Linear and Combinatorial Optimization

Sara Maad Sasane

Center for Mathematical Sciences, Lund University

1 March 2019

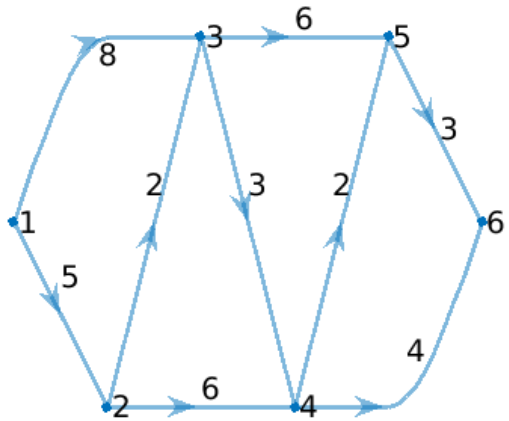
The shortest route problem

The problem.

Given a graph or a directed graph with distances on each edge, what is the shortest distance between two particular nodes?

Example

We would like to travel from node 1 *to node* 6 *in the digraph in the figure to the right, as quickly as possible. On every edge, the time of travel is marked. What route should we take, and how long will it take?*



Network for quickest path

The shortest route problem

We formulate the shortest route problem as an LP problem: We are given a digraph $G = (V, E)$ with nodes V and edges E and costs c_{ij} if $(i, j) \in V$. The variables are $x_{ij} \in \{0, 1\}$, and $x_{ij} = 1$ if and only if the edge from node i to node j belongs to the route. The route should start in node s and end in node t, and for the other nodes we should have that if one edge in the route ends in node j, then another edge that leaves node j has to be part of the route too (and vice versa). The problem is then

$$\text{minimize} \sum_{(i,j)\in E} c_{ij}x_{ij}$$

$$\text{subject to} \begin{cases} \displaystyle\sum_{j:(j,i)\in E} x_{ji} - \sum_{j:(i,j)\in E} x_{ij} = \begin{cases} -1 & \text{if } i = s \\ 1 & \text{if } i = t \\ 0 & \text{otherwise,} \end{cases} \text{ for all } i \in V. \\ x_{ij} \geq 0 \qquad (i, j) \in E. \end{cases}$$

The shortest path problem

The reason why the last constraint is $x_{ij} \geq 0$ and not $x_{ij} \in \{0, 1\}$ is because of the following theorem:

Theorem

If $c_{ij} \geq 0$ for all (i, j) then the optimal solution (if it exists) satisfies $x_{ij} \in \{0, 1\}$.

- The problem for a graph instead of a digraph can be solved with the same methods after forming a digraph from the graph by replacing each edge by two edges of different direction.
- The costs can be permitted to be negative when the graph is directed. For undirected graphs with some negative costs, the solution is always unbounded, and so for undirected graphs, we require the costs to be nonnegative.

The shortest route problem

- The solution methods that we will see use the dual problem, and so we need to formulate this first.
- The number of constraints in the primal problem is the same as the number of nodes, and so we denote the dual variables by y_i, with the special names y_s and y_t for the dual variables corresponding to the source and sink nodes, respectively.
- The dual problem is

$$\begin{cases} \text{maximize } y_t - y_s \\ \text{subject to } y_j - y_i \leq c_{ij}, \qquad (i,j) \in E. \end{cases}$$

- Just as for the transportation problem, one of the constraints in the primal problem is redundant.
- The consequence of this for the dual problem is that the solution is not completely unique, but if we specify one of the dual variables (e.g. putting $y_s = 0$), then the rest of the dual variables will be uniquely determined.

