15. Bike Sharing

金力 Li Jin
li.jin@sjtu.edu.cn
上海交通大学密西根学院
Shanghai Jiao Tong University UM Joint Institute

Recap

- Introduction
- Facility location in Euclidean space
  - k-median problem
  - Coverage problem
- Facility location on networks
  - k-median problem
  - Center problem
  - Requirement problem

Ref: Larson R C, Odoni A R. Urban operations research[M]. 1981.

Outline

- Background
- Formulation
- [Not required] Branch & bound algorithm

[Ref]Mauro, Dell'Amico, Eleni, et al. The bike sharing rebalancing problem: Mathematical formulations and benchmark instances[J]. Omega, 2014.

Background

Bike sharing systems offer a mobility service in which public bicycles are available for shared use.

Motivation

These systems are an important instrument used by public administrations to

1. obtain a more sustainable mobility,
2. decrease traffic and pollution caused by car transportation, and
3. solve the so-called last mile problem related to proximity travels.

Cost

- Operating bike sharing systems has a cost that may vary greatly with a consistent impact on the budget of the public administration.
- (depending on the system itself, the population density, the service area and the fleet size)

Rebalancing

https://haokan.baidu.com/v?pd=wisenatural&vid=2278301419593956991

Station

- Stations are made of different slots, each of which hosts a single bicycle.

Station

- A commonly adopted rule for rebalancing is to keep each station only partly occupied.

Repositioning

- Repositioning is usually done by means of capacitated vehicles based on a central depot that pick up bicycles from stations where the level of occupation is too high and deliver them to stations where the level is too low.

Optimization formulation

It can be modeled as either a *dynamic* or a *static* optimization problem.

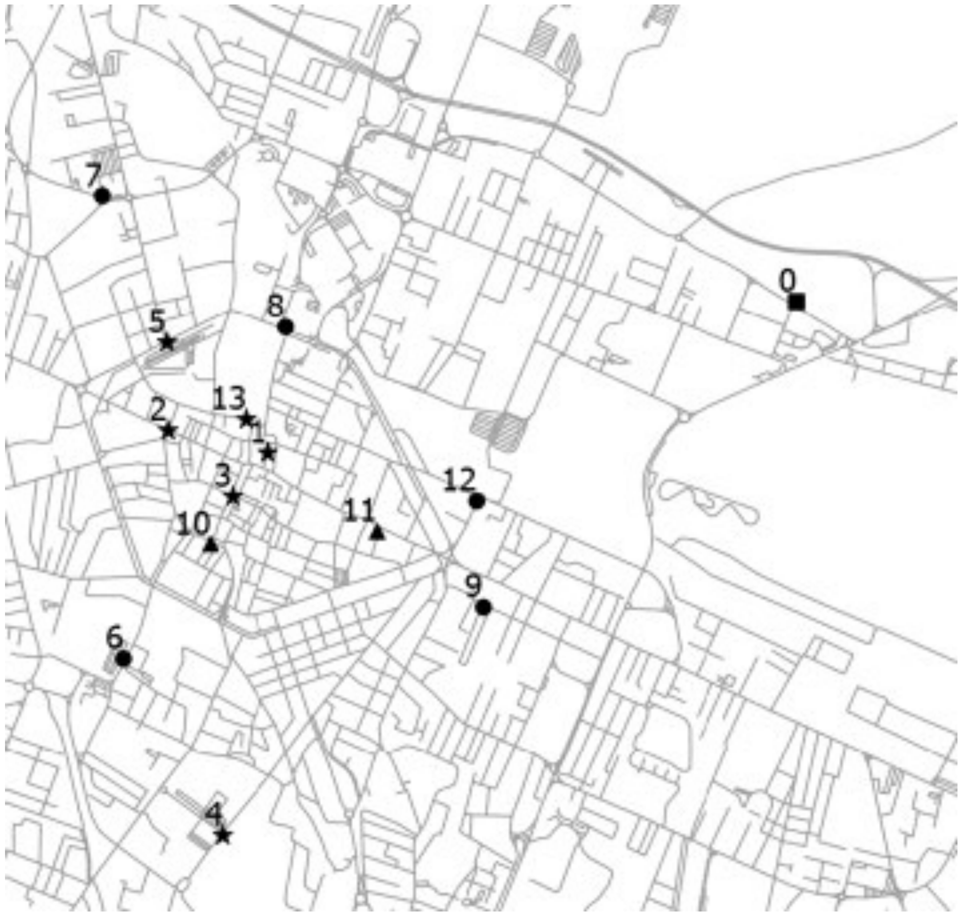
- In the static version, a snapshot of the level of occupation at the stations is taken and then used to plan the redistribution.
- In the dynamic version, the real-time usage of the system is taken into account, and the redistribution plan is possibly updated as soon as the information required to make decisions is revealed over time.

Optimization formulation

- Static rebalancing is associated with a redistribution process that is performed during the night, when the system is kept closed or the demand is very low
- Dynamic rebalancing is associated to redistributions operated during the day, when demand may be high.
- In the real-world case that we studied in detail the redistribution is performed during the night, and hence we focus on the static version of the problem.

