE2, [lam[i] for i in range(len(xs))], xs)

# Example objective: minimize y subject to x >= 2.4 (forces interpolation)
m.addConstr(x >= 2.4)
m.setObjective(y, GRB.MINIMIZE)
m.optimize()
```

From x^2 to $x \cdot y$

Univariate: x^2 is already nonlinear.

From x^2 to $x \cdot y$

Univariate: x^2 is already nonlinear.
Bivariate: $x \cdot y$ is the basic building block of *most* nonlinear models.

From x^2 to $x \cdot y$

Univariate: x^2 is already nonlinear.
Bivariate: $x \cdot y$ is the basic building block of *most* nonlinear models. Why?
Because once you can express products, you get:

- quadratic terms ($x^2 = x \cdot x$),
- interaction terms ($x_i x_j$),
- and much more complex expressions.

Bilinear Constraints as a Universal Modeling Language

Consider:

$$w = \frac{x + z^2}{x - y}.$$

Bilinear Constraints as a Universal Modeling Language

Consider:

$$w = \frac{x + z^2}{x - y}.$$

Looks like: rational + quadratic + division $\Rightarrow$ hopeless?

Bilinear Constraints as a Universal Modeling Language

Consider:

$$w = \frac{x + z^2}{x - y}.$$

Looks like: rational + quadratic + division $\Rightarrow$ hopeless?
Modeling trick: introduce variables for subexpressions.

Subexpression Variables

Introduce:

$$t = z^2, \qquad u = x + t, \qquad v = x - y.$$

Then:

$$w = \frac{u}{v}.$$

Subexpression Variables

Introduce:

$$t = z^2, \qquad u = x + t, \qquad v = x - y.$$

Then:

$$w = \frac{u}{v}.$$

Ratios become bilinear constraints:

$$w = \frac{u}{v} \iff u = w \cdot v.$$

Final Reformulation (Only Bilinear + Linear)

The original expression is equivalent to:

$$
\begin{aligned}
t &= z \cdot z, \\
u &= x + t, \\
v &= x - y, \\
u &= w \cdot v.
\end{aligned}
$$

Grid Linearization for $z \approx xy$ (Idea)

If $x \in [L_x, U_x]$ and $y \in [L_y, U_y]$:

- discretize x into K bins, y into K bins,
- select a cell using binaries,
- interpolate inside the cell (or use a bilinear plane per cell).

Grid Linearization for $z \approx xy$ (Idea)

If $x \in [L_x, U_x]$ and $y \in [L_y, U_y]$:

- discretize x into K bins, y into K bins,
- select a cell using binaries,
- interpolate inside the cell (or use a bilinear plane per cell).

Cost: K in 1D becomes K^2 in 2D.
Small ϵ (high resolution) $\Rightarrow$ very large MILP.

Modeling Tradeoff: Accuracy vs Complexity

- PWL accuracy improves as segment length shrinks.
- But: number of variables/constraints grows quickly.

Modeling Tradeoff: Accuracy vs Complexity

- PWL accuracy improves as segment length shrinks.
- But: number of variables/constraints grows quickly.

Rule of thumb:

$$\text{Model size} \approx (\text{resolution})^{-d}$$

where d is the number of continuous dimensions you linearize.

1 Case Study: Travelling Salesman Problem (TSP).

2 Mixed Integer Non Linear Programming (MINLP)

3 Limits of Uniform Linearization and Spatial Branch and Bound

4 Summary and Outlook

Why Uniform Grids Are Wasteful

Uniform linearization spends the same resolution everywhere.

Why Uniform Grids Are Wasteful

Uniform linearization spends the same resolution everywhere.
But the optimum typically lives in a tiny part of the domain:

Why Uniform Grids Are Wasteful

Uniform linearization spends the same resolution everywhere.
But the optimum typically lives in a tiny part of the domain:

- most cells/segments are never used,

Why Uniform Grids Are Wasteful

Uniform linearization spends the same resolution everywhere.
But the optimum typically lives in a tiny part of the domain:

- most cells/segments are never used,
- we pay the variable/constraint cost anyway.