Dijkstra's method

- In our first encounter with the shortest route problem (see Lecture 9), we had an informal solution method involving a distance d which we increased in steps, while considering nodes that could be reached with a path of maximum length d from the origin s.
- It turns out that the values of this variable d is exactly the values of the dual variables y_j for the optimal solution. These are called the node prices.
- We first prove that the value of the dual variable y_k cannot be higher than the length of any path from s to k.
- For a path P_{sk} from node s to node k, let $c(P_{sk})$ denote the sum of all the costs for edges belonging to P_{sk}

The shortest route problem

Theorem

If $\mathbf{y}$ *is a feasible solution of the dual problem, and* P_{sk} *is a path from node* s *to node* k*, then* $c(P_{sk}) \geq y_k$*.*

Proof.

$$c(P_{sk}) = \sum_{(i,j)\in P_{sk}} c_{ij} \geq \sum_{(i,j)\in P_{sk}} (y_j - y_i) = y_k - y_s = y_k,$$

where we have used that the sum telescopes. □

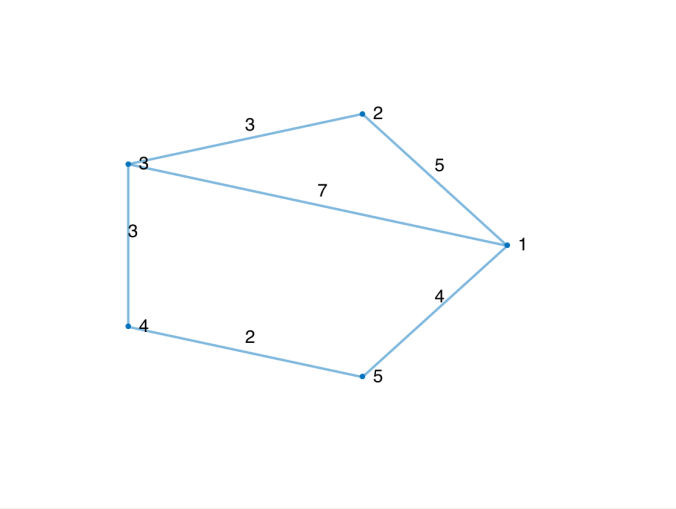
- It follows that any path from s to k for which the cost is y_k must be a shortest path.

Dijkstra's algorithm

- We will describe the same algorithm as we introduced in Lecture 9, but we will be more detailed, so that the method becomes unambiguous and as efficient as possible.
- Besides y_k being the node prices, we will also need the notation p_k for the predecessor index, and ν_k and q_k will be the temporary node price and predecessor index, respectively.
- Recall that the idea of the algorithm is to use two sets of nodes N_r and N_u, where N_r consists of the nodes that can be reached from s within an allowed distance, while N_u are the nodes that cannot (yet) be reached.
- Before the general algorithm, we will work through the example from Lecture 9 with this new notation.

Dijkstra's algorithm

Example

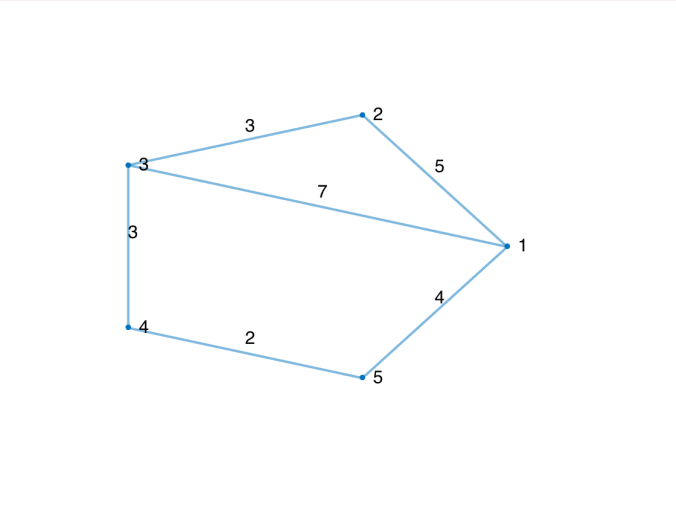


What is the shortest distance between node 1 and node 4?

