

Optimierung und Entscheidungsunterstützung (OuE)

Anna Prenzel

17.05.2013

Hochschule Zittau/Görlitz

Topics for today:

- ① JSSP as disjunctive graph
- ② Network Flow Programming
- ③ Dynamic Programming
 - Dijkstra Shortest Path Algorithm
 - Equipment Replacement Problem
 - Traveling Salesman Problem

JSSP - an instance

+++++

Adams, Balas, and Zawack 10x10 instance

10 10

4	88	8	68	6	94	5	99	1	67	2	89	9	77	7	99	0	86	3	92
5	72	3	50	6	69	4	75	2	94	8	66	0	92	1	82	7	94	9	63
9	83	8	61	0	83	1	65	6	64	5	85	7	78	4	85	2	55	3	77
7	94	2	68	1	61	4	99	3	54	6	75	5	66	0	76	9	63	8	67
3	69	4	88	9	82	8	95	0	99	2	67	6	95	5	68	7	67	1	86
1	99	4	81	5	64	6	66	8	80	2	80	7	69	9	62	3	79	0	88
7	50	1	86	4	97	3	96	0	95	8	97	2	66	5	99	6	52	9	71
4	98	6	73	3	82	2	51	1	71	5	94	7	85	0	62	8	95	9	79
0	94	6	71	3	81	7	85	1	66	2	90	4	76	5	58	8	93	9	97
3	50	0	59	1	82	8	67	7	56	9	96	6	58	4	81	5	59	2	96

+++++

JSSP

- let J be the set of N jobs $\{J_1, J_2, \dots, J_N\}$,
- let M be the set of R machines $\{M_1, M_2, \dots, M_R\}$
- Each job J_i must be processed once on every machine M_j , so each job consists of a chain of m operations.

Variables:

- $Y_{mj_1j_2} \rightarrow 1$ if, in machine m , operation from job j_1 is processed before the operation from j_2 .
- X_{jm} is the start time of the operation from job j processed on machine m .
- C_{max} is the time in which all operations for all jobs are completed.

JSSP

min C_{max}
 s.t.

$$\forall j \in J, \forall m \in M : X_{jm} \leq K$$

$$\forall j \in J, \forall m \in M : X_{jm} \geq 0$$

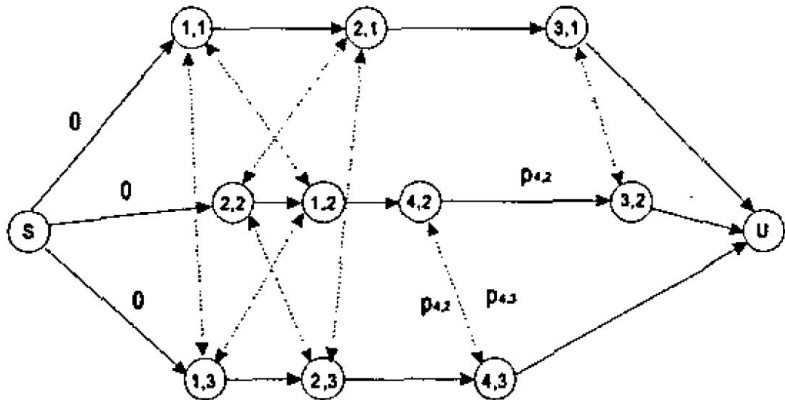
$$\forall j \in J, \forall o \in \{2, \dots, R\} : X_{j, machineat[j, o-1]} + p_{j, machineat[j, o-1]} \leq X_{j, machineat[j, o]}$$

$$\forall j_1 \in J, \forall j_2 \in J, \forall m \in M : X_{j_1 m} \geq X_{j_2 m} + p_{j_2 m} - K * Y_{mj_1 j_2}, j_1 \neq j_2$$

$$\forall j_1 \in J, \forall j_2 \in J, \forall m \in M : X_{j_2 m} \geq X_{j_1 m} + p_{j_1 m} - K * (1 - Y_{mj_1 j_2}), j_1 \neq j_2$$

$$\forall j \in J : C_{max} \geq p_{j, machineat[j, M]} + X_{j, machineat[j, M]}$$

