tightcenter

panikzettel.htwr-aachen.de

Logistics Systems Planning 1 Panikzettel

Philipp Schröer, Luca Oeljeklaus

Version 1 — 05.08.2020

Contents

1 Introduction

This Panikzettel is about the lecture Logistics Systems Planning 1 by Prof. Schneider held in the winter semester 2019/20 and is based on the corresponding slides.

This Panikzettel is Open Source. We appreciate comments and suggestions at https://git.rwth-aachen.de/philipp.schroer/panikzettel.

In this Panikzettel, we introduce new notation for better readability: $\overline{n} := \{1, \ldots, n\}$ for some n.

2 Logistic Systems

Logistic activities include:

- *Supply logistics*: managing raw materials, parts, etc.,
- *Internal logistics*: stocking, transportation between facilities, packaging, etc.,
- *Distribution logistics*: supplying of sales points, anything between production and sale.

Planning of a logistics system can be split into three decision levels: *strategic*, *tactical*, and *operational*.

Storage

- Strategic:
  - Warehouse planning,
  - Selection of warehouse equipment,
  - Choice of warehouse layout.
- Tactical:
  - Product allocation at storage points,
  - Choice of inventory policies.
- Operational:
  - Warehouse picking,
  - Consolidation in loading area.

Distribution

- Strategic:
  - Choice of transport mode,
  - Fleet size and composition.
- Tactical:
  - Freight assigment,
  - Vehicle assignment,
  - Crew rostering.
- Operational:
  - Vehicle routing,
  - Positioning of vehicles and containers,
  - Consolidation of orders.

3 Graph Theory

3.1 Graphs

Definition: Undirected Graph

An *undirected graph* $G = (V, E)$ consists of
- a *vertex set* V, and
- *edges* $E \subseteq \{ \{v, w\} \in V^2 \mid v \neq w \}$.

The *star* (incident edges) of a vertex v is $\delta(v) = \{ \{v, w\} \in E \}$.

The *degree* of a vertex v is $d(v) = |\delta(v)|$.

The set of *neighbours* of a vertex v is $\Delta(v) = N(v) = \{ w \in V \mid \{v, w\} \in E \}$.

A *mixed graph* $G = (V, A, E)$ consists of both directed and undirected edges. A *multigraph* is an undirected graph with potentially multiple edges between two nodes.

Definition: Directed Graph

A *directed graph* (*digraph*) $G = (V, A)$:
- a *vertex set* V, and
- *arcs* $A \subseteq \{ (v, w) \mid v, w \in V, v \neq w \}$.

The *backward star* (incoming arcs) of v is $\delta^-(v) = \{ (w, v) \in A \mid w \in V \}$. ↳ *forward star* $\delta^+(v)$ with outgoing arcs.

The *in-degree* of v is $d^-(v) = |\delta^-(v)|$.
↳ Similar with *out-degree* $d^+(v)$.

The set of *predecessors* of v is given by $\Delta^-(v) = N^-(v) = \{ w \in V \mid (w, v) \in E \}$. ↳ Similar with *successors* $\Delta^+(v)$.

The *underlying graph* of a graph is obtained by removing the direction of its edges. The number of edges stays the same. A *Hamiltonian path* in an undirected graph visits all nodes exactly once.

An **undirected graph** is *Eulerian* if for every vertex v, $d(v)$ is even.

A **directed graph** is Eulerian if for every vertex v it holds that $d^-(v) = d^+(v)$.

A **mixed graph** is Eulerian if the sum of all degrees is even and for every vertex set $S \subset V$ it holds that $|d^-(S) - d^+(S)| \leq d^u(S)$. This basically states that for any vertex set, either we have the same number of in- and outgoing edges or we have enough undirected edges to compensate the lack of in- or outgoing edges.

A *strong (weak) component* is a maximum set of nodes which are mutually reachable via directed (undirected) paths.

3.2 Minimum Spanning Tree

A *minimum spanning tree* is a subset of edges in an **undirected graph** with edge weights, so that the subset is a tree (all nodes are connected by exactly one path) and with minimum edge weight.

The idea of *Prim's Algorithm* is to select an arbitrary start node and to iteratively build a set of "reached" vertices which at the end should include all vertices.