- *We choose node* $s = 1$ *as the origin. Initially,* $N_r = \{1\}$ *and* $N_u = \{2, 3, 4, 5\}$*. For* $j \in N_u$*, let* $\nu_j = \infty$ *(or some large number, e.g. the sum of all the edge costs).*
- *We label the nodes* 2, 3 *and* 5 *(that are reachable from node* 1*) with temporary node prices* $\nu_2 = 5$*,* $\nu_3 = 7$ *and* $\nu_5 = 4$*, and let* $q_j = 1$ *for* $j = 2$, 3 *and* 5*. Since* $\nu_5 = \min_{j \in N_u} \nu_j$*, we let* $y_5 = 4$ *and* $p_5 = 1$*. Let* $N_r = \{1, 5\}$ *and* $N_u = \{2, 3, 4\}$*.*

Dijkstra's algorithm

Example (Cont.)

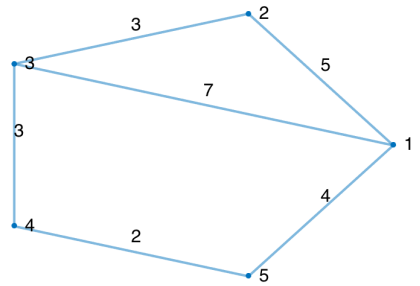


What is the shortest distance between node 1 and node 4?

- *We can now (temporarily) label node* 4 *with* $\nu_4 = 4 + 2 = 6$ *and* $q_4 = 5$. *Since* $\nu_2 = \min_{j \in N_u} \nu_j = 5$, *we let* $y_2 = 5$ *and* $p_2 = 1$. *We update* $N_r = \{1, 2, 5\}$ *and* $N_u = \{3, 4\}$.
- *Node* $3 \in N_u$ *can be reached from node* 2 *(as well as from node* 1*). Recall that* $\nu_3 = 7$ *since before, and now we compare it with* 8, *which is larger so we don't change* ν_3. *We have* $\min_{j \in N_u} \nu_j = \nu_4 = 6$, *and so we let* $y_4 = 6$ *and* $p_4 = q_4 = 5$. *Let* $N_r = \{1, 2, 4, 5\}$ *and* $N_u = \{3\}$.

Dijkstra's algorithm

Example (Cont.)



What is the shortest distance between node 1 and node 4?

- *Node* $4 = t$ *has been labelled with the permanent node price* 6, *which is the shortest distance from node* 1 *to node* 4.
- *We can find a shortest path by following the predecessor index:* $p_4 = 5$ *and* $p_5 = 1$, *which tells us that an optimal path is* $1 \to 5 \to 4$.

Dijkstra's algorithm

In the examples, we followed the following rules, which defines Dijkstra's algorithm. We only need to specify the set of unreached nodes N_u, since $N_r = V \setminus N_u$.

1. Let $N_u = V \setminus \{s\}$ and $y_s = 0$. For every $j \in N_u$, let ν_j be a large number (e g $\sum_{(i,j)\in E} c_{ij}$).
2. For nodes j such that $(1, j) \in E$, let $\nu_j = c_{1j}$. and $q_j = 1$.
3. For $k = \text{argmin}_{j\in N_u} \nu_j$ let $y_k = \nu_k$ and $p_k = q_k$.
4. Let $N_u = N_u \setminus \{k\}$.
5. If $t \notin N_u$ (or $N_u = \emptyset$), then stop.
6. For every $j \in N_u$ for which $(k, j) \in E$, if $y_k + c_{kj} < \nu_j$, let $q_j = k$ and $\nu_j = y_k + c_{kj}$.
7. Repeat from 3.

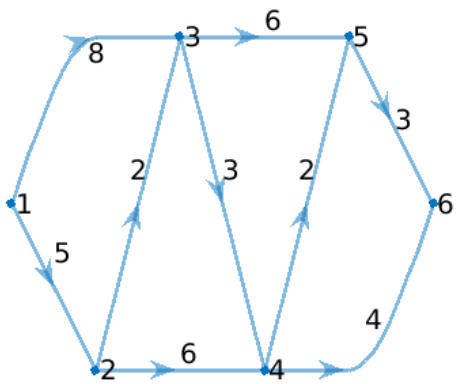
Dijkstra's algorithm

- The number of iterations in the algorithm can be at most $|V|$, the number of nodes, and in every iteration the number of operations is at most proportional to $|V|$. This shows that the complexity of Dijkstra's algorithm is $\mathcal{O}(|V|^2)$ as $|V| \to \infty$.
- It is recommended that you go through the example again, and check that the steps 1–6 were followed.

