Examples for Walmart Interview

Author: Daniel A. Zuniga Vazquez

Written in **LaTeX** and coded as specified in:
- **C++** and **C#** using Visual Studio
- **Java** using Eclipse
- **Python** using Anaconda with Jupyter Notebook

Contents

1 Vehicle Routing Problem (VRP) 2
1.1 Two Index Vehicle Routing Formulation . . . 2

2 Facility Location 3
2.1 Capacitated Facility Location . . . 3
2.2 Uncapacitated Facility Location . . . 4

3 Depth First Search (DFS) 5
3.1 Graph Depth First Search . . . 5
3.2 Mother Vertices from Directed Graph . . . 5
3.3 Longest Path in Graph . . . 5
3.4 Detect Cycle Under Directed Graph . . . 5
3.5 Print All Paths from a Source and Destination . . . 5
3.6 Transitive Closure of Adjacency Matrix . . . 5
3.7 Island Number in 2D Binary Matrix . . . 5

4 Decomposition Methods and Branch & Bound 6

5 Sorting Problems 7
5.1 IP Sorting Formulation . . . 7
5.2 Sorting Algorithms . . . 8
5.2.1 Quick sort algorithm . . . 8
5.2.2 Merge sort algorithm . . . 8
5.2.3 Bubble sort algorithm (recursive) . . . 8
5.2.4 Selection sort algorithm . . . 8
5.2.5 Insertion sort algorithm . . . 8

6 Metaheuristics 9
6.1 Simulated Annealing . . . 9
6.2 Tabu Search . . . 9
6.3 Particle Swarm Optimization . . . 10
6.4 External Libraries . . . 11
6.4.1 OR-Tools - Tabu Search and Simulated Annealing . . . 11

7 Dynamic Programming and Other Examples 12
7.1 0-1 Knapsack Problem . . . 12
7.2 0-1 Knapsack Problem - Dynamic Programming . . . 13
7.3 Rope Cutting - Dynamic Programming . . . 14
7.4 Coin Change - Dynamic Programming . . . 15
7.5 Shortest Path - Dijkstra's - Dynamic Programming . . . 16
7.6 Minimum Dominating Set . . . 18
7.7 Optimum Assignment Problem . . . 19

8 SQL 20

1 Vehicle Routing Problem (VRP)

1.1 Two Index Vehicle Routing Formulation

The following vehicle routing example is coded in **C++**, **C#**, **Python** and **Java**:

Sets and indices

- D: Set of destinations, indexed by i, j.

Parameters

- c_{ij}: Vehicle routing cost from vertex i to vertex j.
- K: Number of vehicles.
- MAX: Maximum number of destinations a vehicle can be routed to.

Variables

- x_{ij}: Binary variables that is 1 if a vehicle is routed from destination i to destination j and 0 otherwise.
- y_i: Integer variable that denotes the destination i position in the vehicle routing.

The vehicle routing formulation is presented as follows:

$$\min_{x,y} \sum_{i \in D} \sum_{j \in D} c_{ij} x_{ij} \qquad \text{(Minimize total cost)}$$

$$\text{s.t.} \sum_{i \in D} x_{ij} = 1, \ \forall j \in D \setminus \{0\} : j \neq i \qquad \text{(1.1a - Only one vehicle can enter a destination)}$$

$$\sum_{j \in D} x_{ij} = 1, \ \forall i \in D \setminus \{0\} : i \neq j \qquad \text{(1.1b - Only one vehicle can leave a destination)}$$

$$\sum_{i \in D \setminus \{0\}} x_{i0} = K \qquad \text{(1.1c - Number of vehicles leaving the depot at destination 0)}$$

$$\sum_{j \in D \setminus \{0\}} x_{0j} = K \qquad \text{(1.1d - Number of vehicles entering the depot at destination 0)}$$

$$y_i - y_j + MAX \ x_{ij} \leq MAX - 1, \ \forall i, j \in D \setminus \{0\} : i \neq j \qquad \text{(1.1e - Subtour elimination (Miller-Tucker-Zemlin))}$$

$$x_{i,j} \in \{0,1\}, y_i \in \mathbb{Z}, \ \forall i \in I, \forall j \in J \qquad \text{(1.1f - Domain)}$$

2 Facility Location

2.1 Capacitated Facility Location

The following facility location example is coded in **C++**, **C#**, **Python** and **Java**:

Sets and indices

- I: Set of facilities, indexed by i.
- J: Set of customers, indexed by j.