JSSP - Graph



JSSP - Graph

- Consider a directed graph G with a set of nodes A and two sets of arcs P and D .
- The nodes A are all pairs (i, j) where i is a machine and j is the operation of job j on this machine.
- The **conjunctive** arcs P represent the routes of the jobs.
- Two operations that belong to two different jobs and that have to be processed on the same machine are connected to one another by two so-called **disjunctive** arcs.
- The disjunctive arcs D form m cliques of double arcs, one clique for each machine l .
- We denote this graph by $G = (A, P, D)$.

JSSP with setups

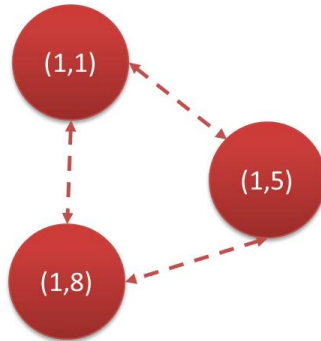
- Machines often have to be reconfigured or cleaned **between jobs**.
- This process is known as a changeover or **set-up**.
- If the length of the set-up depends on the job just completed and on the one about to be started, then the set-up times are **sequence-dependent**.

Previous model:

- Whenever the binary variable $Y_{mj_1j_2} = 1$ we know that operation j_1 precedes operation j_2 ,
- but we do not know if i comes immediately before j or if some other operation is performed among them.

Example: $Y_{115} = 1, Y_{158} = 1, Y_{118} = 1 \rightarrow Y_{118} = 1$ is redundant

JSSP with setups



JSSP with setups

Now, we want to change the model such that $Y_{mj_1j_2} = 1$ means, that job 1 is processed **directly** before job 2.

Prerequisite:

- prevent, that **redundant** disjunctive arcs are active
- consider, we have n arcs $x_{ij} \in \{0, 1\}, i, j \in \{1, \dots, n\}$, we need the constraints:
- $\sum_{i=1}^n \sum_{j=1; j \neq i}^n x_{ij} = n - 1$
- $\sum_{i=1; i \neq j}^n x_{ij} \leq 1, \forall j \in \{1, \dots, n\}$
- $\sum_{j=1; j \neq i}^n x_{ij} \leq 1, \forall i \in \{1, \dots, n\}$

JSSP with setups

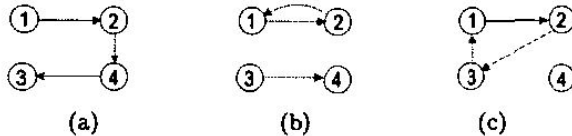


Fig. 3. Feasible solutions.

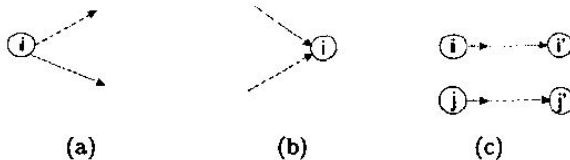


Fig. 4. Unfeasible solutions.

Updated JSSP-Model with setup times $s_{j_1j_2}$:

$$\forall j \in J, \forall m \in M : X_{jm} \leq K$$

$$\forall j \in J, \forall m \in M : X_{jm} \geq 0$$

$$\forall j \in J, \forall o \in \{2, \dots, R\} : X_{j, \text{machineat}[j, o-1]} + p_{j, \text{machineat}[j, o-1]} \leq X_{j, \text{machineat}[j, o]}$$

$$\forall j_1 \in J, \forall j_2 \in J, \forall m \in M : X_{j_1m} \geq X_{j_2m} + p_{j_2m} + s_{j_2j_1} - K * Y_{mj_1j_2}, j_1 \neq j_2$$