Bellman's equations

- Dijkstra's method is applicable if all the cost are nonnegative.
- Bellman's method that we will see now is applicable also when (some of) the costs are negative, but instead it is required that the digraph is acyclic.
- In particular, the method is not applicable to undirected graphs (which always have cycles).
- As an example, we will study the digraph of quickest time as we saw earlier:

Example



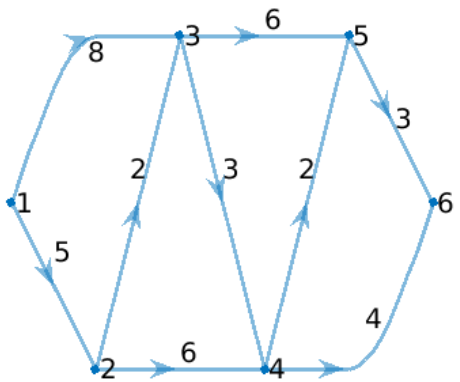
Network for quickest path

Bellman's equations

- If the digraph is acyclic, it is possible to renumber the nodes so that all edges go from a node with a lower index to a node with a higher index.
- It is then possible to label the nodes in order (with node prices and predecessor indices), and temporary labels are not required.
- We will demonstrate the method for the example before stating the general algorithm.

Bellman's equations

Example



Network for quickest path

- *We see that this digraph is already labelled so that* $i < j$ *for every* $(i, j) \in E$*, so we don't need to renumber the nodes.*

- *We start by letting* $y_1 = 0$*, and continue marking nodes with node prices and predecessor indices.*
- *We get* $y_2 = 5$ *and* $p_2 = 1$*.*
- $y_3 = \min(8, 5 + 2) = 7$ *and* $p_3 = 2$*.*
- $y_4 = \min(7 + 3, 5 + 6) = 10$ *and* $p_4 = 3$*.*
- $y_5 = \min(7 + 6, 10 + 2) = 12$ *and* $p_5 = 4$*.*
- $y_6 = \min(10 + 4, 12 + 3) = 14$ *and* $p_6 = 4$*.*

Bellman's equations

Example (Cont.)

- *All nodes have been marked, and we have found that the shortest time it can take to go from node* 1 *to node* 6 *is* 14.
- *We found that* $p_6 = 4$, $p_4 = 3$, $p_3 = 2$ *and* $p_2 = 1$, *and so an optimal path is* $1 \to 2 \to 3 \to 4 \to 6$.

Now we state the general algorithm:

1. Renumber the nodes so that $i < j$ for all $(i, j) \in E$.
2. Let $y_1 = 0$.
3. For $j = 1, \ldots, n$, let $k = \text{argmin}_{i:(i,j)\in E}(y_i + c_{ij})$ and then let $y_j = y_k + c_{kj}$ and $p_j = k$.

The equations in step 3 are called Bellman's equations.

Bellman's equations

- The complexity of the algorithm is $\mathcal{O}(|V|^2)$, assuming that a sorting algorithm with the same complexity (or better) is used in step 1.
- If the graph has cycles and some negative costs, neither Dijkstra's algorithm nor Bellman's equations are applicable. For this situation, it is possible to solve the problem with Ford's method. See Holmgren's book.
- Note that the approach of Bellman's equations is recursive: The nodeprices were computed in order, and we used the results of the previous step when computing the nodeprices for the next step.
- We will now use the same idea for this and other problems when we introduce Dynamic programming.