In every iteration, it looks at the star (set of incident edges) of the "reached" vertex set, that is, all edges that have one endpoint in the set and one outside and selects the one with the lowest weight. It then adds the newly reached vertex to the set.

Kruskal's algorithm on the other hand sorts all vertices by increasing weights. Then it always selects the next smallest edge that it can be added to the current tree without introducing a cycle. Should this not be the case, the edge is discarded. The algorithm terminates when $n-1$ edges have been selected.

3.3 Shortest Paths

There are different algorithms to find shortest paths in graphs: either between two nodes, between one node and all others, or between all pairs of nodes.

Dijkstra's algorithm computes the shortest path between one node and all other nodes in a graph.

The algorithm keeps track of a distance $d(v)$ and a predecessor $pred(v)$ for each node v. Starting from s, we keep updating the distances and predecessors of neighbours of a node. If a node was updated, we mark it so its own neighbours will be visited again. We do this until no node is marked (no changes happened).

Algorithm: Prim

Input: Graph $G = (V, E)$ with weights.

Output: MST $T \subseteq E$.

1. Let $T \leftarrow \{\{v, w\}\}$ where $\{v, w\}$ is the cheapest edge.
2. Until $\bigcup T = V$ (all vertices reached):
   - Find cheapest edge $\{v, w\}$ that connects a new node ($|\{v, w\} \cap \bigcup T| = 1$).
   - Set $T \leftarrow T \cup \{\{v, w\}\}$.
3. Return T.

Algorithm: Kruskal

Input: Graph $G = (V, E)$ with weights.

Output: MST $T \subseteq E$.

1. Let $T \leftarrow \emptyset$.
2. Until $|T| = |V| - 1$:
   - Take next cheapest edge $\{u, v\} \notin T$.
   - If `is_acyclic`$(T \cup \{u, v\})$, then
     - $T \leftarrow T \cup \{\{u, v\}\}$.
3. Return T.

Algorithm: Dijkstra

Input: Weighted graph $G = (V, E)$ with weights $w(i, j) \in \mathbb{N}$, and a source $s \in V$.

Output: A predecessor function *pred* and a distance function d.

1. $d(s) \leftarrow 0$ and $d(j) = \infty \quad \forall j \in V \setminus \{s\}$.
2. Set $pred(s) \leftarrow 0$.
3. Set $L \leftarrow \{s\}$.
4. While $L \neq \emptyset$:
   a) Select $i \leftarrow \text{argmin}_{k \in L}\, d(k)$.
   b) Set $L \leftarrow L \setminus \{i\}$.
   c) Each $j \in N(i)$ with $d(j) > d(i) + w(i, j)$:
      - Set $d(j) \leftarrow d(i) + w(i, j)$,
      - Set $pred(j) \leftarrow i$,
      - Add $L \leftarrow L \cup \{j\}$.
5. Return *pred* and d.

4 Location Planning

Location planning is done on either *continuous* or *discrete* location spaces. Then there is the distinction into *single-commodity* and *multi-commodity location problems*. Finally, we can also have *single-echelon location problems* with flow only in one direction (as opposed to *two-echelon location problems*).

4.1 Qualitative Methods

Qualitative methods can be used if the number of solutions is discrete and small.

The *weighted scoring method* weighs a set of facilities V by m *location factors*. Each factor k has a weight $w_k \in (0,1)$, and each facility i has a score s_{ik}. The weights are assumed to sum up to 1.

Algorithm: Weighted scoring method

Input: A set V of facilities, and weights $w_1, \ldots, w_m$, and scores s_{ik}.

Output: Preferred site i^*.

1. For every site $i \in V$, calculate $r_i = \sum_{k=1}^{m} w_k s_{ik}$.
2. $i^* = \operatorname{argmax}_{i \in V} r_i$.

4.2 Continuous SESC

The *(continuous) single-echelon single-commodity (SESC)* location problem is to find the optimal location of a single facility in a two-dimensional Cartesian plane (L2-metric) with a single commodity that is to be supplied to a set of successor nodes.

We basically want to find a "central" point that has least overall distances where nodes can have different weights. Note that the transport cost c is completely irrelevant for the problem.