Why Uniform Grids Are Wasteful

Uniform linearization spends the same resolution everywhere.
But the optimum typically lives in a tiny part of the domain:

- most cells/segments are never used,
- we pay the variable/constraint cost anyway.

Question: Can we refine only where it matters?

McCormick Envelopes on The Atomic Constraint: $z = xy$

Suppose:

$$x \in [L_x, U_x], \qquad y \in [L_y, U_y], \qquad z = xy.$$

McCormick Envelopes on The Atomic Constraint: $z = xy$

Suppose:

$$x \in [L_x, U_x], \qquad y \in [L_y, U_y], \qquad z = xy.$$

The set $\{(x, y, z) : z = xy\}$ is nonconvex.

McCormick Envelopes on The Atomic Constraint: $z = xy$

Suppose:

$$x \in [L_x, U_x], \qquad y \in [L_y, U_y], \qquad z = xy.$$

The set $\{(x, y, z) : z = xy\}$ is nonconvex.
Solvers start with a convex outer approximation: **McCormick envelopes**.

McCormick Envelopes (4 Inequalities)

Define $z = xy$ with bounds $x \in [L_x, U_x]$, $y \in [L_y, U_y]$.
The convex hull relaxation over the box is:

$$
\begin{aligned}
z &\geq L_x y + L_y x - L_x L_y, \\
z &\geq U_x y + U_y x - U_x U_y, \\
z &\leq U_x y + L_y x - U_x L_y, \\
z &\leq L_x y + U_y x - L_x U_y.
\end{aligned}
$$

McCormick Envelopes (4 Inequalities)

Define $z = xy$ with bounds $x \in [L_x, U_x]$, $y \in [L_y, U_y]$.
The convex hull relaxation over the box is:

$$
\begin{aligned}
z &\geq L_x y + L_y x - L_x L_y, \\
z &\geq U_x y + U_y x - U_x U_y, \\
z &\leq U_x y + L_y x - U_x L_y, \\
z &\leq L_x y + U_y x - L_x U_y.
\end{aligned}
$$

Interpretation:

- These are the tightest possible linear outer bounds on xy over the box.

McCormick Envelopes (4 Inequalities)

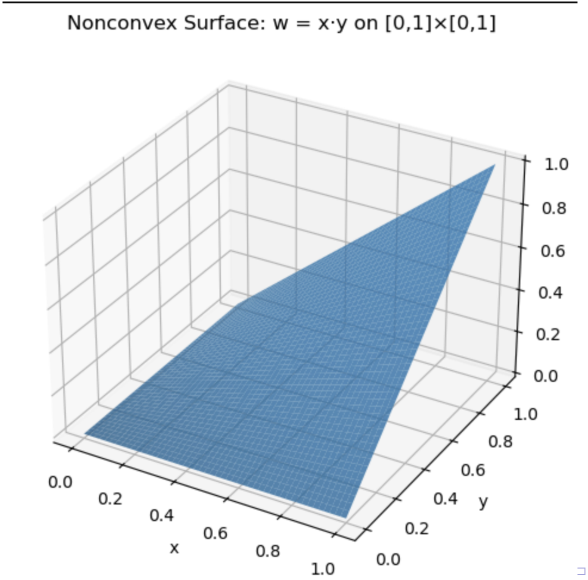
Define $z = xy$ with bounds $x \in [L_x, U_x]$, $y \in [L_y, U_y]$.
The convex hull relaxation over the box is:

$$\begin{aligned} z &\geq L_x y + L_y x - L_x L_y, \\ z &\geq U_x y + U_y x - U_x U_y, \\ z &\leq U_x y + L_y x - U_x L_y, \\ z &\leq L_x y + U_y x - L_x U_y. \end{aligned}$$

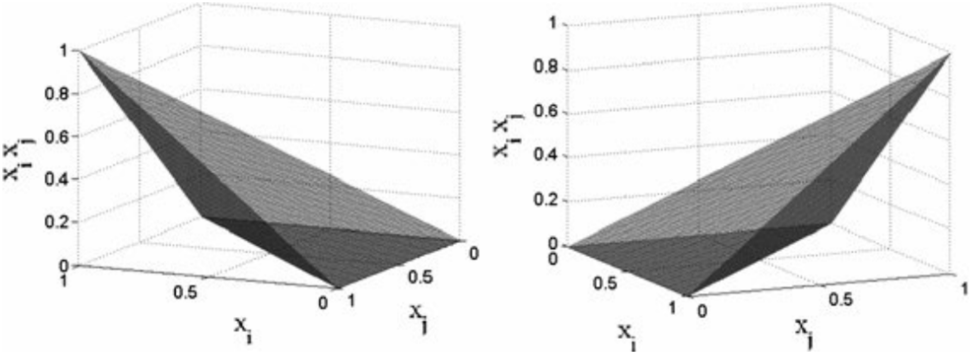
Interpretation:

- These are the tightest possible linear outer bounds on xy over the box.