Parameters

- c_{ij}: Cost of facility i to supply to customer j.
- d_j: Demand of customer j.
- f_i: Cost of adding facility i.
- u_i: Capacity of facility i.

Variables

- x_i: Binary variables that is 1 if facility i is assigned and 0 otherwise.
- y_{ij}: Fraction of demand supplied from facility i to customer j.

The capacitated facility location formulation is presented as follows:

$$\min_{\boldsymbol{x},\boldsymbol{y}} \sum_{i\in I}\sum_{j\in J} c_{ij}d_j y_{ij} + \sum_{i\in I} f_i x_i \qquad \text{(Minimize total cost)}$$

$$\text{s.t.} \sum_{i\in I} y_{ij} = 1,\ \forall j \in J \qquad \text{(2.1a - Satisfied fraction of demand)}$$

$$\sum_{j\in J} d_j y_{ij} \le u_i x_i,\ \forall i \in I \qquad \text{(2.1b - Facility capacity)}$$

$$y_{ij} \le 0, x_i \in \{0,1\},\ \forall i \in I, \forall j \in J \qquad \text{(2.1c - Domain)}$$

2.2 Uncapacitated Facility Location

The following facility location example is coded in **C++**, **C#**, **Python** and **Java**:

Sets and indices

- I: Set of facilities, indexed by i.
- J: Set of customers, indexed by j.

Parameters

- c_{ij}: Cost of facility i to supply to customer j.
- d_j: Demand of customer j.
- f_i: Cost of adding facility i.
- M: Big M, i.e., sufficiently big number.

Variables

- x_i: Binary variables that is 1 if facility i is assigned and 0 otherwise.
- y_{ij}: Fraction of demand supplied from facility i to customer j.

The uncapacitated facility location formulation is presented as follows:

$$\min_{\boldsymbol{x},\boldsymbol{y}} \sum_{i\in I}\sum_{j\in J} c_{ij}d_j y_{ij} + \sum_{i\in I} f_i x_i \qquad \text{(Minimize total cost)}$$

$$\text{s.t.} \sum_{i\in I} y_{ij} = 1,\ \forall j \in J \qquad \text{(2.2a - Satisfied fraction of demand)}$$

$$\sum_{j\in J} d_j y_{ij} \le M x_i,\ \forall i \in I \qquad \text{(2.2b - Uncapacitated facility)}$$

$$y_{ij} \le 0, x_i \in \{0,1\},\ \forall i \in I, \forall j \in J \qquad \text{(2.2c - Domain)}$$

3 Depth First Search (DFS)

The following DFS examples are coded in **C++**.

3.1 Graph Depth First Search

Given a directed graph and a source, identify the DFS.

3.2 Mother Vertices from Directed Graph

Prints mother vertices of a directed graph, i.e., the nodes from which all other nodes can be reached.

3.3 Longest Path in Graph

Identifies the longest path size per node and prints the longest one of the graph.

3.4 Detect Cycle Under Directed Graph

Given a directed graph, identifies if there is a cycle.

3.5 Print All Paths from a Source and Destination

Given a source, a destination and a directed graph, print all their paths.