For the continuous SESC problem, there is an ugly analytical **and recursive** expression:

$$x^* = \frac{\sum_{i \in V} \left(\frac{d_i x_i}{\sqrt{(x_i - x^*)^2 + (y_i - y^*)^2}} \right)}{\sum_{i \in V} \left(\frac{d_i}{\sqrt{(x_i - x^*)^2 + (y_i - y^*)^2}} \right)}$$

$$y^* = \text{(analogous)}$$

We can heuristically approach this result by doing a fixpoint iteration called *Weiszfeld heuristic*: To calculate x^* (respectively y^*), start with the "average point", and then plug in the formulas above recursively until the error threshold ε is reached. For $\varepsilon = 0$, this heuristic is actually exact.

Definition: Continuous SESC

In the *single-echelon single-commodity location problem*, we have a *transport cost* c and for each node $i \in V$:

- coordinates (x_i, y_i),
- a demand d_i.

We want to find the optimal location of the new facility $(x, y) = \operatorname{argmin}_{(x,y) \in \mathbb{R}^2} f(x, y)$ where

$$f(x, y) = \sum_{i \in V} c \cdot d_i \cdot \left(\sqrt{(x_i - x)^2 + (y_i - y)^2} \right).$$

Algorithm: Weiszfeld heuristic

(SESC input and output)

1. Start with $x^0 = \frac{\sum_{i \in V} d_i x_i}{\sum_{i \in V} d_i}$ and analogously for y^0.
2. Iteratively compute $x^{(h+1)}$ ($y^{(h+1)}$) by

$$x^{(h+1)} = \frac{\sum_{i \in V} \left(\frac{d_i x_i}{\sqrt{(x_i - x^h)^2 + (y_i - y^h)^2}} \right)}{\sum_{i \in V} \left(\frac{d_i}{\sqrt{(x_i - x^h)^2 + (y_i - y^h)^2}} \right)}$$

until $f(x^{(h+1)}, y^{(h+1)}) - f(x^{(h)}, y^{(h)}) \leq \varepsilon$.

4.3 Discrete SESC

In the discrete *single-echelon single-commodity location problem (SESC)*, we work on a complete bipartite graph. We now also add a maximum facility output q_i for each facilities $i \in V_1$, because why not. We also add a production cost F_i for each facility i.

It is also possible to add a time horizon $1, \ldots, T$ and make facility outputs time-dependent (q_{it}); similar for demands.

Definition: Discrete SESC

In the *single-echelon single-commodity location problem*, we are given

- a complete directed bipartite graph $G = (V_1 \cup V_2, A)$, with facilities V_1 and clients V_2 and arcs $A = V_1 \times V_2$,
- a maximum output q_i for $i \in V_1$,
- a demand d_j for $i \in V_2$,
- a transport cost $C_{ij}(s)$ for $s \in \mathbb{N}$,
- a production cost $F_i(s)$ for $s \in \mathbb{N}$.

$$
\begin{aligned}
&\text{Variables: } s_{ij} \geq 0 && \forall\, i \in V_1, \forall\, j \in V_2 && \text{(delivery quantity from } i \text{ to } j\text{)} \\
&\qquad\quad u_i \geq 0 && \forall\, i \in V_1 && \text{(production at facility } i\text{)} \\
&\min \quad \sum_{i \in V_1} \sum_{j \in V_2} C_{ij}(s_{ij}) + \sum_{i \in V_1} F_i(u_i) \\
&\text{where} \quad \sum_{j \in V_2} s_{ij} = u_i && \forall\, i \in V_1 && \text{(every facility } i \text{ ships exactly all it produces)} \\
&\qquad\quad \sum_{i \in V_1} s_{ij} = d_j && \forall\, j \in V_2 && \text{(every client } j \text{ receives its demand)} \\
&\qquad\quad \sum_{j \in V_2} s_{ij} \leq q_i && \forall\, i \in V_1 && \text{(facility output limit)}
\end{aligned}
$$

An instance in this category is the *capacitated plant location problem*. The *simple plant location problem* does not have maximum outputs for facilities.