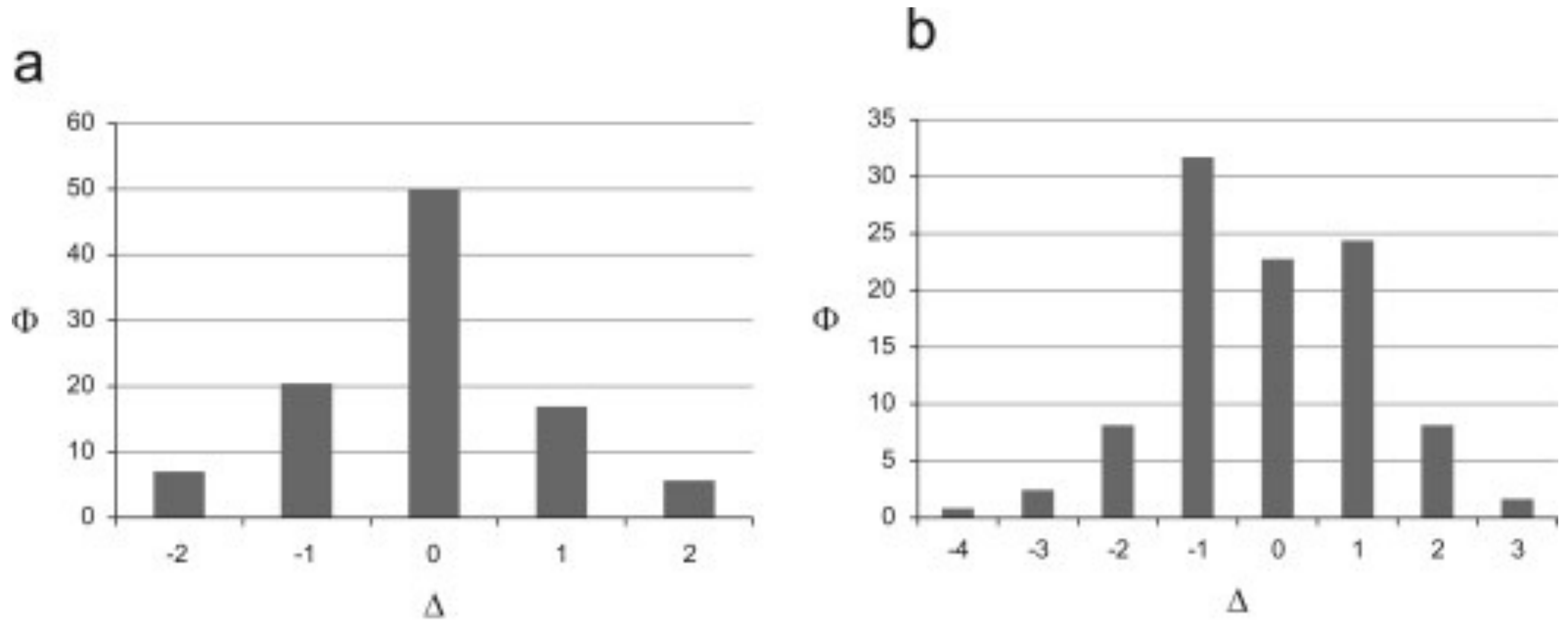
Data analysis of a real-world case

Reggio Emilia, a city of around 170 thousand inhabitants located in a very flat area in northern Italy.



Net flow

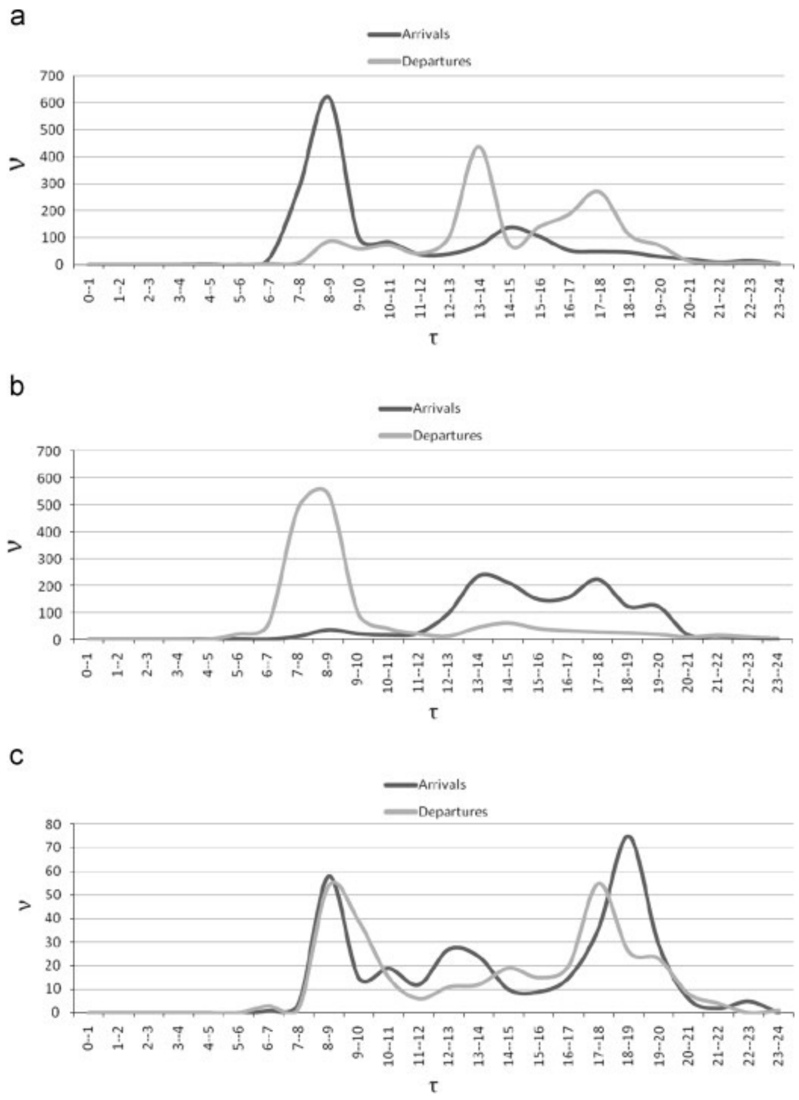
- The data associated with a seven-month usage of the system provided by the municipality of the city.



Net flow of bicycles per day

Groups of stations

- Divide the stations into three groups.
- The first group has a peak of incoming bikes between 7 and 9 am, and a smaller one between 1 and 3 pm.
- The second group is complementary to the first one.
- The third group contains stations where arrivals and departures follow a similar pattern during the day.