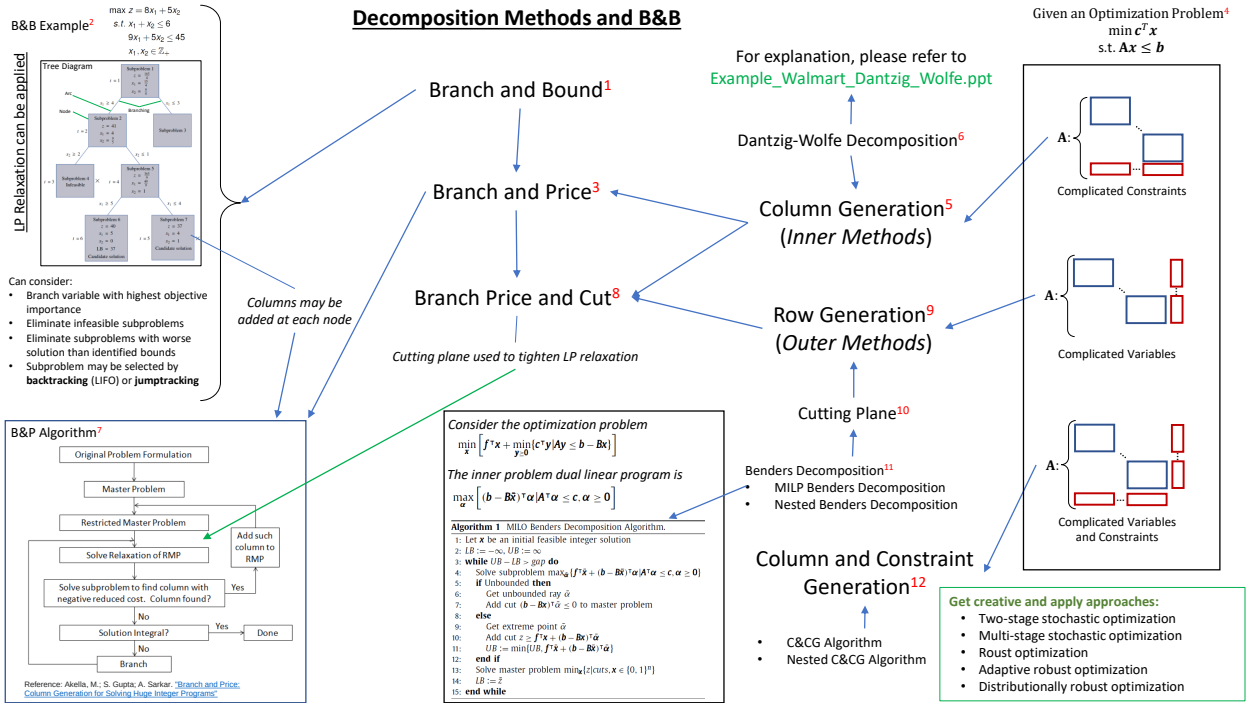
3.6 Transitive Closure of Adjacency Matrix

For a specified adjacency matrix (i.e., a square matrix that represents a graph), the transitive closure is identified, i.e., given a vertex v, identify if it is reachable from another vertex u given every link or edge (v,u).

3.7 Island Number in 2D Binary Matrix

Given a 2D binary matrix, identifies the number of islands.

4 Decomposition Methods and Branch & Bound



5 Sorting Problems

5.1 IP Sorting Formulation

IP formulation. Therefore, given an array C with elements $c_i, i = 1, \ldots, |C|$, consider the following:

Sets and indices

- I: Set of numbers to be sorted, indexed by i, j.

Parameters

- c_j: Value of the j number, i.e., sorting parameter.

Variables

- x_{ij}: Binary variables that is 1 if i value is assigned j place and 0 otherwise.

The IP sorting formulation is presented as follows:

$$\max_{\boldsymbol{x}} \sum_{i \in I} \sum_{i \in J} x_{ij} \qquad \text{(Maximize the sum of } x_{ij} \text{ will determine the sorted array)}$$

$$\text{s.t.} \sum_{j \in I} x_{ij} = 1, \ \forall i, \in I \qquad \text{(Sorted array elements can only be equal to one of original array)}$$

$$\sum_{i \in I} x_{ji} = 1, \ \forall j, \in I \qquad \text{(Original array elements can only be equal to one of the sorted array)}$$

$$\sum_{j \in I} c_j x_{ij} \leq \sum_{j \in I} c_j x_{1+1,j}, \ \forall i, \in I \qquad \text{(Sorting restriction)}$$

$$x_{ij} \in \{0, 1\}, \forall i, j \in I \qquad \text{(Domain)}$$

Reference:
- Coded by - Daniel Zuniga

5.2 Sorting Algorithms

The following sorting algorithm examples are coded in **C++**.
Reference:
- Coded by - Daniel Zuniga