4.3.1 p-Median Model

The *p-median model* is a discrete SESC model with a few simplifications. We want to find exactly p facility locations and have a fixed cost $c_{ij} \geq 0$ for each facility $i \in V_1$ that supplies $j \in V_2$.

$$
\begin{aligned}
&\text{Variables: } x_{ij} \in \{\, 0, 1 \,\} && \forall\, i \in V_1, \forall\, j \in V_2 && (x_{ij} = 1 \text{ iff } i \text{ supplies } j) \\
&\qquad\quad y_i \in \{\, 0, 1 \,\} && \forall\, i \in V_1 && (y_i = 1 \text{ iff facility } i \text{ is opened}) \\
&\min \quad \sum_{i \in V_1} \sum_{j \in V_2} c_{ij} x_{ij} \\
&\text{where} \quad \sum_{i \in V_1} x_{ij} = 1 && \forall\, j \in V_2 && \text{(every client } j \text{ is connected to one facility)} \\
&\qquad\quad \sum_{j \in V_2} x_{ij} \leq |V_2| \cdot y_i && \forall\, i \in V_1 && \text{(facility } i \text{ can only serve if opened)} \\
&\qquad\quad \sum_{i \in V_1} y_i = p && && \text{(exactly } p \text{ facilities opened)}
\end{aligned}
$$

4.3.2 Demand Allocation Problem

In the *demand allocation problem*, we want to find out **average quantities** to be supplied for a fixed set of facilities $\overline{V_1}$.

$$\begin{aligned}
&\text{Variables: } 0 \leq x_{ij} \leq 1 && \forall\, i \in \overline{V_1}, \forall\, j \in V_2 && (x_{ij} \text{ is the average supply from } i \text{ to } j)\\
&\min && \sum_{i \in \overline{V_1}} \sum_{j \in V_2} c_{ij} \cdot x_{ij} + \sum_{i \in \overline{V_1}} f_i\\
&\text{where} && \sum_{i \in \overline{V_1}} x_{ij} = 1 && \forall\, j \in V_2 && \text{(averages sum to 1)}\\
& && \sum_{j \in V_2} d_j \cdot x_{ij} \leq q_i && \forall\, i \in \overline{V_1} && \text{(facility output limit)}
\end{aligned}$$

4.4 Location-Covering Problem

In the *location-covering problem*, we want to cover a set of users V_2 by a subset of facilities V_1 in a graph $G = (V_1 \cup V_2,\ A)$ with edges A. We minimise fixed costs of selected facilities $f_i\, \forall\, i \in V_1$, while having each user being served by at least one facility to which it is connected. This is expressed by boolean constants $a_{ij}\ \forall\, i \in V_1, j \in V_2$ which is 1 if the edge from i to j exists.

$$\begin{aligned}
&\text{Variables: } y_i \in \{\, 0,\ 1 \,\} && \forall\, i \in V_1 && (y_i = 1 \text{ iff facility } i \text{ is opened})\\
&\min && \sum_{i \in V_1} f_i \cdot y_i\\
&\text{where} && \sum_{i \in V_1} a_{ij} \cdot y_i \geq 1 && \forall\, j \in V_2 && \text{(at least one serving facility per client)}
\end{aligned}$$

4.4.1 LCP Heuristic

Algorithm: Location-Covering Problem Heuristic

Input: An LCP instance.

Output: A set of facilities to be opened.

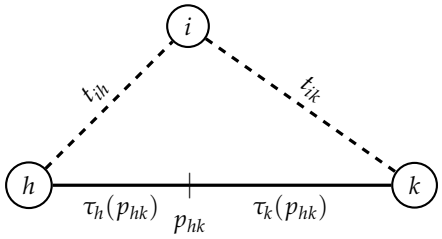
Define **open(i)**:
- Open i: Set $y_i = 1$.
- Mark i's neighbours as covered: Delete all constraints containing $a_{ij} \cdot y_i$ where $a_{ij} = 1$.

1. Construct the LCP LP above.
2. While there are still constraints left:
   a) For facility i that is free (where $f_i = 0$):
      - **open(i)**.
   b) For facility i that is not connected ($a_{ij} = 0 \ \forall j \in V_2$) and not free:
      - **close(i)**: Set $y_i = 0$.
   c) Find facility i that is cheapest per covered neighbour ($\operatorname{argmin}_i f_i/n_i$):
      (n_i is the number of constraints y_i appears in, i.e. the number of covered neighbours.)
      - **open(i)**.
3. Return $\{ i \in V_1 \mid y_i = 1 \}$.