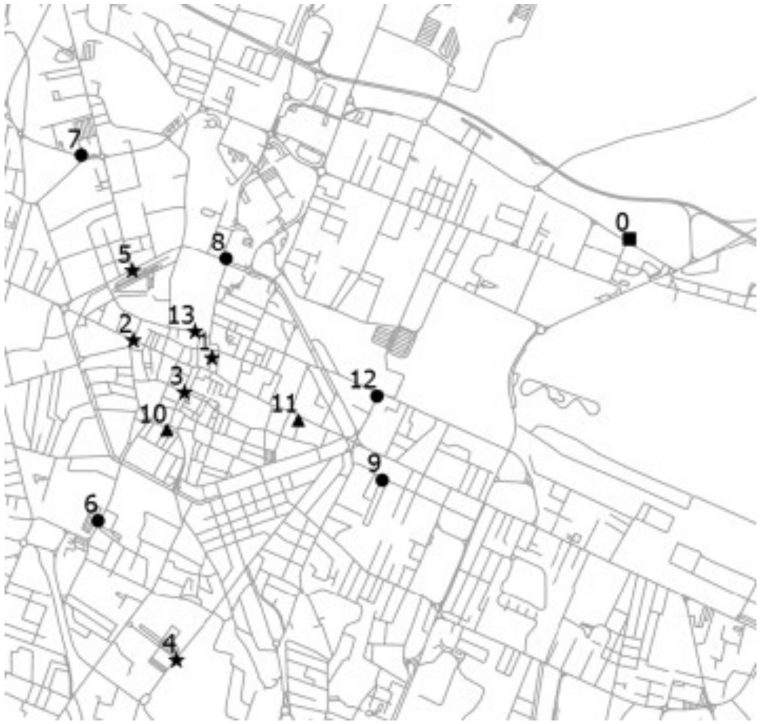
Outline

- Background
- Formulation
- [Not required] Branch & bound algorithm

[Ref]Mauro, Dell'Amico, Eleni, et al. The bike sharing rebalancing problem: Mathematical formulations and benchmark instances[J]. Omega, 2014.

Problem description

- We are given a complete digraph $G = (V, A)$, where the set of vertices $V = \{0,1,\dots,n\}$ is partitioned into the depot, vertex 0, and the stations, vertices $\{1,2,\dots,n\}$.



Request

- Each station i has a request q_i, which can be either positive or negative.
- If $q_i \geq 0$ then i is a *pickup* node, where q_i bikes should be removed.
- If $q_i < 0$ then i is a *delivery* node, where q_i bikes should be supplied.
- The bikes removed from pickup nodes can either go to a delivery node or back to the depot.
- Bikes supplied to delivery nodes can either come from the depot or from pickup nodes.

Shipping vehicles

- A fleet of m identical vehicles of capacity Q is available at the depot to transport the bikes.
- A traveling cost c_{ij} is associated with each arc $(i, j) \in A$.
- The Bike sharing Rebalancing Problem (BRP) involves determining how to drive at most *m* vehicles through the graph, with the aim of minimizing the total cost.

Constraints

The BRP ensures the following constraints:

1. each vehicle performs a route that starts and ends at the depot,
2. each vehicle starts from the depot empty or with some initial load (i.e., with a number of bikes that vary from 0 to Q),
3. each station is visited exactly once and its request is completely fulfilled by the vehicle visiting it, and
4. the sum of requests of the visited stations plus the initial load is never negative or greater than Q in the route performed by a vehicle.

Discussion

- Each request q_i is computed as the difference between the number of bikes present at station i when performing the redistribution, and the number of bikes in the station in the final required configuration.
- Note that a station with request $q_i = 0$ must be visited, even if this implies that no bike has to be dropped off or picked up there.
- This case arises, for example, when the driver of the vehicle is supposed to check that the station is correctly working.
- The case in which stations with null requests have to be skipped can be simply obtained by removing in a preprocessing phase those stations from the set of vertices.

Discussion

- The fact that each vehicle is allowed to start its route with some bikes enlarges the space of feasible BRP solutions, and allows to obtain a more flexible redistribution plan.
- Note also that we do not impose the sum of redistributed bikes to be null, and hence there can be a positive or a negative flow of bikes on the depot.
- This consideration is useful to model cases in which some bikes enter or leave the depot for maintenance.

Discussion

- The traveling cost c_{ij} is computed as the shortest length of a path in the road network connecting i and j, for $i, j \in V$.
- It is important to work on a directed graph, because bike sharing systems are usually located in urban areas, and thus one-way streets typically have a strong impact on the choice of the routes performed by the vehicles during the redistribution.
- The BRP belongs to the wide class of *Pickup and Delivery Vehicle Routing Problems* (PDVRPs), where a fleet of vehicles is used to transport requests from the depot and/or some nodes to the depot and/or other nodes in the network.

Multiple Traveling Salesman Problem

- Let’s begin with a simplified formulation:

  Multiple Traveling Salesman Problem (m-TSP).

- At most m uncapacitated vehicles based on a central depot have to visit a set of vertices, with the constraint that each vertex is visited exactly once.
- Decision variable

$$x_{ij} = \begin{cases} 1 & \text{if arc } (i, j) \text{ is used by a vehicle} \\ 0 & \text{o. w.} \end{cases}$$

- A two-index formulation (i.e., i, j) that does not track individual vehicles.

Multiple Traveling Salesman Problem

m-TSP:

min $\sum_{i\in V}\sum_{j\in V} c_{ij}x_{ij}$ (total travel cost)

s.t. $\sum_{j\in V} x_{ij} = 1, \forall i \in V\backslash\{0\}$, (every node visited once)

$\sum_{j\in V} x_{ji} = 1, \forall i \in V\backslash\{0\}$, (every node visited once)

$\sum_{i\in V} x_{0i} \le m$, (at most m vehicles)

$\sum_{i\in V\backslash\{0\}} x_{i0} = \sum_{i\in V\backslash\{0\}} x_{0i}$, (all vehicles return to depot)

$\sum_{i\in S}\sum_{j\in S} x_{ij} \le |S| - 1, \forall S \subseteq V\backslash\{0\}, S \neq \emptyset$, (sub-tour elimination)

$x_{ij} \in \{0,1\}, \forall (i,j) \in A$. (binary)

Multiple Traveling Salesman Problem

- Note that the above formulation ignores the vehicle capacity (i.e., Q) constraint.
- Consequently, feasibility of the m-TSP problem does not guarantee feasibility of the BRP.
- To guarantee the feasibility of a solution with respect to the BRP, we need to include additional constraints in the m-TSP formulation, so as to ensure that demands are satisfied and vehicle capacities are not exceeded.
- This may be done in different ways.
- Next, let's look at four different formulations.