- Still can be very loose when the box is large.

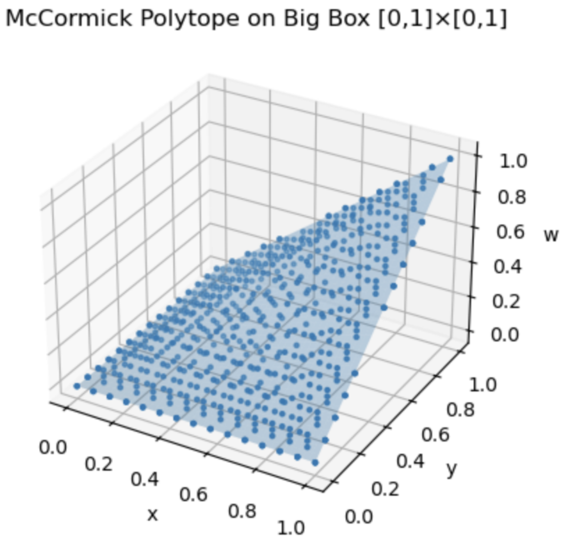
Geometry


Nonconvex Surface: w = x·y on [0,1]×[0,1]
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
x
0.0
0.2
0.4
0.6
0.8
1.0
y

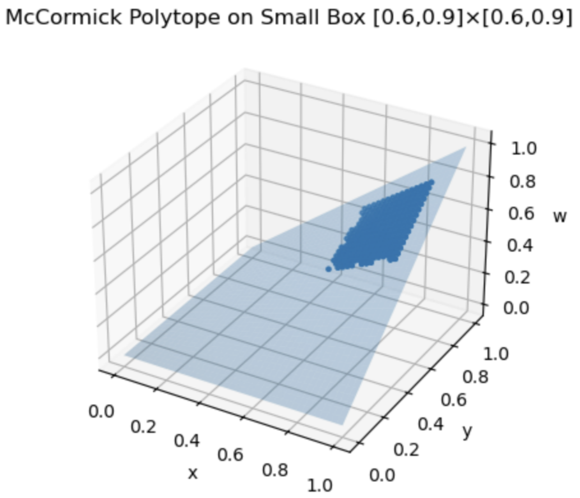
Geometry



Geometry


McCormick Polytope on Big Box [0,1]×[0,1]
0.0
0.2
0.4
0.6
0.8
1.0
x
0.0
0.2
0.4
0.6
0.8
1.0
y
0.0
0.2
0.4
0.6
0.8
1.0
w

Geometry


McCormick Polytope on Small Box [0.6,0.9]×[0.6,0.9]
1.0
0.8
0.6
0.4
0.2
0.0
w
x
0.0
0.2
0.4
0.6
0.8
1.0
y
0.0
0.2
0.4
0.6
0.8
1.0

Spatial B&B: Adaptive Refinement of Bounds

If McCormick is loose over a big box, tighten it by splitting the box:

$$x \in [L_x, U_x] \Rightarrow [L_x, \frac{L_x + U_x}{2}] \cup [\frac{L_x + U_x}{2}, U_x].$$

Spatial B&B: Adaptive Refinement of Bounds

If McCormick is loose over a big box, tighten it by splitting the box:

$$x \in [L_x, U_x] \Rightarrow [L_x, \frac{L_x + U_x}{2}] \cup [\frac{L_x + U_x}{2}, U_x].$$

At each node:

- shrink variable bounds,

Spatial B&B: Adaptive Refinement of Bounds

If McCormick is loose over a big box, tighten it by splitting the box:

$$x \in [L_x, U_x] \Rightarrow [L_x, \frac{L_x + U_x}{2}] \cup [\frac{L_x + U_x}{2}, U_x].$$

At each node:

- shrink variable bounds,
- rebuild McCormick envelopes on the smaller box for bi-linear terms,