1. The shortest route problem

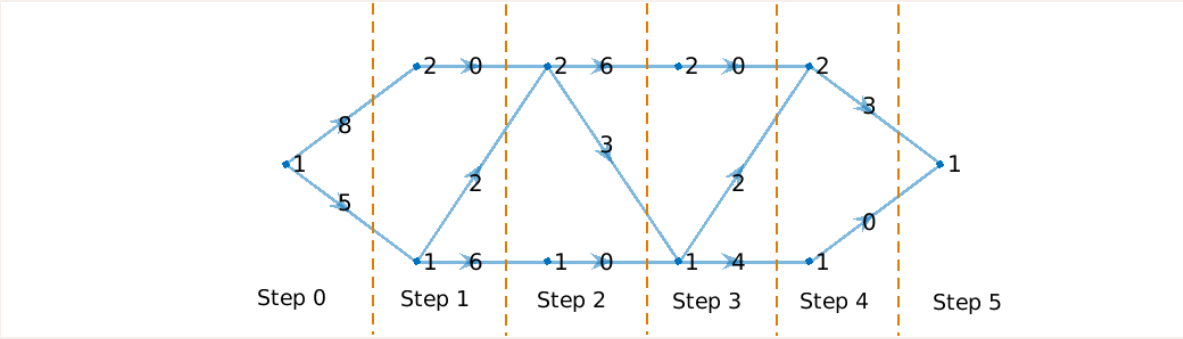
2. **Dynamic programming**

Dynamic programming for the shortest route problem

- We will introduce dynamic programming using the same shortest route problem solved with Bellman's equations.

Example

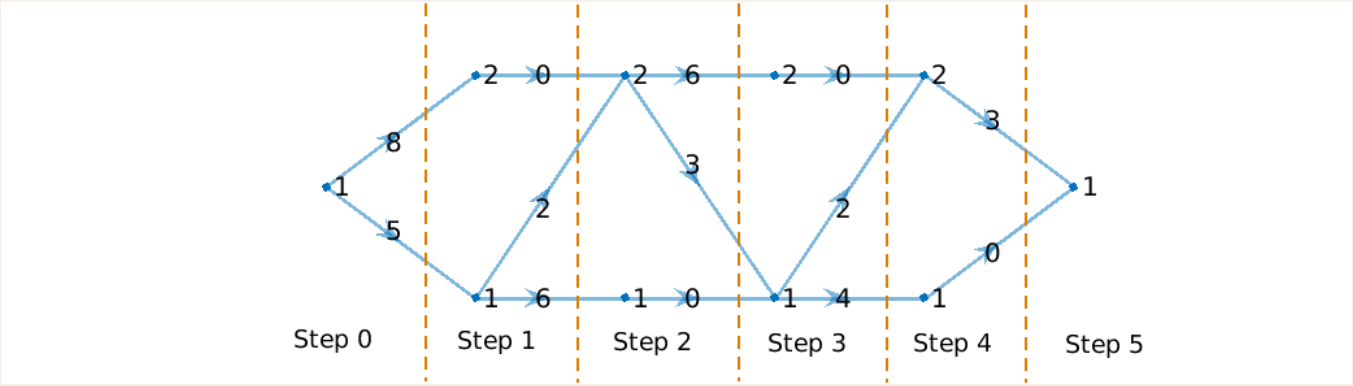
We use the same graph as before, but organize it into levels (steps). We need all arrows to go from a node in level j to a node in level $j+1$, and for this reason, we have introduced some new nodes and then renumbered all the nodes.



The graph of quickest path organized for dynamic programming.

Dynamic programming for the shortest route problem

Example



- *The nodes in each step are are denoted by s_k and called states.*
- *For each state, we compute the node price $f_k(s_k)$.*
- *The steering x_k (in this example), is the set of possible nodes in step $k-1$ that lead to a certain node in step k.*
- *The optimal steering, $\widehat{x}_k(s_k)$ is the best choice of steering to the node s_k.*
- *There is also a transfer function which for this example happens to be the same as the optimal steering: $T_k(s_k) = \widehat{x}_k$.*

Dynamic programming for the shortest route problem

Example (Cont.)