$$\forall j_1 \in J, \forall j_2 \in J, \forall m \in M : X_{j_2m} \geq X_{j_1m} + p_{j_1m} + s_{j_1j_2} - K * (1 - Y_{mj_1j_2}), j_1 \neq j_2$$

$$\forall j \in J : C_{\max} \geq p_{j, \text{machineat}[j, M]} + X_{j, \text{machineat}[j, M]}$$

$$\forall m \in M : \sum_{j_1=1}^N \sum_{j_2=1, j_1 \neq j_2}^N Y_{mj_1j_2} = N - 1$$

$$\forall m \in M : \sum_{j_1=1, j_1 \neq j_2}^N Y_{mj_1j_2} \leq 1, \forall j_2 \in J$$

$$\forall m \in M : \sum_{j_2=1, j_1 \neq j_2}^N Y_{mj_1j_2} \leq 1, \forall j_1 \in J$$

Network Flow Programming

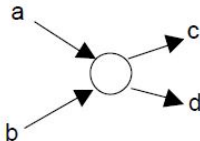
- Linear Programs often can be depicted as networks and vice versa.
- Sometimes there are specialized algorithms that can solve network linear programs many times faster than if they are solved by the general-purpose simplex method.
- Formulating and solving network problems via linear programming is called **network flow programming**.
- **Variables x_i are the unknown flows in the arcs of the network.**

Now we look at some modeling techniques.

Flow Conservation

The total flow into a node equals the total flow out of a node:

$$\sum_{\text{outflows}} x_i - \sum_{\text{inflows}} x_i = 0 \quad (1)$$



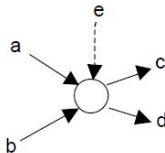
$$c + d - a - b = 0$$

(a) flow conserving node

Source and sink nodes

Some nodes are connections to the environment surrounding the network. At these entryway nodes, there may be a net gain of flow into the network (source node), or a net loss of flow out of the network (sink node). b_i is a positive constant for a source node, and a negative constant for a sink node.

$$\sum_{\text{outflows}} x_i - \sum_{\text{inflows}} x_i = b_i \quad (2)$$



$$c + d - a - b = e$$

(b) source node

Objective function

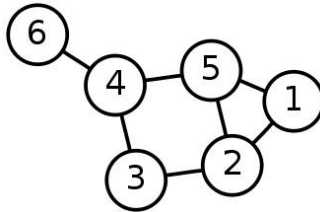
In a minimum cost network flow problem, the objective is to find the values of the variables (the x_j) that minimize the total cost of the flows over the network:

$$\min \sum_{arcs} c_j x_j \quad (3)$$

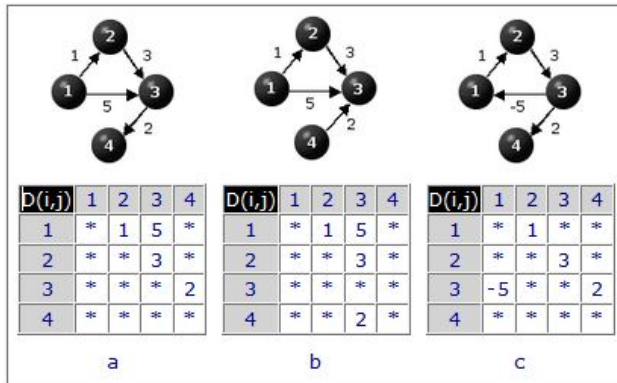
Shortest Path Problem

Single Source Shortest Path Problem:

- The problem of finding *shortest paths* from a source vertex v to all other vertices in the graph.

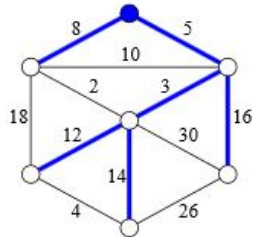


Shortest Path Problem



- a) has optimal solution for nodes 1, 2, 3, 4
- b) has no solution for node 4
- c) no optimal solution exists (**paths may include cycles!**)

Shortest Path Problem



Application area: Routing in IP networks

Shortest Path Problem - MIP

Question

How can we model this as MIP?