5.2.1 Quick sort algorithm

Recursively divides the array into two subarrays and assign values depending if each value is greater or smaller than a selected pivot (first or last value of the array cell that is divided).

Performance:

- Best $O(nlogn)$ - Worst $O(n^2)$, depends on the pivot size.
- Auxiliary space $O(n)$.

5.2.2 Merge sort algorithm

Iteratively divides the array in two subarrays until single value arrays are defined and merges them in a sorted order.

Performance:

- Worst $O(nlogn)$.
- Auxiliary space $O(n)$.

5.2.3 Bubble sort algorithm (recursive)

In each iteration, each pair of the array is compared and swapped if it is in the wrong order.

Performance:

- Best $O(n)$ - Worst $O(n^2)$, depending on the initial arrangement of the values (worst case for reversed sorted).
- Auxiliary space $O(1)$ - $O(n)$ for recursive version.

5.2.4 Selection sort algorithm

Divides the array into two subsets (sorted and unsorted). Initial sorted subset is empty. In each iteration, the smallest number is identified and assigned to the sorted subset.

Performance:

- $O(n^2)$
- Auxiliary space $O(n)$ for recursive version.

5.2.5 Insertion sort algorithm

Divides the array into two subsets (sorted and unsorted), where the first number ($i = 0$) of the array assign to the sorted and the rest to the unsorted. Each iteration assigns the $i+1$ value of the array from the unsorted into the sorted subset in its sorted location.

Performance:

- Best $O(n)$ - Worst $O(n^2)$, depending on the initial arrangement of the values (worst case for reversed sorted).
- Auxiliary space $O(1)$ - $O(n)$ for the recursive version.

6 Metaheuristics

6.1 Simulated Annealing

Simulated Annealing is coded in **Java** based on the following pseudocode and applied to the Traveling Salesman Problem (TSP) as an illustrative example.

Algorithm 1: Simulated Annealing pseudocode

Result: Return x as the best solution
$h(.)$ is the function to optimize;
Generate an initial Temperature $T > 0$;
while $T >$ *desired* T **do**
    Sample from a symmetrical distribution, e.g., uniform or normal;
    Add noise to create new candidate solution $x' = x + noise$;
    Calculate probability $p = exp(\Delta h/T_i)$, where Δh is the difference between solutions from x and x';
    Accept of reject candidate solution with probability p; $u \sim U(0,1)$ accept $x = x'$ if $u \leq p$;
    Update temperature (cooling), e.g., $T = \alpha T$, where $0 < \alpha < 1$
end

Reference:
- Coded by - Daniel Zuniga

6.2 Tabu Search

Tabu Search is coded in **Java** based on the following pseudocode and applied to the Traveling Salesman Problem (TSP) as an illustrative example.

Algorithm 2: Tabu Search pseudocode

Result: Return x_o as the best solution
Set tabu list to null, set x_o;
for *No. Iterations* **do**
    $\{x_1, x_2, \ldots, x_n\}$ = Generate neighborhood (x_o);
    for $i = 1, i <= n$ **do**
        if x_i *not in tabu list and solution* $x_i < x_o$ **then**
            replace $x_o = x_i$ and;
            update tabu list:;
            - add i to tabu list;
            - decrease tabu list;
        end
    end
end

Reference:
- Coded by - Daniel Zuniga

6.3 Particle Swarm Optimization

Particle Swarm Optimization is coded in **Java** based on the following pseudocode and applied to the Traveling Salesman Problem (TSP) as an illustrative example.