4.4.2 p-Center Problem

In a *p-center problem*, we want to locate exactly p facilities so that the maximum travel time from a user to the closest facility is minimised. The p facilities are either on vertices or arcs/edges on a (potentially mixed) graph $G = (V, E, A)$. This problem is NP-hard for $p \geq 2$. If G is directed, then there is an optimal solution with the facilities located solely on vertices.

The 1-*center problem* is solved by *Hakimi's algorithm*. Here, we have travel times t_{ij} from node i to j. Then we have a distance function $\tau_h : P \to \mathbb{R}$ that maps a set of points $P = \{ p_{hk}, \dots \mid \{h,k\} \in E \}$ to a distance. These points lie somewhere on the edges (but may also lie directly on one vertex).



The travel time from one point p_{hk} on the edge h,k to a vertex i is given by $T_i(p_{hk})$. The algorithm first computes minimal travel times from every point to every vertex. Then, for every edge h,k, the best p_{hk} is selected. Finally, we choose the best of those over all edges.

Algorithm: Hakimi

Input:
- A graph $G = (V, E)$,
- distances $\tau_h : P \to \mathbb{R}$
- for points $P = \{ p_{hk}, \dots \mid \{h,k\} \in E \}$.

Output: A point p_{hk}.

1. **Minimal travel times** for every vertex $i \in V$ to any point on the edge $\{h,k\}$:
   $T_i : P \to \mathbb{R}$
   $T_i(p_{hk}) = \min \{ t_{ih} + \tau_h(p_{hk}),\ t_{ik} + \tau_k(p_{hk}) \}$
2. **Local center** for every edge, the point that minimises the maximum distance to all vertices:
   $$p^*_{hk} = \operatorname*{argmin}_{p_{hk} \in P} \max_{i \in V} T_i(p_{hk}).$$
3. **Find 1-center**, the best local center:
   $$p^* = \operatorname*{argmin}_{p_{hk} \in P} \max_{i \in V} T_i(p^*_{hk}).$$

5 Transportation Planning

5.1 Modes of Transport

Rail Transport Rail transport is inexpensive, but slow and unreliable. This is due to: It having a lower priority than passenger rail, direct connections being rare, and requiring a large quantity of goods in order to be profitable. It is mostly used to transport raw materials.

Road Transport Road transport is mostly used to transfer (semi-)finished products using trucks and can be separated into two categories, *truckload (TL)* and *less-than-truckload (LTL)*. TL is used to transfer directly from one location to another using the full capacity of the vehicle. LTL is used when single shipments are significantly smaller than vehicle capacity. Shipments are then transfered between vehicles covering different legs of the trip.

Air Transport Air transport is fast, but slowed by airport handling, and only competitive for long-haul transport. Planes have very low capacity. Thus they are mostly used for high-value goods over long distance.

Water Transport Water transport is mostly used to send bulk raw materials, represents 99% of the weight and 50% of the value of international trade. Also it's pretty cheap.

Pipeline Transport Pipeline transport only usable for specific goods, very slow, can provide a constant flow of goods, and is very reliable.

5.2 Vehicle Routing Problems

In *Vehicle Routing Problems (VRPs)*, we want to route m vehicles on a graph, starting and ending at a depot $o \in V$. Usually, we minimise costs of edges/arcs and number of vehicles used.

- **??**: *Node Routing Problem (NRP)*
  if $R = \emptyset$ (only target vertices)
  - *Traveling Salesperson Problem (TSP)*
    if $m = 1$ (one vehicle)
- **??**: *Arc Routing Problem (ARP)*
  if $U = \emptyset$ (only target arcs/edges)
  - *Rural Postman Problem (RPP)*
    if $m = 1$ (one vehicle)
    - *Chinese Postman Problem (CPP)*
      if G is complete

Definition: Vehicle Routing Problem

A instance of a VRP is defined on a graph $G = (V, A, E)$ with:

- V: a vertex set,
- A: a directed edge set,
- E: an undirected edge set,
- $o \in V$: the origin,
- $U \subseteq V$: the target vertices,
- $R \subseteq A \cup E$: the target arcs/edges,
- m: number of vehicles,
- (some costs to minimise).