Shortest Path Problem - MIP

- d_{ij} distance between node i and node j
- $T \rightarrow$ node to which the shortest path is to be found

Variables:

- $x_{ij} \in \{0, 1\} \rightarrow$ arcs in the network

$$d_{ij} * x_{ij}$$

s.t.

$$\begin{aligned} \sum_j x_{ij} - \sum_i x_{ji} &= 0 \\ \sum_j x_{0j} - \sum_i x_{j0} &= 1 \\ \sum_j x_{iT} - \sum_i x_{Ti} &= -1 \end{aligned}$$

Definitions for Shortest Path Problem

- Let $N = \{1, 2, \dots, n\}$ denote the set of Nodes,
- for each node j in N let $P(j)$ denote the set of its immediate predecessors,
- and let $S(j)$ denote the set of its immediate successors.
- $D(i, j)$ is the distance between nodes i and j .

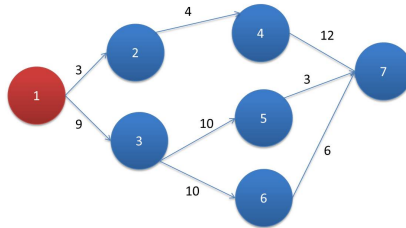
Dynamic Programming Perspective

Bellman's Principle of Optimality

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

- The state of the optimal decision is **not** independent from the state of previous decisions.
- → Dynamic programming is not equal to *divide and conquer*, where the subproblems are independent from each other.

Dynamic Programming Perspective



Consider the the arcs connected with node 7:

Question

Which arc lies on the optimal path to node 7?

- a) 4 – 7
- b) 5 – 7
- c) 6 – 7

Dynamic Programming Perspective

- It is **not** the arc with the smallest weight (5 – 7)!
- Because we have to consider the shortest paths to the nodes 4, 5 and 6!
 - ① shortest path to 4 = 12 $\rightarrow$ $12 + 12 = 24$
 - ② shortest path to 5 = 19 $\rightarrow$ $19 + 3 = 22$
 - ③ shortest path to 6 = 19 $\rightarrow$ $19 + 6 = 25$

Dynamic Programming Perspective

$f(j)$:= length of the shortest path from node 1 to node j

Dynamic programming functional equation

- $f(j) = \min \{D(k, j) + f(k) : k \in P(j)\}$, if $P(j) \neq \{\}$
- $f(j) = \text{Infinity}$, if $P(j) = \{\}$ and $j > 1$
- $f(1) = 0$, (We assume that $P(1) = \{\}$)

Find optimal path to node j :

- ① for each predecessor k in $P(j)$, find optimal path to node k
- ② add $D(k, j)$ to the length of each optimal path
- ③ choose the shortest path, this is the optimal path to node j

Dynamic Programming Perspective

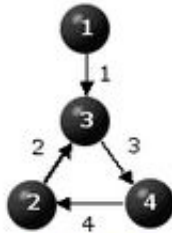


Figure 3

$$f(1) = 0$$

$$f(2) = 4 + f(4)$$

$$f(3) = \min \{2 + f(2), 1 + f(1)\}$$

$$f(4) = 3 + f(3)$$

Dijkstra Algorithm

Dijkstra Shortest Path Algorithm

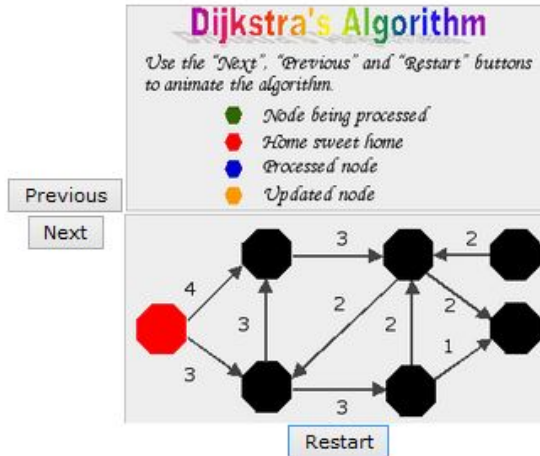
- is a **non-recursive** solution to the single source shortest path algorithm in graph theory
- works on both directed and undirected graphs. However, all edges must have nonnegative weights.