Formulation 1

- The first option is to define an additional continuous variable θ_j, representing the load of a vehicle after it visited node j, for $j \in V$.
- The load should be updated along the route by taking into consideration the fact that, if the vehicle travels along arc (i, j), then θ_j should be equal to $\theta_i + q_j$.
- This leads to non-linear constraints:

$$\theta_j \geq \left(\theta_i + q_j\right)x_{ij},\ \ i \in V, j \in V\backslash\{0\};\text{ (flow conservation)}$$

$$\theta_i \geq \left(\theta_j - q_j\right)x_{ij},\ \ i \in V\backslash\{0\}, j \in V;\text{ (flow conservation)}$$

$$\max\{0, q_j\} \leq \theta_j \leq \min\{Q, Q + q_j\}, j \in V.\text{ (capacity)}$$

Formulation 1

- These constraints can be linearized with the standard "big M" method, obtaining:

$$\theta_j \geq \theta_i + q_j - M_1(1 - x_{ij}), \text{ (flow conservation)}$$

$$\theta_i \geq \theta_j - q_j - M_2\left(1 - x_{ij}\right). \text{ (flow conservation)}$$

- We can select the big M to be

$$M_1 = Q + \max_{j \in V} q_j$$

$$M_2 = Q - \min_{j \in V} q_j \text{ or } M_2 = \max\{M_1, M_2\}$$

Formulation 1

$$
\begin{aligned}
\text{min } & \sum_{i\in V}\sum_{j\in V} c_{ij}x_{ij} \\
\text{s.t. } & \sum_{j\in V} x_{ij} = 1, \forall i \in V\backslash\{0\}, \\
& \sum_{j\in V} x_{ji} = 1, \forall i \in V\backslash\{0\}, \\
& \sum_{i\in V} x_{0i} \le m, \\
& \sum_{i\in V\backslash\{0\}} x_{i0} = \sum_{i\in V\backslash\{0\}} x_{0i} \\
& \sum_{i\in S}\sum_{j\in S} x_{ij} \le |S| - 1, \forall S \subseteq V\backslash\{0\}, S \neq \emptyset, \\
& \theta_j \ge \theta_i + q_j - M_1(1 - x_{ij}), \\
& \theta_i \ge \theta_j - q_j - M_2\left(1 - x_{ij}\right), \\
& \max\left\{0, q_j\right\} \le \theta_j \le \min\left\{Q, Q + q_j\right\}, j \in V. \\
& x_{ij} \in \{0,1\}, \forall (i,j) \in A.
\end{aligned}
$$

Formulation 2

- The second formulation also builds upon the *m*-TSP, but makes use of an additional variable f_{ij} giving the *flow* over arc (i,j), i.e., the load on the vehicle traveling along arc (i,j), if any, for $(i,j) \in A$.

$$\sum_{i \in V} f_{ji} - \sum_{j \in V} f_{ij} = q_j, \qquad j \in V \backslash \{0\};$$

$$\max\{0, q_i, -q_j\} x_{ij} \le f_{ij}$$
$$\le \min\{Q, Q + q_i, Q - q_j\} x_{ij}, (i,j) \in A.$$

Formulation 3

- We start again from the *m*-TSP, but ensure that solutions satisfy demands, without exceeding the vehicle capacity, by adding to the model the following family of constraints:

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \leq |S| - \max\left\{1, \left\lceil \frac{\sum_{i\in S} q_i}{Q} \right\rceil\right\},$$

$$S \subseteq V\backslash\{0\}, S \neq \emptyset.$$

- Generalized subtour elimination constraints.
- They state that, for each subset S of vertices, the number of arcs with both tail and head in S should not exceed the cardinality of S minus the minimum number of vehicles required to serve S.

Formulation 4

- This formulation first requires to modify the graph by adding a copy of the depot, defined in the following by a new vertex $n+1$.
- We define $\tilde{V} = V \cup \{n+1\}$ the resulting set of vertices.
- And $\tilde{A} = A \cup \{(j, n+1): j \in \tilde{V}\} \cup \{(n+1, j): j \in \tilde{V}\}$ the resulting set of arcs.
- Let us als define $\tilde{V}_0 = \tilde{V} \backslash \{0, n+1\}$ and $Q_{tot} = \sum_{j \in \tilde{V}_0} q_j$.
- We set $q_{n+1} = 0$, $c_{n+1,j} = +\infty$, and $c_{j,n+1} = c_{j0}$ for $j \in \tilde{V}$, knowing that $c_{0,n+1} = 0$, and then we set $c_{j0} = +\infty$ for $j \in \tilde{V}$.

Formulation 4

- We use three sets of variables.
- The first two have already been used in formulation F2:

$$x_{ij} = \begin{cases} 1 & \text{if arc } (i, j) \text{ is used by a vehicle} \\ 0 & \text{o. w.} \end{cases}, (i, j) \in \tilde{A}.$$

and f_{ij} gives the flow over arc (i, j), i.e., the load of the vehicle passing over arc (i, j), if any, for $(i, j) \in \tilde{A}$.

- The third variable is another continuous variable g_{ji} that gives the residual space of the vehicle traveling along arc (i, j), if any, for $(i, j) \in \tilde{A}$.

Formulation 4

- In this formulation, vehicles leave the initial depot, vertex 0, visit customers, and then end their route in the final depot, vertex $n + 1$.
- Variable f_{ij} gives the load along the route performed by the vehicle, starting from the initial depot, vertex 0, following the vertices visited by the route and then ending in the final depot, vertex $n + 1$.
- In the opposite way, g_{ji} gives the free space on the vehicle, starting from $n + 1$, retracing back the route and then ending 0.