The following table with s_k, $f_k(s_k)$, x_k and $\widehat{x}_k(s_k)$ can be computed:

Level	*States*	*Node prices*	*Steering*	*Optimal*
k	s_k	$f_k(s_k)$	$\{x_k(s_k)\}$	*steering* $\widehat{x}_k(s_k)$
0	1	0	$\emptyset$	–
1	1	5	$\{1\}$	1
	2	8	$\{1\}$	1
2	1	$5+6=11$	$\{1\}$	1
	2	$\min(5+2, 8+0)=7$	$\{1,2\}$	1
3	1	$\min(11+0, 7+3)=10$	$\{1,2\}$	2
	2	$7+6=13$	$\{2\}$	2
4	1	$10+4=14$	$\{1\}$	1
	2	$\min(10+2, 13+0)=12$	$\{1,2\}$	1
5	1	$\min(14+0, 12+3)=14$	$\{1,2\}$	1

Dynamic programming for the shortest route problem

Example (Cont.)

- *In the table, the recursive formula*

$$f_k(s_k) = \min_{x_k}(f_{k-1}(x_k) + c_k(x_k, s_k))$$

  was used to find the node prices, where the minimization is taken over all possible steerings $x_k(s_k)$.
- *The steering that gives the minimum above, is the optimal steering $\widehat{x}_k(s_k)$.*
- *We will now try a dynamic programming approach to the knapsack problem as well.*

Dynamic programming for the knapsack problem

Example

The problem that we will solve is

$$\begin{aligned} &\textit{maximize } 7y_1 + 2y_2 + 4y_3 \\ &\textit{subject to } \begin{cases} 2y_1 + 3y_2 + 2y_3 \leq 4, \\ y_1, y_2 \in \{0, 1\}, \\ y_3 \in \{0, 1, 2\}. \end{cases} \end{aligned}$$

In general, the method will work on problems of the form

$$\begin{aligned} &\text{maximize } \sum_{j=1}^{n} c_j y_j \\ &\text{subject to } \begin{cases} \sum_{j=1}^{n} a_j y_j \leq b, \\ 0 \leq y_j \leq u_j, \text{ integers, } j = 1, \ldots, n. \end{cases} \end{aligned}$$

Dynamic programming for the knapsack problem

Example (Cont.)

We will use a network for the dynamic programming, where each variable (item) makes one level. Level 0 *will correspond to a slack item of no value.*

State s_k: *The part of the RHS* b *to be used for the first* k *items (variables) + slack.*

Steering x_k: *The number (of copies) of item* k *that were chosen in step* $k-1$.

Transfer function s_{k-1}: $T_k(s_k, x_k) = s_k - a_k x_k$ *(predecessor index)*

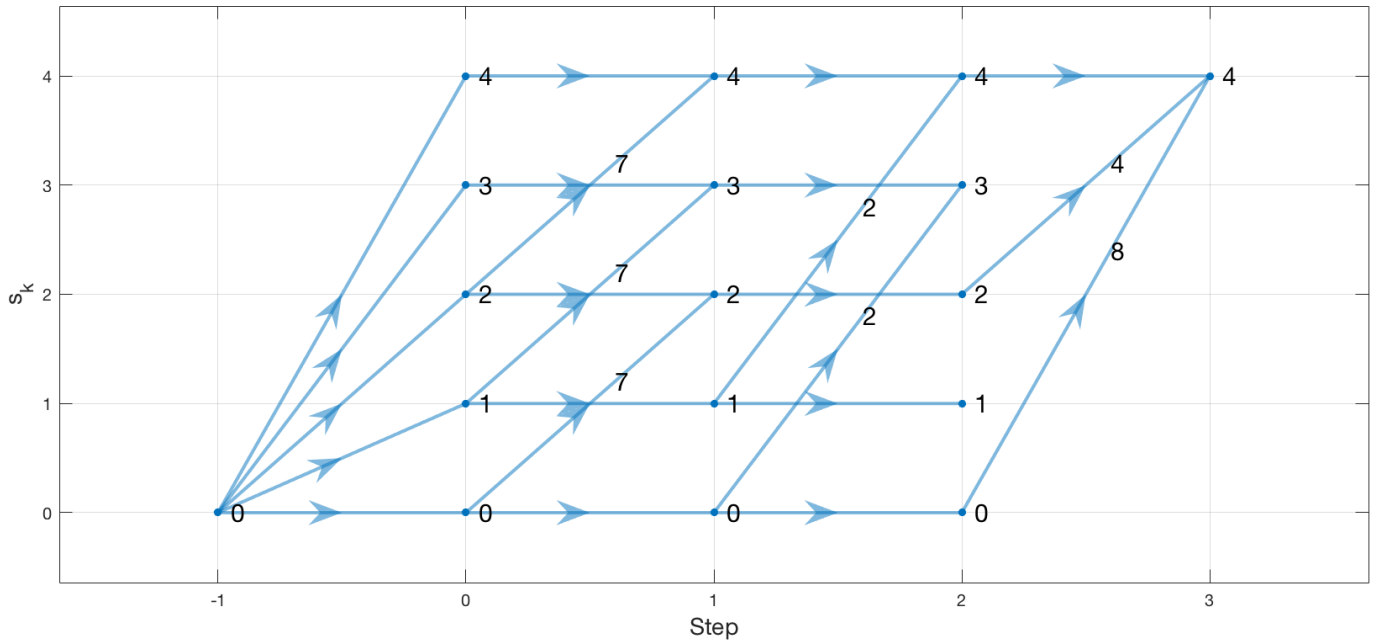
Boundary constraints: $f_0(s_0) = 0$, $s_0 \geq 0$, $s_n = b$ (s_0 *is the slack.)*