Dijkstra Algorithm

- ① **Start at origin node:** $F(\text{origin})=0$, $F(j)=\text{infinity}$ for all other nodes.
- ② **update the immediate successors $i, i \in S(0)$:**
$$f(i) = D(0, i) + f(0)$$
- ③ identify the next nearest (already visited) node j to the origin
- ④ **update the immediate successors of $k, k \in S(j)$:**
$$f(k) = \min \{D(k, p) + f(p) : p \in P(k)\}$$
- ⑤ repeat steps 3 and 4 until all nodes are visited

Dijkstra Algorithm

Demo:



Equipment Replacement Problem

Description - part 1

A bicycle courier must have a bicycle at all times that he can use for his deliveries. He wants to determine the cheapest way to buy and maintain bicycles over a 5 year timeframe. He has collected some information about costs.

Equipment Replacement Problem

Description - part 2

- A new bicycle costs **\$500**.
- Maintenance costs vary with the age of the bike: **\$30** in the first year, **\$40** in the second year, and **\$60** in the third year.
- Bicycles will last a maximum of **three years** before they are sold.
- He can get **\$400** if he sells a bicycle after one year of use, **\$300** after two years of use, and **\$250** after three years of use.

Equipment Replacement Problem

Description - part 3

He has no bicycle at the moment. When should he buy bicycles and how long should he keep them to minimize his total costs over the 5 - year timeframe?

Let's formulate this as a dynamic programming problem.

Equipment Replacement Problem

Parameters:

- c_{tx} , net cost of buying a bicycle at year t and operating it until year x , including the initial purchase cost, the maintenance costs, and the salvage value when it is sold.
- Keep bicycle 1 year:

$$c_{01} = c_{12} = c_{23} = c_{34} = c_{45} = 500 + 30 - 400 = \$130$$
- Keep bicycle 2 years:

$$c_{02} = c_{13} = c_{24} = c_{35} = 500 + (30 + 40) - 300 = \$270$$
- Keep bicycle 3 years:

$$c_{01} = c_{03} = c_{14} = c_{25} = 500 + (30 + 40 + 60) - 250 = \$380$$

Equipment Replacement Problem

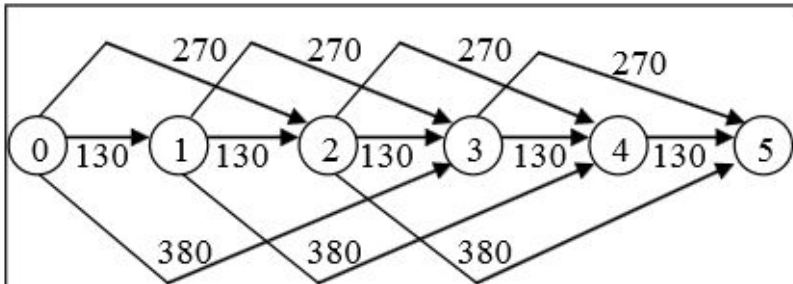


Figure 15.1: Converting the equipment replacement problem to a shortest route problem.

Equipment Replacement Problem

- Parameters c_{tx} are **weights of the vertices** of the graph.
- Years 1 to 5 are the **nodes** of the graph.
- Let $f(t)$ be the total cost from year 1 to year t
→ **Goal:** $\min f(5)$

Possible solutions:

- $f(5) = c_{01} + c_{13} + c_{35} = 570$
- $f(5) = c_{03} + c_{35} = 650$
- each path from node 0 to node 5 is a solution

Dynamic programming functional equation

- $f(t) = \min \{c(k, t) + f(k) : k \in P(t)\}$, if $P(t) \neq \{\}$
- $f(1) = 0$

Traveling Salesman Problem

Question

What is the difference between the Shortest Path Problem and the Traveling Salesman Problem?