Formulation 4

$$f_{ij} + g_{ji} = Qx_{ij} \quad \left(i,j\right) \in \tilde{A}$$

$$\sum_{i\in\tilde{V}} \left(f_{ji} - g_{ij}\right) - \sum_{i\in\tilde{V}} \left(f_{ij} - g_{ji}\right) = 2q_j, \quad j \in \tilde{V}_0$$

$$\sum_{j\in\tilde{V}_0} f_{0j} \geq \max\{0, -Q_{tot}\}$$

$$\sum_{j\in\tilde{V}_0} f_{j,n+1} \geq \max\{0, Q_{tot}\}$$

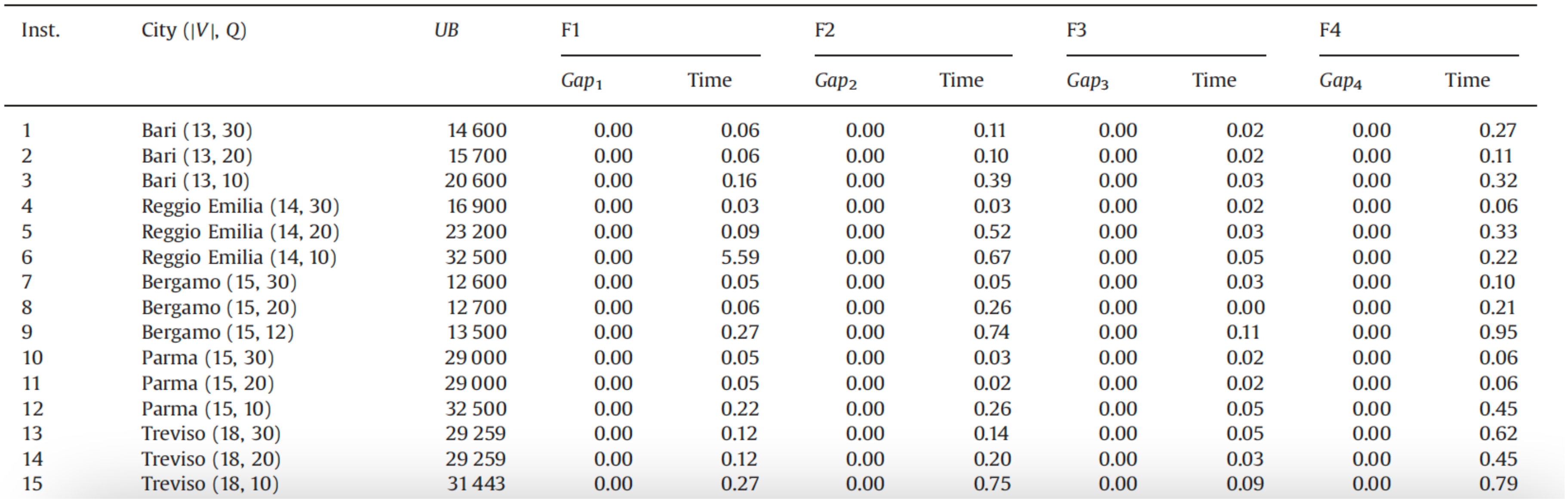
$$\sum_{j\in\tilde{V}_0} g_{j0} \leq \min\{mQ, mQ + Q_{tot}\}$$

$$\max\{0, q_i, -q_j\}x_{ij} \leq f_{ij} \leq \min\{Q, Q + q_i, Q - q_j\}x_{ij}$$
$$\left(i,j\right) \in \tilde{A}$$

$$\left(Q - \min\{Q, Q + q_i, Q - q_j\}\right)x_{ij} \leq g_{ji}$$
$$\leq \left(Q - \max\{0, q_i, -q_j\}\right)x_{ij} \quad \left(i,j\right) \in \tilde{A}$$

Computation results

- *UB*: The value of the best feasible solution found by the B&C algorithm using formulations F1–F4.
- *Time*: Computational time for running the B&C algorithm, in CPU seconds.



Inst.	City ($\lvert V\rvert$, Q)	*UB*	F1		F2		F3		F4	
			Gap_1	Time	Gap_2	Time	Gap_3	Time	Gap_4	Time
1	Bari (13, 30)	14 600	0.00	0.06	0.00	0.11	0.00	0.02	0.00	0.27
2	Bari (13, 20)	15 700	0.00	0.06	0.00	0.10	0.00	0.02	0.00	0.11
3	Bari (13, 10)	20 600	0.00	0.16	0.00	0.39	0.00	0.03	0.00	0.32
4	Reggio Emilia (14, 30)	16 900	0.00	0.03	0.00	0.03	0.00	0.02	0.00	0.06
5	Reggio Emilia (14, 20)	23 200	0.00	0.09	0.00	0.52	0.00	0.03	0.00	0.33
6	Reggio Emilia (14, 10)	32 500	0.00	5.59	0.00	0.67	0.00	0.05	0.00	0.22
7	Bergamo (15, 30)	12 600	0.00	0.05	0.00	0.05	0.00	0.03	0.00	0.10
8	Bergamo (15, 20)	12 700	0.00	0.06	0.00	0.26	0.00	0.00	0.00	0.21
9	Bergamo (15, 12)	13 500	0.00	0.27	0.00	0.74	0.00	0.11	0.00	0.95
10	Parma (15, 30)	29 000	0.00	0.05	0.00	0.03	0.00	0.02	0.00	0.06
11	Parma (15, 20)	29 000	0.00	0.05	0.00	0.02	0.00	0.02	0.00	0.06
12	Parma (15, 10)	32 500	0.00	0.22	0.00	0.26	0.00	0.05	0.00	0.45
13	Treviso (18, 30)	29 259	0.00	0.12	0.00	0.14	0.00	0.05	0.00	0.62
14	Treviso (18, 20)	29 259	0.00	0.12	0.00	0.20	0.00	0.03	0.00	0.45
15	Treviso (18, 10)	31 443	0.00	0.27	0.00	0.75	0.00	0.09	0.00	0.79

Instruction for final project option 2

- Solve the rebalancing problem for Minhang campus.
- Methodology is flexible but must address the following issues:

1. Identify key nodes of significant demands, either positive or negative, over a day.
2. Generate the network, with realistic link travel times, for BRP.
3. Estimate demands at each node.
4. Determine the actual location of the depot (if any).
5. Find optimal rebalancing plan using any tool.
6. Identify hypothetical/alternative depot locations and find the optimal one.