Algorithm 3: Particle Swarm Optimization pseudocode

```
Result: Return x_o as the best solution
Initialize particle list, locations, velocities and solutions x_oi;
Set tabu list to null, set x;
for No. iterations do
    for i = 1; i <= No. particles do
        Determine neighbors of particle i;
        for j = 1 ; j <= No. neighbor iterations do
            Determine solution x_ij ;
            if x_ij < x_oi then
                x_oi = x_ij;
            end
        end
        Sort(x_oi);
        if x_o1 < x_o then
            Update best solution;
            x_o = x_o1;
        end
    end
    for i = 1; i <= No. particles do
        for k = 1; k <= No. locations do
            Find particle with best solution x_oi;
            Select a random vector rand for particle velocity ;
            Update velocity v_ik constriction;
            Update location l_ik constriction;
        end
    end
end
```

Reference:
- Coded by - Daniel Zuniga

6.4 External Libraries

6.4.1 OR-Tools - Tabu Search and Simulated Annealing

A Vehicle Routing Problem (VRP) is presented using **OR-Tools** and considering two metaheuristic searches are selected to escape local optimum, Tabu Search and Simulated Annealing. The VRP is coded in **C++**.

Problem Formulation. The goal is to find optimal routes for multiple vehicles visiting a set of destinations, i.e., minimize the total distance from all vehicles.

- Capacity constraints: vehicles have a maximum distance that can travel.
- Solution limit: iterations before search stops.
- Time limit: time before stopping the search

Numerical Experiments. For the numerical experiment, the parameters are determined as follows:

- Distance matrix between all destinations:

```
{ {0,   658,  931,  835,  698,  329,  602,  233,  370,  233,  643,  602,  466,  425,  562,  931,  794 },
{658,   0,    821,  370,  233,  602,  876,  425,  835,  890,  1301, 713,  576,  809,  1219, 1042, 1452},
{931,   821,  0,    1190, 1054, 602,  329,  972,  562,  890,  480,  1534, 1397, 1356, 946,  1862, 905 },
{835,   370,  1190, 0,    137,  780,  1054, 602,  1013, 1068, 1478, 617,  754,  986,  1397, 672,  1630},
{698,   233,  1054, 137,  0,    643,  917,  466,  876,  931,  1342, 480,  617,  850,  1260, 809,  1493},
{329,   602,  602,  780,  643,  0,    274,  370,  233,  288,  698,  931,  794,  754,  617,  1260, 850 },
{602,   876,  329,  1054, 917,  274,  0,    643,  233,  562,  425,  1205, 1068, 1027, 617,  1534, 576 },
{233,   425,  972,  602,  466,  370,  643,  0,    410,  466,  876,  562,  425,  384,  794,  890,  1027},
{370,   835,  562,  1013, 876,  233,  233,  410,  0,    329,  466,  972,  835,  794,  384,  1301, 617 },
{233,   890,  890,  1068, 931,  288,  562,  466,  329,  0,    410,  643,  506,  466,  329,  972,  562 },
{643,   1301, 480,  1478, 1342, 698,  425,  876,  466,  410,  0,    1054, 917,  876,  466,  1382, 425 },
{602,   713,  1534, 617,  480,  931,  1205, 562,  972,  643,  1054, 0,    137,  370,  780,  329,  1013},
{466,   576,  1397, 754,  617,  794,  1068, 425,  835,  506,  917,  137,  0,    233,  643,  466,  876 },
{425,   809,  1356, 986,  850,  754,  1027, 384,  794,  466,  876,  370,  233,  0,    410,  506,  643 },
{562,   1219, 946,  1397, 1260, 617,  617,  794,  384,  329,  466,  780,  643,  410,  0,    917,  233 },
{931,   1042, 1862, 672,  809,  1260, 1534, 890,  1301, 972,  1382, 329,  466,  506,  917,  0,    958 },
{794,   1452, 905,  1630, 1493, 850,  576,  1027, 617,  562,  425,  1013, 876,  643,  233,  958,  0 },}
```

- Vehicle available: 4
- Maximum distance per vehicle: 3,000
- Solution limit: 30
- Time limit: 150 seconds

Reference:
- Ortools, available at: https://developers.google.com/optimization
- Coded by - Daniel Zuniga

7 Dynamic Programming and Other Examples

7.1 0-1 Knapsack Problem

The following example is coded in **C++**.