Note: Often, the term *Vehicle Routing Problem* is used to refer to NRPs. ARPs are explicitly named.

5.3 Node Routing Problems (NRPs)

5.3.1 Traveling Salesperson Problem (TSP)

In the *traveling salesperson problem*, we want to find a route for one vehicle that visits all required vertices and the depot. The input graph G is not necessarily complete, so we consider the auxiliary complete graph $G' = (V', A')$ where $V' = U \cup \{0\}$ and $(i, j) \in A'$ with costs c_{ij}, the cost of the least-cost path between i and j.

G' satisfies the triangle inequality: $c_{ij} \leq c_{ik} + c_{kj} \quad \forall (i, j) \in A',\ k \in V',\ k \neq j, j.$

$$\begin{aligned}
&\text{Variables: } x_{ij} \in \{0,1\} && \forall (i,j) \in A' && (x_{ij} = 1 \text{ iff } (i,j) \text{ is part of the solution})\\
&\min \quad \sum_{(i,j)\in A'} c_{ij} \cdot x_{ij}\\
&\text{where} \quad \sum_{i \in V' \setminus \{j\}} x_{ij} = 1 && \forall j \in V' && (\text{every vertex } j \text{ has an incoming arc})\\
&\qquad \sum_{j \in V' \setminus \{i\}} x_{ij} = 1 && \forall i \in V' && (\text{every vertex } i \text{ has an outgoing arc})\\
&\qquad \sum_{i,j \in S} x_{ij} \leq |S| - 1 && \forall S \subsetneq V',\ |S| \geq 2 && (\text{subtour elimination constraints})
\end{aligned}$$

Asymmetric TSP (ATSP) as presented above is NP-hard. We can obtain a lower bound by solving the *assignment problem*, a relaxation of ATSP. The idea is to remove the subtour elimination constraints and the integrality of x_{ij} (so that only $x_{ij} \geq 0$). Additionally, set $c_{ii} = \infty \quad \forall i \in V'$ so that $x^*_{ii} = 0$. Then the optimal solution of the LP (solvable in polynomial time!) represents a collection of subtours in G' over all vertices. This gives us a lower bound for the optimal cost of ATSP.

5.3.2 Capacitated VRP (CVRP)

We refer to our Operations Research 1 Panikzettel which has a section on VRP.

5.3.3 VRP with Time Windows (VRPTW)

We refer to our Operations Research 1 Panikzettel which has a section on VRP.

5.4 VRP/NRP Heuristics

5.4.1 Patching Heuristic

The *patching heuristic* is useful for the ATSP, and the solutions it yields for the STSP are pretty bad. This is due to the fact that solving the STSP LP without SECs yields a lot of tours of length 2.

We solve the ATSP LP without the SECs. All edges in the optimal solution are on at exactly one tour. Then the two subtours with the largest number of nodes are merged so that the overall cost is least increased. Repeat until we have only one tour left.

5.4.2 Nearest Neighbour Heuristic

Select a start node. Always select nearest neighbour to continue the tour. If vehicle capacity is reached, end tour and start over with next vehicle.

5.4.3 Insertion Heuristics

Insertion heuristics add heuristically chosen nodes to the current subtour. One may start with a tour of three nodes, for example the largest triangle.

- *Nearest insertion*: Select vertex closest to any vertex already in the tour.
- *Cheapest insertion*: Greedily select vertex that is cheapest to insert.
- *Farthest insertion*: Select the farthest vertex from any vertex in the tour.
- *Random insertion*: Add a randomly chosen vertex.

Nearest and cheapest insertion are 2-approximations, while farthest and random insertion are only 6.5-approximations. On empirical examples however, farthest and random insertion perform better than the two others.

5.4.4 CVRP Construction Heuristics

There are two types of *construction heuristics* given m vehicles:

- Cluster first, route second: Partition vertices into m clusters and then solve the STSP on each induced complete subgraph (exactly or heuristically).
- Route first, cluster second: Compute a single route and then split into m feasible subroutes.