Constraints: $0 \leq x_k \leq u_k$ *integers.* $0 \leq s_k \leq b$, $x_k \leq \lfloor s_k/a_k \rfloor$.

Objective function value: $f_k(s_k)$ *is the maximum value of a knapsack of size* s_k *using items* $1, \ldots, k$.

Dynamic programming for the knapsack problem

Example (Cont.)



Dynamic programming for the knapsack problem. The steps corresponds to the variables/items + slack, and on the y-axis, we mark how much of the knapsack that has been used.

Dynamic programming for the knapsack problem

Example (Cont.)

- *Each step correspond to a variable (item), except level* 0 *which corresponds to a slack item (air).*
- *The edges that are not marked have value* 0*.*
- *Each node should be marked with a value and the steering. Fill in this yourself!*
- *The problem becomes that of maximizing the total value.*
- *For example, the nodes in step* 0 *all have value* 0*, and steering* 0, 1, 2, 3 *and* 4*, respectively.*
- *The nodes in step* 1 *have values* 0, 0, 7, 7 *and* 7*, while the steering for those nodes are* 0, 0, 1, 1, *and* 1*, respectively. (*0 *if item* 1 *is not chosen, and* 1 *if item* 1 *is chosen).*

Dynamic programming for the knapsack problem

Example

- *We label the nodes in step* 2 *as follows: Clearly, both node* 0 *and node* 1 *have value* 0*, and steering* 0 *(since no item* 2 *were chosen to reach these nodes).*
- *Node* 2 *and* 3 *in step* 2 *both have value* 7 *and steering* 0*, since they can only be reached from node* 2 *and* 3 *of step* 1 *whose value are* 7*, via an edge of value* 0*. Finally, node* 4 *has value* $\max(7, 2) = 7$ *and steering* 0*.*
- *The node in step* 4 *has value* $\max(8, 7 + 4, 7) = 11$ *and steering* 1 *(since we need to bring one copy of item* 3 *to achieve the optimum).*
- *The optimum value of the knapsack is* 11*, and this is achieved if we bring one copy of item* 1*, none of item* 2 *and one of item* 3*.*

Dynamic programming for the knapsack problem

- In a previous lecture, I mentioned that the knapsack problem belongs to the class of *NP*-complete problems.
- Actually, that only applies to the decision version of the problem: Can a value of at least f be achieved without exceeding the weight/size b?
- The optimization version of the problem is *NP*-hard.
- A dynamic programming for the knapsack problem is *pseudopolynomial*: The algorithm is $\mathcal{O}(nb)$, where n is the number of items and b is the maximum weight/size of the knapsack. So, in other words, we can solve all knapsack problems with allowed knapsack sizes less than any *fixed* number b in polynomial time (even in linear time), but there is no polynomial algorithm without this bound on the knapsack size.