Spatial B&B: Adaptive Refinement of Bounds

If McCormick is loose over a big box, tighten it by splitting the box:

$$x \in [L_x, U_x] \Rightarrow [L_x, \frac{L_x + U_x}{2}] \cup [\frac{L_x + U_x}{2}, U_x].$$

At each node:

- shrink variable bounds,
- rebuild McCormick envelopes on the smaller box for bi-linear terms,
- solve the resulting relaxation to get a bound,

Spatial B&B: Adaptive Refinement of Bounds

If McCormick is loose over a big box, tighten it by splitting the box:

$$x \in [L_x, U_x] \Rightarrow [L_x, \frac{L_x + U_x}{2}] \cup [\frac{L_x + U_x}{2}, U_x].$$

At each node:

- shrink variable bounds,
- rebuild McCormick envelopes on the smaller box for bi-linear terms,
- solve the resulting relaxation to get a bound,
- prune if bound is bad, or continue branching.

Spatial B&B: Adaptive Refinement of Bounds

If McCormick is loose over a big box, tighten it by splitting the box:

$$x \in [L_x, U_x] \Rightarrow [L_x, \frac{L_x + U_x}{2}] \cup [\frac{L_x + U_x}{2}, U_x].$$

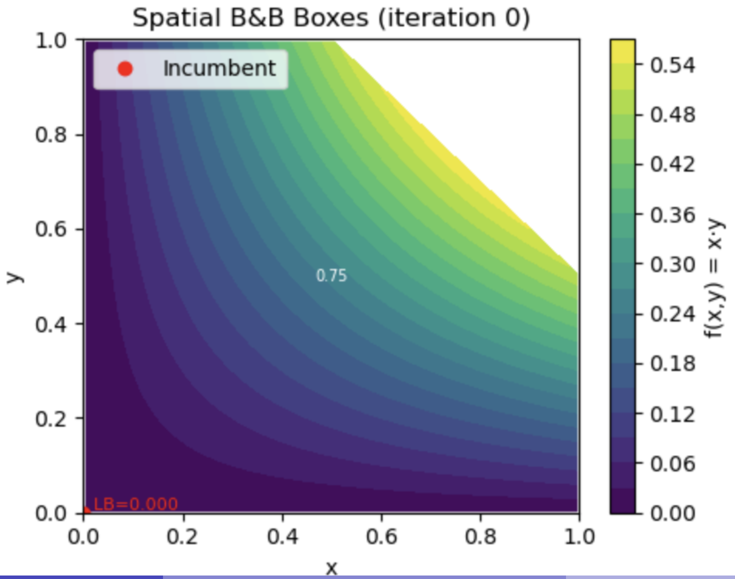
At each node:

- shrink variable bounds,
- rebuild McCormick envelopes on the smaller box for bi-linear terms,
- solve the resulting relaxation to get a bound,
- prune if bound is bad, or continue branching.

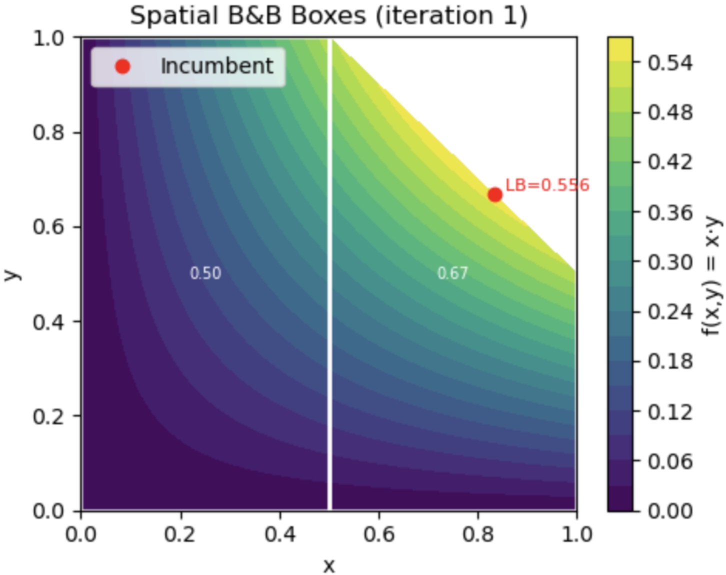
This is **branch-and-bound on continuous domains.** What Gurobi implements.