5.4.5 Savings Heuristic

The *savings heuristic* is for symmetric VRP only. Start by connecting every vertex by an individual tour to the origin. Then compute for every pair of tours the *savings*: what we would save by fusing that pair of tours compared to having two separate tours. After sorting all pairs of tours by decreasing savings, repeatedly fuse the pair which provides the most savings, until no more positive savings can be found.

Algorithm: Savings Heuristic

Input: A graph $G = (V, E, c)$ with $c : E \to \mathbb{Q}^+$.
Output: A set of tours C.

1. Compute the set C of n tours $C_i := (0, i, 0)$ for all $i \in V$.
2. For every pair of vertices v and w with $v \neq w$:
   - compute **potential savings** $s_{vw} = c_{0v} + c_{w0} - c_{vw}$.
3. Let L be the sorted list of savings that are **non-negative**.
4. While L is non-empty:
   a) Select the first feasible element (v, w) of L. (v, w) is feasible if
      - v is the **last vertex of its tour**,
      - and w **the first of its own** (or vice versa) and,
      - if applicable, if the current vehicle **meets the capacity requirements**.
   b) Remove all unfeasible edges from L.
   c) Remove $(v, 0)$ and $(0, w)$ and fuse the tours C_v and C_w by adding the edge (v, w).
5. Return C.

5.4.6 k-opt Local Search Heuristic

Given a heuristic solution to a node routing problem, the *k-opt heuristic* can be used to further improve the solution. The idea is to explore all similar solutions and choose a better one. Here, we choose the best solution among all those where k edges are substituted by k other edges. Repeatedly do this until the solution cannot be improved in this way. Then we have a local optimum. The heuristic has exponential runtime in k and thus usually $k = 2$ or $k = 3$.

5.5 Arc Routing Problems

In *arc routing problems (ARP)*, one has to traverse a (sub-)set of arcs in a graph at least once. ARPs are classified by two aspects.

Link type and weight **(U)** undirected, **(D)** directed, **(M)** mixed (i.e. both directed and undirected), and **(W)** windy, where edges always have a reverse edge, but the cost is not necessarily symmetric ($c_{ij} \neq c_{ji}$).

Connection of links The *Chinese postman problem (CPP)* is defined on connected graphs. Otherwise we have a *pural postman problem (RPP)*.

	Chinese Postman	Rural Postman
U	UCPP (P)	URPP (NP)
D	DCPP (P)	DRPP (NP)
M	MCPP (NP)	MRPP (NP)
W	WCPP (NP)	WRPP (NP)

(Only directed and undirected Chinese Postman Problems are solvable in polynomial time.)

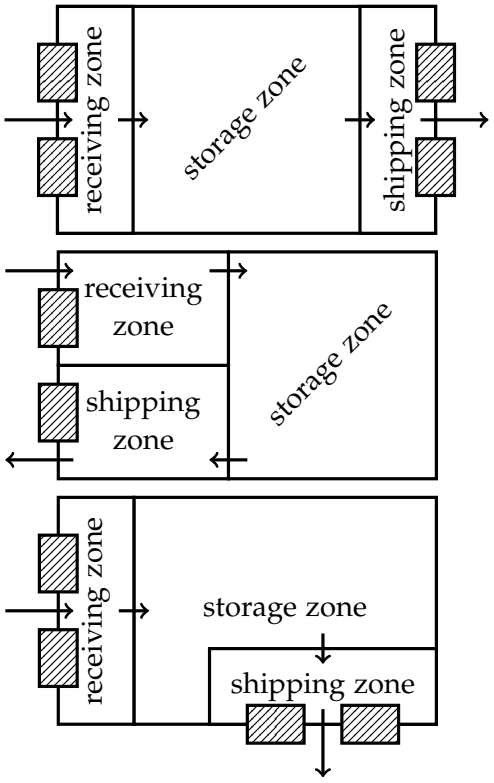
6 Warehouse Planning

6.1 Warehouse Layouts

Here, we go through three different types of warehouse layouts, which are illustrated on the right. The first one is the *flow-through layout*, with one end of the warehouse having entry points, the other having exit points, and the storage lying inbetween. This layout is useful if most load units require the same operations.