There are different kind of items with a weight and value associated and a weight limitation to the bag. The objective is to maximize the value of the objects in the bag.

This model has an exponential running time $O(2^n)$. Please refer to 0-1 Knapsack Problem - Dynamic Programming 5 to increase the computational performance.

Sets and indices

- $\mathscr{I}$: Set of items,indexed by s.

Parameters

- v_i: Value of product i. ($)
- w_i: Weight of product i. (kg)
- W: Weight limit of the bag. (kg)

Variables

- x_i: Binary variables that is 1 if product i is selected for the bag and 0 otherwise.

$$\max_{\boldsymbol{x}} \sum_{i \in \mathscr{I}} v_i x_i \tag{5a}$$

$$\text{s.t.} \sum_{i \in \mathscr{I}} w_i x_i \leq W \tag{5b}$$

$$x_i \in \{0,1\},\ \forall i \tag{5c}$$

Objective (5a). Ensures that the weight does not exceed the bag limit (5b); domains (5c).

Numerical experiments with the a weight limit $W = 50$ and the following products, weights, and values:

	Product 1	Product 2	Product 3	Product 4	Product 5
Weight w_i	12	32	33	5	34
Value v_i	100	200	50	60	150

Reference:
- Model - CodesDope, Knapsack Problem, available at: https://www.codesdope.com/course/algorithms-knapsack-problem/
- Coded by - Daniel Zuniga

7.2 0-1 Knapsack Problem - Dynamic Programming

The following example is coded in **C++**.

There are different kind of items with a weight and value associated and a weight limitation to the bag. The objective is to maximize the value of the objects in the bag.

Sets and indices

- $\mathscr{I}$: Set of items,indexed by s.
- $\mathscr{W}$: Set of remaining weights available,indexed by w, $W = |\mathscr{W}|$.

Parameters

- v_i: Value of product i. ($)
- w_i: Weight of product i. (kg)
- W: Weight limit of the bag. (kg)

Variable

- $CostArray_{i,w}$: Matrix to store the solution of the function $F(i,w)$.

Given a function $F(i,w)$ that optimizes the value for the first i products and counter index w for weight limit W:

$$F(i,w) = \begin{cases} F(i-1,w), \text{ if } w_i > w \\ \max\{F(i-1,w), (F(i-1,w-w_i)+v_i)\}, \text{ if } w_i \leq w \end{cases} \tag{6a}$$

Dynamic programming function for 0-1 Knapsack problem (6a).

Numerical experiments with the a weight limit $W = 5$ and the following products, weights, and values:

	Dummy Product	Product 1	Product 2	Product 3	Product 4
Weight w_i	0	3	2	4	1
Value v_i	0	8	3	9	6

Reference:
- Model - CodesDope, Knapsack Problem, available at: https://www.codesdope.com/course/algorithms-knapsack-problem/
- Coded by - Daniel Zuniga

7.3 Rope Cutting - Dynamic Programming

The following example is coded in **C++**.

Given a rope of size n and a certain number of cuts c of different lengths l_c and sale prices p_c, maximize the revenue.

Sets and indices

- $\mathscr{C}$: Set of rope cuts available, indexed by c.
- $\mathscr{N}$: Set of remaining rope lengths, indexed by n, $N = |\mathscr{N}|$.

Parameters

- l_c: Length of cut c. (m)
- p_c: Sale price of cut c. ($)
- N: Total length of rope i. (m)

Variable

- r_n: Variable to store the revenue solution.

The maximize the income, the dynamic approach will be to sell the first cut p_c with $c = n$ and optimize the revenue for the remaining length r_{N-n} as follows:

$$r_N = \max_{1 \leq n \leq N} \{p_n + r_{N-n}\} \tag{7}$$

Dynamic programming function for rope cutting problem (7).