Example: $\max xy$ s.t $x + y \leq 1.5, 0 \leq x, y \leq 1.$

Geometry

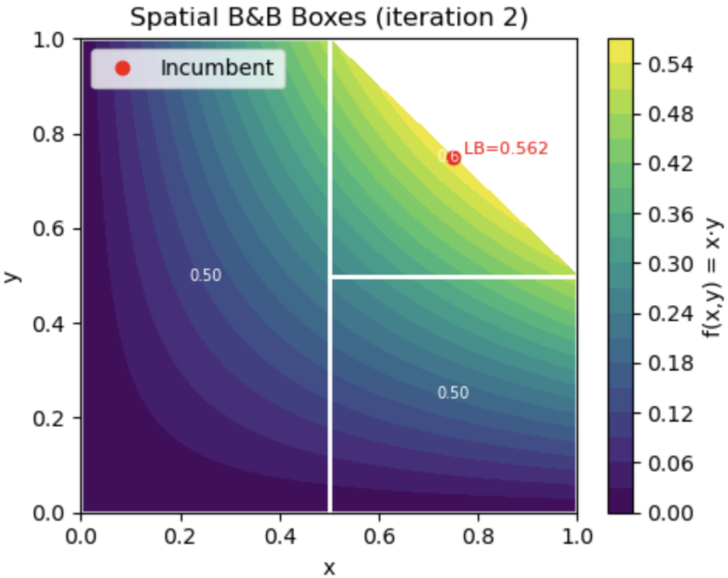

Spatial B&B Boxes (iteration 0)
Incumbent
0.75
LB=0.000
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x
0.54
0.48
0.42
0.36
0.30
0.24
0.18
0.12
0.06
0.00
f(x,y) = x·y

Geometry



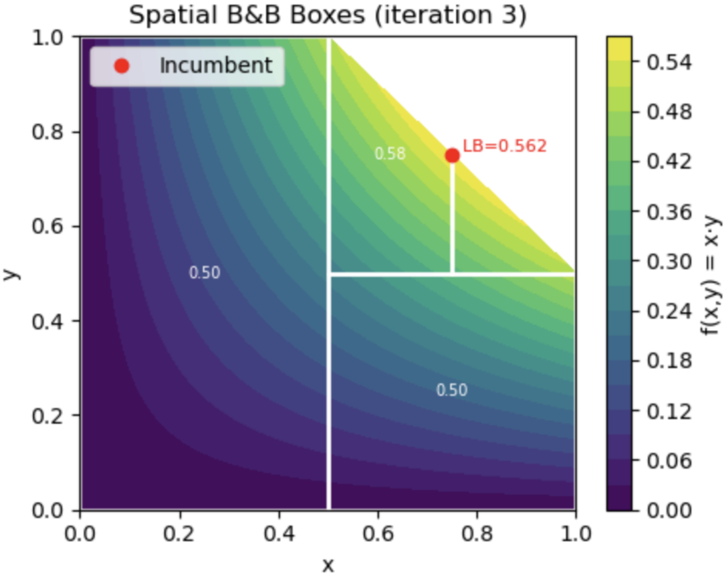
Spatial B&B Boxes (iteration 1)
Incumbent
LB=0.556
0.50
0.67
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x
0.54
0.48
0.42
0.36
0.30
0.24
0.18
0.12
0.06
0.00
f(x,y) = x·y

Geometry



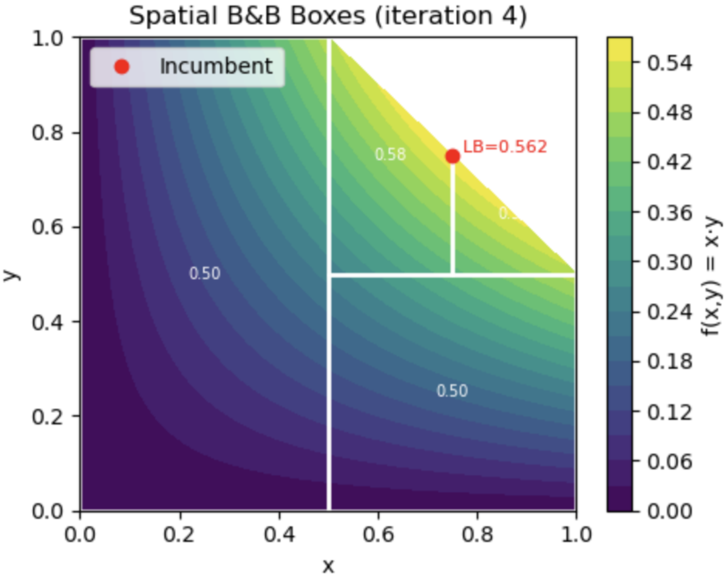
Spatial B&B Boxes (iteration 2)
Incumbent
LB=0.562
0.50
0.50
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
x
y
0.54
0.48
0.42
0.36
0.30
0.24
0.18
0.12
0.06
0.00
f(x,y) = x·y

Geometry



Spatial B&B Boxes (iteration 3)
Incumbent
LB=0.562
0.58
0.50
0.50
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x
0.54
0.48
0.42
0.36
0.30
0.24
0.18
0.12
0.06
0.00
f(x,y) = x·y

Geometry


Spatial B&B Boxes (iteration 4)
Incumbent
LB=0.562
0.58
0.50
0.50
x
y
f(x,y) = x·y
0.0
0.2
0.4
0.6
0.8
1.0
0.00
0.06
0.12
0.18
0.24
0.30
0.36
0.42
0.48
0.54

Practical MINLP in Gurobi

Since version 9.0, Gurobi automates Spatial B&B. You do not need to implement McCormick envelopes manually.

Practical MINLP in Gurobi

Since version 9.0, Gurobi automates Spatial B&B. You do not need to implement McCormick envelopes manually.

The "NonConvex" Parameter

By default, Gurobi assumes quadratic constraints are **convex** (PSD). If you add $z = x \cdot y$ or $y = \sin(x)$, it will throw an error. You must explicitly enable the global solver.

Practical MINLP in Gurobi