The *U-layout*, illustrated in the middle, describes a situation in which entry and exit points are on the same side of the warehouse. It is useful for low material flows and when needing flexibility with regards to the assignment of shipping/receiving space.

The *hybrid layout* has entry and exit points on different, though non-opposed sides of the building.



6.2 Picker Routing

The following descriptions of heuristics for efficient package picking will seem rather vague and informal. The author wants to apologise for this, however the material provided during the course of the lecture did not allow for a more precise write-up.

6.2.1 S-Shaped routing

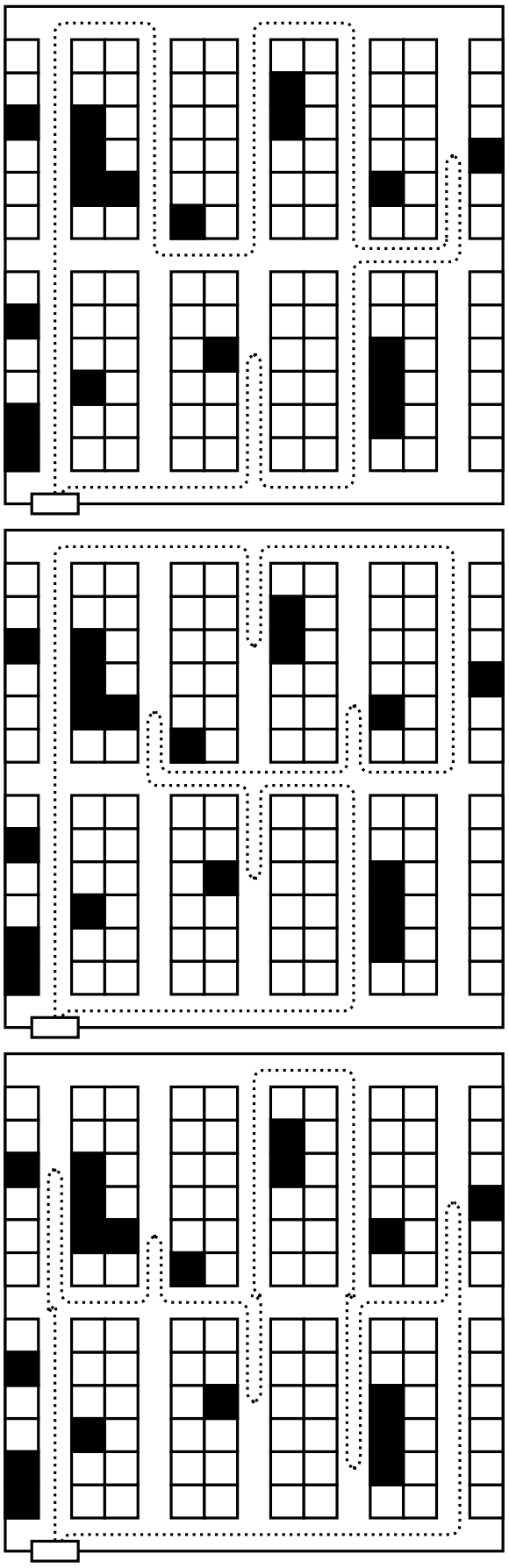
1. From the entrance, move to the upmost row that contains packages.
2. move down that row in an S-shape, entering only aisles that contain packages, traversing them completely.
3. after having picked up the last package of the row, immediately rejoin the lower aisle.
4. There, traverse the row in the same manner, choose direction freely.
5. Continue until last row, then return to entrance.

6.2.2 Largest Gap Heuristic

1. For every aisle, compute the largest gap, that is, the largest distance between two packages or a package and an entrance such that no other package lies between. This gap shall not be crossed except if we reach the end of a row.
2. Move to the upmost row that contains packages. Walk along the aisles and pick up all packages that can be reached without crossing the largest gap. Return when reaching the largest gap.
3. After reaching the last aisle that contains a package, cross it completely and go in reverse, picking up packages that haven't been picked up.

6.2.3 Aisle-by-Aisle Heuristic

1. Starting from the leftmost aisle containing a package, traverse every aisle from left to right.
2. Select crossaisle for traversal to minimise travel distance.



From top to bottom: S-shaped routing, Largest Gap Heuristic and Aisle-by-Aisle Heuristic.