Numerical experiments with a rope length $n = 5$ and the following length cuts l_c and sale price per cut p_c:

	Dummy Cut	Cut 1	Cut 2	Cut 3	Cut 4	Cut 5
Length l_c (m)	0	1	2	3	4	5
Price p_c ($)	0	10	24	30	40	45

Reference:
- Model - CodesDope, Rod Cutting, available at: https://www.codesdope.com/course/algorithms-rod-cutting/
- Coded by - Daniel Zuniga

7.4 Coin Change - Dynamic Programming

The following example is coded in **C++**.

Given an amount to sum A and a set of coins C with their respective values v_c, minimize the number of coins to be used.

Sets and indices

- $\mathscr{C}$: Set of coin, indexed by c.
- $\mathscr{N}$: Set of remaining sum to be achieved, indexed by n, $N = |\mathscr{N}|$.

Parameters

- v_c: Value of coin c. ($)
- A: Amount to sum with coins. ($)

Variable

- n_c: Variable to store the number of coins c.

Given a function M_n that minimizes the number of coins c to be used:

$$M_n = \begin{cases} \min_{c:v_c \leq A}\{M_{n-v_c} + 1\}, \text{ if } n > 0 \\ 0, \text{ if } n = 0 \end{cases} \tag{8a}$$

Dynamic programming function for coin change problem (8a).

Numerical experiments with an amount to sum of $A = 38$ and the following coins c and coin values v_c:

	Dummy Coin	Coin 1	Coin 2	Coin 3	Coin 4
Coin value v_c ($)	0	1	2	5	10

Reference:
- Model - CodesDope, Coin Change, available at: https://www.codesdope.com/course/algorithms-coin-change/
- Coded by - Daniel Zuniga

7.5 Shortest Path - Dijkstra's - Dynamic Programming

The following example is coded in **C++**.

Given a direct graph $G = (N, E)$ with edge (or arc) length (or length) $d_{ij} \geq 0$ for edge $(i, j) \in E$, identifies the shortest path from the source node (initial) s to the sink node (destination) t.

Sets and indices

- $\mathscr{N}$: Set of nodes (or vertex), indexed by n.
- $\mathscr{E}$: Set of edges (or links, arcs), indexed by e or $\{ij\}$.

Parameters

- d_{ij}: Distance from node i to node j. (m)
- s: Source node (initial)
- t: Sink node (destination)

Variable

- x_{ij}: Binary variable that is 1 is edge between node i and j is selected.

$$\max_{\boldsymbol{x}} \sum_{i,j \in \mathscr{E}} d_{ij} x_{ij} \tag{9a}$$

$$\text{s.t.} \sum_{j:(i,j)\in E} x)ij - \sum_{j:(j,i)\in E} x_{ji} = \begin{cases} 1, & \text{if } i = s \\ 0, & \forall i \in E \setminus \{s,t\} \\ -1, & \text{if } i = t \end{cases} \tag{9b}$$

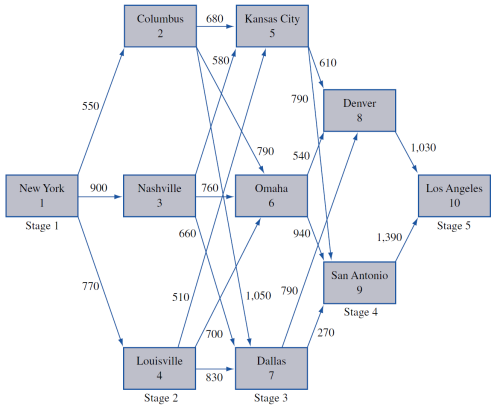
$$x_{ij} \in \{0,1\},\ \forall (i,j) \in E \tag{9c}$$

Shortest path problem general formulation (9); objective (9a); constraint that sums the edges of all paths (9b); domains (9c).