Outline

- Background
- Formulation
- [Not required] Branch & bound algorithm

[Ref]Mauro, Dell'Amico, Eleni, et al. The bike sharing rebalancing problem: Mathematical formulations and benchmark instances[J]. Omega, 2014.

Integer programming

How to solve IPs?

- Discrete variables do not have derivatives – algorithms need to select from solution space
  - 2 binary variables – 4 permutations
  - 3 – 8 permutations
  - 10 – 1024 permutations
  - 20 – 1,048,576 permutations!

- General “exact” algorithm for integer programming, 2 common methods:
  - Branch and bound method (we’ll cover this one)
  - Cutting plane method (the “Gomory cut”)

Integer programming

- Tools

Software	Link	Notes
AMPL	https://ampl.com/	Temporary versions available for students
CPLEX	https://www.ibm.com/products/ilog-cplex-optimization-studio	Free version available to academics through “Academic Initiative”
GAMS	https://www.gams.com/	
Google OR-Tools	https://developers.google.com/optimization/	This is an interface to existing solvers
Gurobi	http://www.gurobi.com/	Free version available to academics through Academic License
Julia	https://julialang.org/	Open-source
LINGO	https://www.lindo.com/index.php/products/lingo-and-optimization-modeling	Trial version available
MATLAB	https://www.mathworks.com/products/matlab.html	
NumPy	http://www.numpy.org/	Python is open-source
R	https://cran.r-project.org/web/views/Optimization.html	R is open-source

Integer programming

Example:

$$\max Z = 5x_1 + 8x_2$$
$$\text{s.t.}$$
$$x_1 + x_2 \le 6$$
$$5x_1 + 9x_2 \le 45$$
$$x_j \in \mathbb{Z}^+, j = 1,2$$

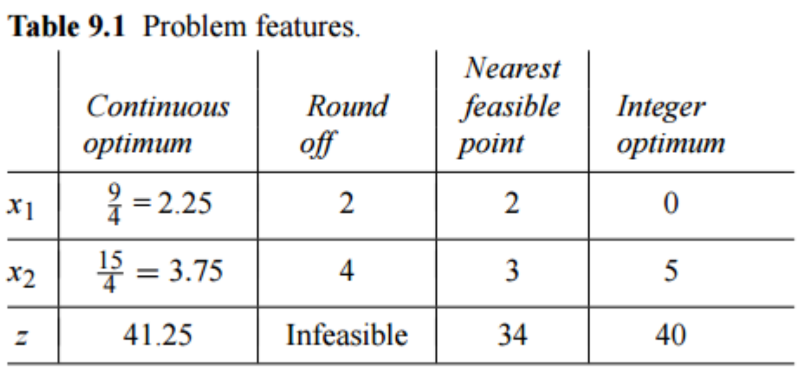


Table 9.1 Problem features.

	Continuous optimum	*Round off*	*Nearest feasible point*	*Integer optimum*
x_1	$\frac{9}{4} = 2.25$	2	2	0
x_2	$\frac{15}{4} = 3.75$	4	3	5
z	41.25	Infeasible	34	40

- Does rounding work?

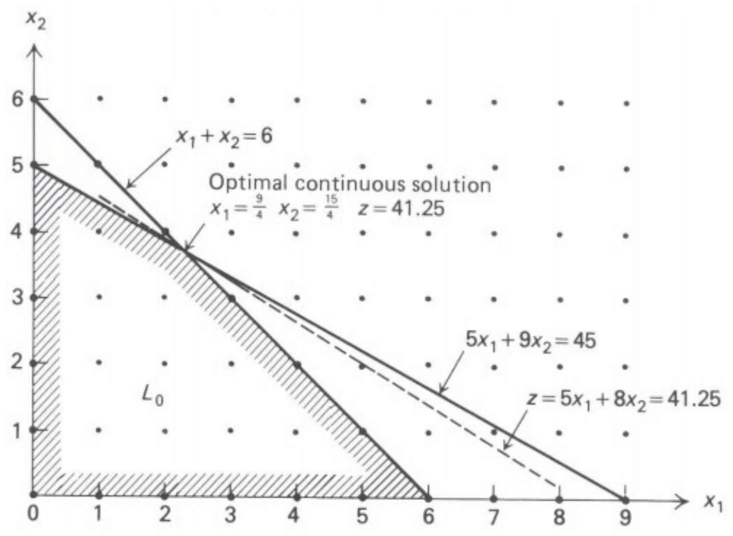


Figure 9.8 An integer programming example.

Branch and bound algorithm

- LP is “easy to solve”
- Fact: A linear IP is simply an LP with extra constraints (per Gomory)
- If we “relax” some constraints, “best case” bound can be obtained
  - Bounds are used to eliminate bad guesses

Branch and bound algorithm

- "Divide and conquer"
- Relax integer constraints and solve the associated LP
- If solution is not integer, take the continuous variable solutions and use those to branch two new constraints
- For each branch, update upper (for max) bound

$$\max Z = 5x_1 + 8x_2$$

s.t.

$$x_1 + x_2 \le 6$$
$$5x_1 + 9x_2 \le 45$$
$$\cancel{x_j \in \mathbb{Z}^+, j = 1,2}$$
$$x_j \ge 0, j = 1,2$$

This is called the "associated LP" of the IP

Branch and bound algorithm

Illustration:

- Solve initial associated LP (L_0) gets $x_1 = 2.25, x_2 = 3.75, z = 41.25$
- This solution tells us that the optimum has $z^* \leq \bar{z} = 41.25$
- To branch, we pick one of the continuous values (x_2), and create two branches by adding constraint to L_0: $L_2: x_2 \leq 3$, $L_1: x_2 \geq 4$
- Each branched region $L_1$and L_2 would have $z^* \leq 41.25$

$$\max Z = 5x_1 + 8x_2$$

s.t.

