Greedy Method (Ahne menetelmä)

Optimization heuristics that make locally best-looking choices

Example (Coin problem)
Pay 85 cents using minimal number of coins {5 Eur, 1 Eur, 50 c, 20 c, 10 c, 5 c}

Obvious solution: 85 = 50 + 20 + 10 + 5

Method: Choose the largest feasible coin, and subtract it from remaining sum

Greed doesn’t always lead to a global optimum: With coins 1c, 10c, and 25c, and sum 30, the greedy solution

\[ 30 = 25 + 1 + 1 + 1 + 1 + 1 \]

is less optimal than

\[ 30 = 10 + 10 + 10 \]

Example (Prim’s algorithm)

Given a connected graph \( G = (V, E) \), find a minimal spanning tree for it:

a connected acyclic subgraph

that contains all nodes of \( G \) and

whose total edge-weight/length is minimal

Prim’s algorithm finds a minimal spanning tree greedily:

An implementation

Maintain for uncovered nodes \( v \) their closest neighbor \( n[v] \) in \( P \), and distance \( d[v] \) to it:

procedure Prim(V: setOfNodes, E: setOfEdges) returns tree

Choose any \( u \in V \);

\( S := \{u\}; T := \emptyset; \)

while \( S \neq V \)

choose \( (u,v) \in E \) with smallest weight, such that \( u \in S \) and \( v \notin S \);

\( E := E \setminus \{(u,v)\}; \)

\( S := S \cup \{v\}; \)

\( T := T \cup \{(u,v)\}; \)

return \( T \);
Example (Weighted knapsack)
Pack into volume $T$ a maximally valuable load of items with volumes and values
$(t_1, a_1), \ldots, (t_n, a_n)$, that is, find $x_i \in \{0, 1\}$ s.t.
\[
\sum_{i} x_i t_i \leq T, \quad \sum_{i} x_i a_i = \max
\]
Greedy solution:

```plaintext
procedure greedyKnapsack(t, a: array of real)
  returns List
  $K := \emptyset$;
  Place pairs $(t_i, a_i)$ into list $L$ in descending order of their unit price $a_i/t_i$;
  repeat
    repeat $(t, a) := \text{head}(L); L := \text{tail}(L)$
    until $t \leq T$ or $L = \emptyset$;
    if $t \leq T$ then
      $T := T - t; K := K \cup \{(t, a)\}$
    until $L = \emptyset$
    return $K$;

$T(n) = O(n \log n)$; load may differ (arbitrarily much) from optimum
```

<table>
<thead>
<tr>
<th>item</th>
<th>$t_i$</th>
<th>$a_i$</th>
<th>$a_i/t_i$</th>
<th>$T = 10$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>9</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.5</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>3</td>
<td>0.5</td>
<td></td>
</tr>
</tbody>
</table>

Optimum: $\{(9, 9), (1, 0.5)\}$

(Arbitrarily) bad case: $\{(1, 2), (M, M)\}, T = M$

At least 50% of maximum as follows:

1. $K := \text{greedyKnapsack}(t, a)$;
2. Select the most valuable $(t_i, a_i)$ s.t. $t_i \leq T$;
3. if $a_i > \text{value}(K)$ then
   1. $K := \{(t_i, a_i)\}$;

"1/2-approximation"

There are better ones, but no polynomial time algorithm is known for producing the exact optimum.