Numerical experiments with ten nodes, $N = 10$, twenty edges, $E = 20$, five decision making stages $S = 5$ (initial node $s = 1$, destination $s = 5$), and the following distance matrix c_{ij}:

		j									
	c_{ij}	1	2	3	4	5	6	7	8	9	10
i	1	0	550	900	770	0	0	0	0	0	0
	2	550	0	0	0	680	790	1050	0	0	0
	3	900	0	0	0	580	760	700	0	0	0
	4	770	0	0	0	510	700	830	0	0	0
	5	0	680	580	510	0	0	0	610	790	0
	6	0	790	760	700	0	0	0	540	940	0
	7	0	1050	700	830	0	0	0	790	270	0
	8	0	0	0	0	610	540	790	0	0	1030
	9	0	0	0	0	790	940	270	0	0	1390
	10	0	0	0	0	0	0	0	1030	1390	0

or



Reference:
- Coded by - Daniel Zuniga

7.6 Minimum Dominating Set

The following example is coded in **C++**.

Minimum dominating set (MDS) problem: given an (undirected) graph $G = (V, E)$ with vertex set V and edge set E, where V has n vertices, and E is expressed by adjacency matrix $A = (a_{ij})_{n*n}$ such that $a_{ij} = 1$ if there is an edge between i and $j (i \neq j)$, and $a_{ii} = 0$. The problem is to find a subset $D \subseteq V$ with smallest cardinality such that every vertex of V is either chosen into the subset D or has at least one neighbor in D.

$x_i \in \{0, 1\}$: a binary variable indicates that whether vertex i is chosen into D if $x_i = 1$ or not if $x_i = 0$

$$\min_{\boldsymbol{x}} \sum_{i=1}^{n} x_i \tag{10a}$$

$$\text{s.t. } x_i + \sum_{j=1, j\neq i}^{n} a_{ij} x_j \geq 1, \ \forall i \in V \tag{10b}$$

$$x_i \in \{0, 1\}, \forall i \in V \tag{10c}$$

Objective (10a). Minimum dominating set constraint (10b); domains (10c).

Numerical experiments with $V = 14$ vertex or nodes and $E = 20$ edges or vertices.

Edge No	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
From Node	2	5	3	4	5	4	5	7	9	6	11	12	13	8	9	10	14	11	14	14
To Node	1	1	2	2	2	3	4	4	4	5	6	6	6	7	7	9	9	10	12	13

Reference:
- Coded by - Daniel Zuniga

7.7 Optimum Assignment Problem

The following example is coded in **C++**.

Identify the job assignment that will maximize the production.
Sets and indices

- $\mathscr{I}$: Set of production factories,indexed by s.
- $\mathscr{J}$: Set of products, indexed by j.

Parameters

- a_{ij}: Production rate of product j at factory i. (products/time)

Variables

- x_{ij}: Binary variables that is 1 if factory i is assigned product j and 0 otherwise.

$$\max_{\boldsymbol{x}} \sum_{i \in \mathscr{I}} \sum_{j \in \mathscr{J}} a_{ij} x_{ij} \tag{11a}$$

$$\text{s.t.} \sum_{j \in \mathscr{J}} x_{ij} = 1, \ \forall i \tag{11b}$$

$$\sum_{i \in \mathscr{I}} x_{ij} = 1, \ \forall j \tag{11c}$$

$$x_{ij} \in \{0,1\}, \ \forall i, \forall j \tag{11d}$$

Objective (11a). Constraints are as follows: each factory i can only produce one product j (11b); each product j can only be assigned to one factory i (11c); domains (11d).

Numerical experiments with the following factories, products, and production rates:

Factory	Product 1	Product 2	Product 3	Product 4	Product 5
Factory 1	12	6	8	7	8
Factory 2	6	8	7	10	3
Factory 3	8	11	12	5	9
Factory 4	9	12	6	11	15
Factory 5	10	7	12	9	7

Reference:
- Coded by - Daniel Zuniga

8 SQL

A SQL example is presented using for **Particle Swarm Optimization**, coded in **Java**, and using **MySQL**.