$$x_1 + x_2 \leq 6$$
$$5x_1 + 9x_2 \leq 45$$
$$x_j \geq 0, j = 1{,}2$$

$$\begin{tikzcd} & L_0 \arrow[dl] \arrow[dr] & \\ L_1 & & L_2 \end{tikzcd}$$

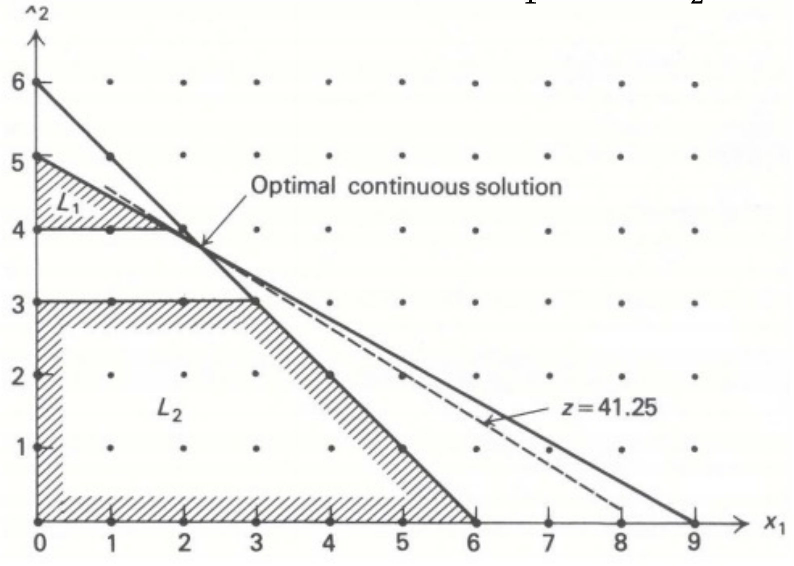
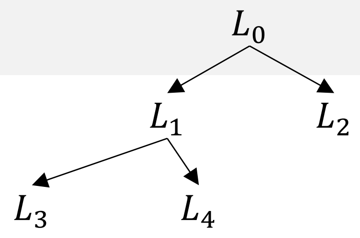


Figure 9.9 Subdividing the feasible region.

Branch and bound algorithm



- What is L_1?

$$\max Z = 5x_1 + 8x_2$$
s.t.
$$x_1 + x_2 \leq 6$$
$$5x_1 + 9x_2 \leq 45$$
$$x_2 \geq 4$$
$$x_j \geq 0, j = 1,2$$

We pick L_1 to solve, and find $x_1 = 1.8, x_2 = 4, z = 41$

We branch L_1 by $x_1 \leq 1$ (L_4) and $x_1 \geq 2$ (L_3)

- What is L_2?

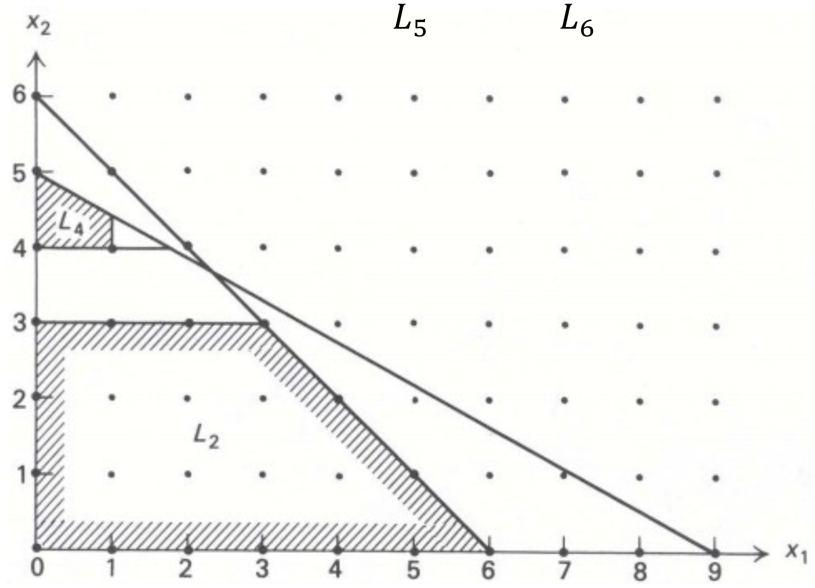
$$\max Z = 5x_1 + 8x_2$$
s.t.
$$x_1 + x_2 \leq 6$$
$$5x_1 + 9x_2 \leq 45$$
$$x_2 \leq 3$$
$$x_j \geq 0, j = 1,2$$

Branch and bound algorithm

$$\begin{tikzcd}
 & & L_0 \arrow[dl] \arrow[dr] & \\
 & L_1 \arrow[dl] \arrow[dr] & & L_2 \\
\cancel{L_3} & & L_4 \arrow[dl] \arrow[dr] & \\
 & L_5 & & L_6
\end{tikzcd}$$

- L_3 is not feasible, while L_4 is:

$$\max Z = 5x_1 + 8x_2$$
s.t.
$$x_1 + x_2 \leq 6$$
$$5x_1 + 9x_2 \leq 45$$
$$x_2 \geq 4$$
$$x_1 \leq 1$$
$$x_j \geq 0, j = 1,2$$



Solving this we get $x_1 = 1, x_2 = 4.444$, so let's say we branch on $x_2 \leq 4$ (L_5) and $x_2 \geq 5$ (L_6)