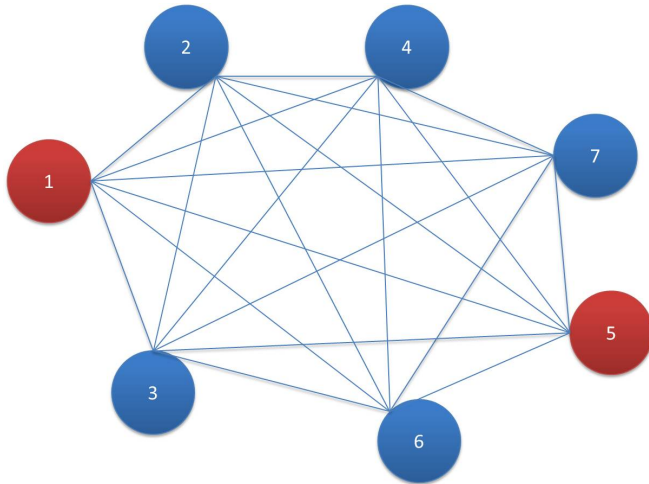
Traveling Salesman Problem

Similarity SPP - TSP: find a path from node 1 to node N

Differences SPP - TSP:

- **TSP:** all nodes must be on the path to node N
- **SPP:** don't need to visit all nodes on the path
- **TSP:** only looking for the path from node 1 to node N
- **SPP:** looking for a path to every single node of the network

Traveling Salesman Problem



Traveling Salesman Problem

How can we use Dynamic Programming to solve the TSP?

Given $S \subseteq N$ with $1 \in S$ and given $j \neq 1, j \in S$, let $C(S, j)$ be the shortest path that starting at 1, visits all nodes in S and ends at j .

Notice:

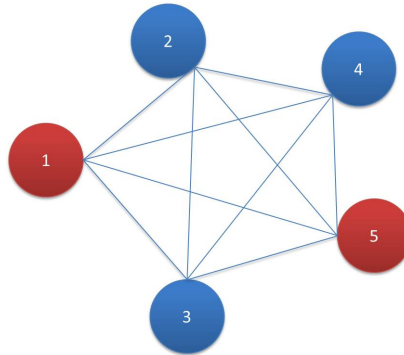
- if $|S| = 2$, then $C(S, k) = d_{1,k}$ for $k=2,3,\dots,n$
- if $|S| > 2$, then $C(S, k)$ is determined as follows:
 - ① for each $m \in S - \{k\}$, find the optimal tour from 1 to m ,
 - ② add $D(m, k)$ to the length of each tour,
 - ③ choose the shortest tour.

Dynamic programming functional equation

$$C(S, k) = \begin{cases} d_{1,m}, & S = \{1, k\} \\ \min_{m \neq k, m \in S} [C(S - \{k\}, m) + d(m, k)] & \text{otherwise} \end{cases}$$

Traveling Salesman Problem

Example:



Traveling Salesman Problem

$$C(\{1, 2, 3, 4, 5\}, 5) = \min[$$

- $C(\{1, 2, 3, 4\}, 4) + D(4, 5),$
- $C(\{1, 2, 3, 4\}, 3) + D(3, 5),$
- $C(\{1, 2, 3, 4\}, 2) + D(2, 5),$
- $C(\{1, 2, 3, 4\}, 1) + D(1, 5)]$

$$C(\{1, 2, 3, 4\}, 4) = \min[$$

- $C(\{1, 2, 3\}, 3) + D(3, 4),$
- $C(\{1, 2, 3\}, 2) + D(2, 4),$
- $C(\{1, 2, 3\}, 1) + D(1, 4)],$

usw.

Das war's für heute :-)