Since version 9.0, Gurobi automates Spatial B&B. You do not need to implement McCormick envelopes manually.

The “NonConvex” Parameter

By default, Gurobi assumes quadratic constraints are **convex** (PSD). If you add $z = x \cdot y$ or $y = \sin(x)$, it will throw an error. You must explicitly enable the global solver.

Engineering Note: Just like Big-M, Spatial B&B relies heavily on **variable bounds** (L_x, U_x) to build tight envelopes. Always bound your continuous variables in MINLP!

Practical MINLP in Gurobi

Practical MINLP in Gurobi

```
m = gp.Model("bilinear_example")
x = m.addVar(lb=0, ub=10, name="x")  # Bounds are CRITICAL for envelopes!
y = m.addVar(lb=0, ub=10, name="y")
z = m.addVar(name="z")

# 1. Add the bilinear constraint directly
#    Gurobi detects this is non-convex
m.addConstr(z == x * y)

# 2. REQUIRED: Enable non-convex handling
#    0 = error if non-convex (default)
#    2 = translate to McCormick & use Spatial B&B
m.setParam("NonConvex", 2)

m.optimize()
```

1. Case Study: Travelling Salesman Problem (TSP).
2. Mixed Integer Non Linear Programming (MINLP)
3. Limits of Uniform Linearization and Spatial Branch and Bound
4. **Summary and Outlook**

Summary of Lecture 7

- TSP and its formulation using Big-M.
- Piecewise linearization (PWL) is the simplest bridge from nonlinear to MILP.
- SOS2 = solver-native structure for tight PWL modeling.
- Bilinear terms $x \cdot y$ are the main “modeling atom” of MINLP.
- Many complicated expressions (including ratios) reduce to bilinear equalities via auxiliary variables.
- Uniform approximation is powerful but can explode in size. Deal with it using spatial branch-and-bound and McCormick envelopes.