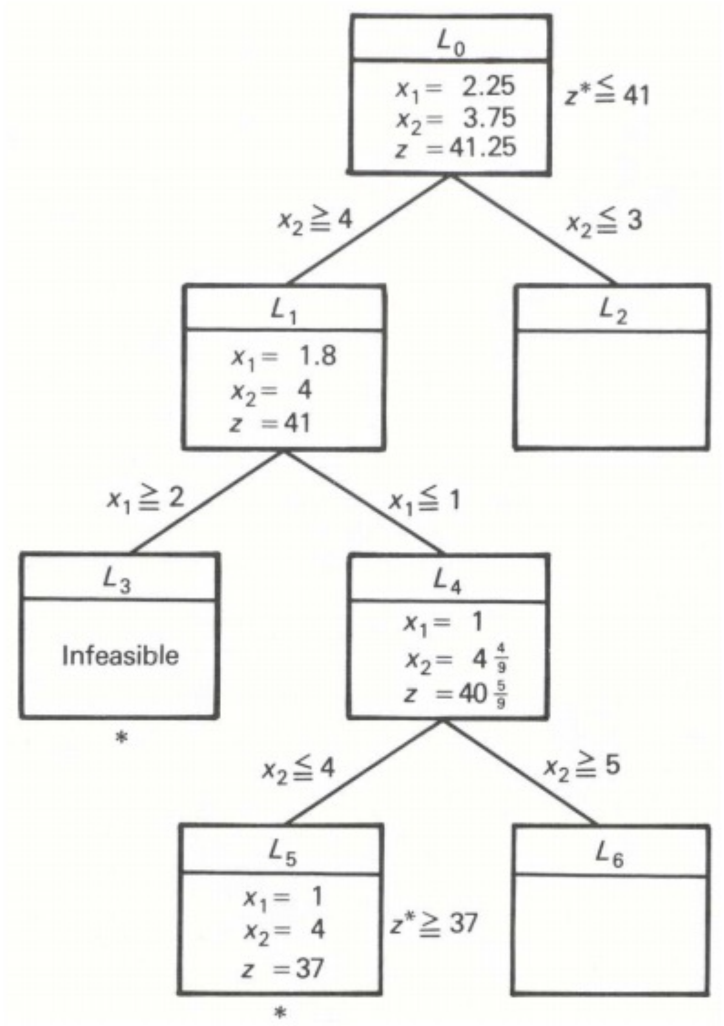
For L_5, we get $x_1 = 1, x_2 = 4, z = 37$. Since it's integer, we stop that branch, and set lower bound to $z^* \geq 37$. If we stop and keep this solution, we have a duality gap of (41-37)/37=10.8%

Branch and bound algorithm

Current status

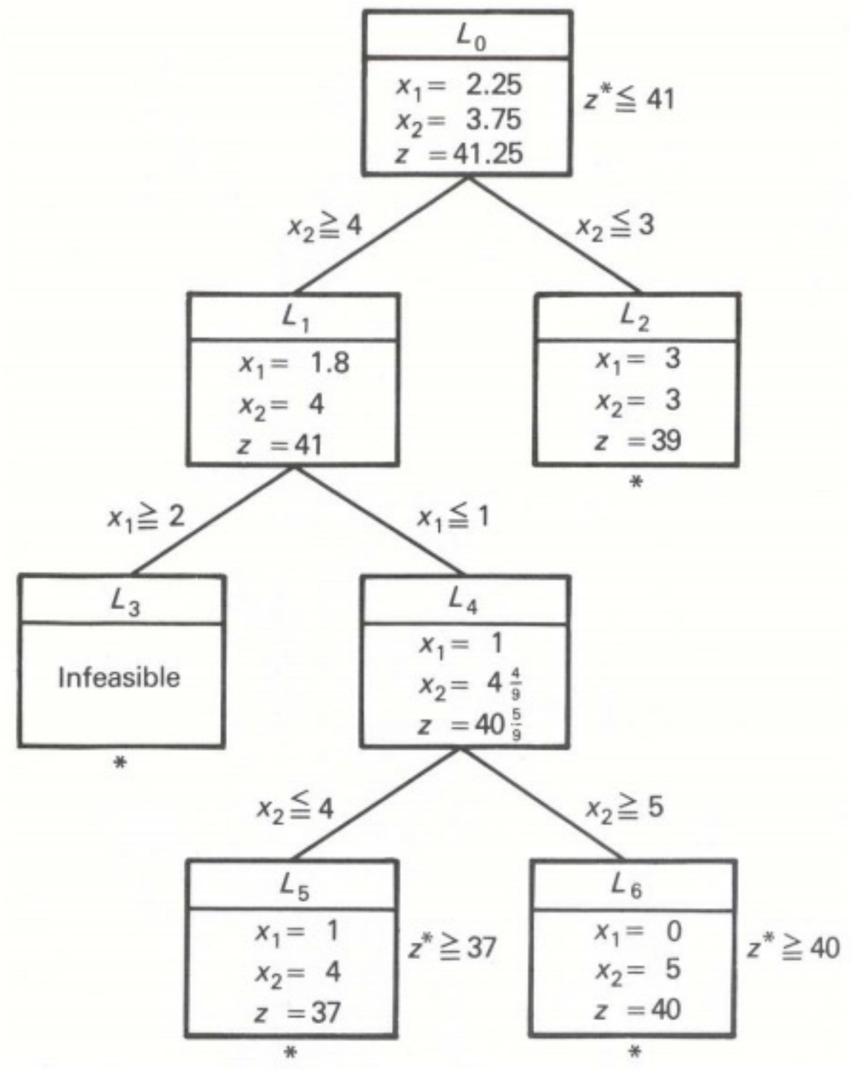
- We still have to close (we call closing "fathoming") L2 and L6.

- With L6, we get $x_1 = 0, x_2 = 5, z = 40$. Since this is higher than the current lower bound of 37, we update: $z^* \geq 40$. Stopping now would have at most 2.5% duality gap.



Branch and bound algorithm

- Now we tackle L2 and get: $x_1 = x_2 = 3, z = 39$
- Regardless of whether the answer is all integer, since $z < z^*$, we can fathom this branch and keep the solution at L_6



Summary of Branch and Bound algorithm

For maximization problem, just reverse for minimization problem

1. Initialize. Set lower bound $z^* = -\infty$ and upper bound $\bar{z}$ from associated LP. Apply steps 2-4 to whole problem. If fathomed, stop.
2. Branching. Among unfathomed subproblems, select one most recently created and create 2 branches by inserting constraints to the parent associated LP: $x_j \le \lfloor x_j^* \rfloor$ for one and $x_j \ge \lfloor x_j^* \rfloor + 1$ for other
3. Bounding. For each new subproblem, solve associated *LP*. Update $\bar{z}$ if appropriate.
4. Fathom. For each subproblem, apply 3 fathom tests:

   Test 1. If new upper bound $\le z^*$

   Test 2. If no feasible solution.

   Test 3. If solution is a feasible integer problem. In this case, if the bound $> Z^*$, update Z^*.
5. Optimality test. Stop when there are no remaining subproblems (in which case z^* is exact solution), or if duality gap $(\bar{z} - z^*)$ is within tolerance.