Section 3: Decision-Based Approaches

Section 3.1: Decision Trees

A few introductory words

- In Section 2, we considered variations of local search.
- Now, in Section 3, we will consider a range of variants to a completely different approach.
- *Idea*:
  - ◇ In most algorithmic problems, each feasible solution can be regarded as the result of a sequence of decisions.
  - ◇ The various options for the individual decisions form a rooted tree, the *decision tree*.

    $\longrightarrow$ Examples and details coming soon.

Generic example: features

- From Section 2.3.2 recall that, in many (if not most) algorithmic problems,
  - ◇ the feasible solutions may be identified with the subsets of a ground set of *features*,
  - ◇ which may often be regarded as the *dimensions* of the underlying ground set.

  $\longrightarrow$ Feature-based problem definitions and algorithms.
- Each feature in the ground set naturally induces one decision: whether or not it shall be a member of the feasible solution.
- Thus, every member of the ground set may be determined by a sequence of "yes" and "no" answers (one for each feature).

Independence systems:

- This is an important special case of feature-based problem definitions as defined in Section 3.2 (later on, we will see why).

- *Concrete specialization*:

  For each set S_1 of features that is a feasible solution and for each subset $S_2 \subseteq S_1$, S_2 is a feasible solution, too.

Remarks

- Obviously, the solution space is empty if, and only if, "$\emptyset$" is not a member of the solution space.
- For convenience, the case that the solution space is empty is typically excluded in the literature on independence systems.

Examples of independence systems:

- The matchings in an undirected graph (cf. Section 1.2).
- The cycle-free edge sets of an undirected graph.
- The sets of pairwise non–connected nodes in an undirected graph (called *independent sets* of nodes in the literature).
- The independent subsets of a finite set of vectors from some vector space.
- The sets of disjoint paths on given points $p_1, \ldots, p_n$ in the plane such that each point is met at most once.

  $\longrightarrow$ The proper subsets of the TSP tours on $p_1, \ldots, p_n$.
- All TSP tours on $p_1, \ldots, p_n$ plus all subsets of all of these tours.

Note:

Since "$\emptyset$" is a member of every (non-empty) independence system, we must adopt the convention that "$\emptyset$" is a feasible solution in each of the examples on the last slide (and in any other example of independence systems as well).

Generic and specific decisions

- To avoid confusion, we will distinguish the generic from the specific kind of "decision".
- *Example*:
  - ◇ *Generic*: whether to use cars or trains.
  - ◇ *Specific*: whether to take a car or a train for a trip to be made from X to Y on day D.
- *Example* feature-based:
  - ◇ *Generic*: whether to select feature i.
  - ◇ *Specific*: whether to select feature i after $X \subseteq \{1, \ldots, i-1\}$ has been selected and $\{1, \ldots, i-1\} \setminus X$ rejected and before a decision is made for any feature $\{i+1, \ldots, n\}$.

Decision trees:

- A decision tree is associated with an instance of an algorithmic problem.
- Each node in the tree stands for a specific decision.
- The arcs that leave a node stand for the options in that specific decision.

  $\longrightarrow$ An arbitrary number of options incl. 0 ("dead end").
- Consequently:
  - ◇ the leaves stand for the feasible (and infeasible) solutions and
  - ◇ the (unique) path from the root to a leaf corresponds to the sequence of specific decisions that determines the solution corresponding to this leaf.

Important special case: feature-based

- In the feature–based case, a natural (though not unique) definition of the decision tree would be the following:

  Every node on height level $i - 1$ stands for the specific decision to select or reject feature i (root: height level 0).

- This yields a *full* binary tree, that is:

  ◇ All leaves are on the same level n (with n = number of features).

  ◇ Any other node has exactly two children (one for "select" and one for "reject").

Alternative interpretation of decision trees:

- The options in a decision partition the solution space into subsets.
- Thus, a branch of the decision tree may be identified with the intersection of the subsets induced by the options along this branch (that is, the arcs of this branch).
- *Example* feature–based problem definitions again: For each feature, the decision whether or not to take this feature partitions the solution space into
  - ◇ those solutions in which this feature is *sel*ected and
  - ◇ those solutions in which this feature is *rej*ected.
- This is an example of the typical case that the options contradict each other like "select" and "reject".

  $\longrightarrow$ The solution space is partitioned into *disjoint* subsets.

Alternative interpretation (cont'd):

- Equivalently, we could say that each node corresponds to a certain subset of the solution space.
- *Example* feature-based: the set of all solutions that
  - ◇ contain the features selected so far and
  - ◇ do not contain the features rejected so far.
- A leaf then corresponds to a singleton.

  $\longrightarrow$ In perfect correspondence to the original interpretation of leaves as the elements of the solution space.

Exploring decision trees:

- Typically, the number of solutions is exponential in the number of decisions.
- Thus, a decision tree for an instance is way too large to be explored exhaustively.
- In principle, there are two different strategies to overcome this problem:
  - ◇ The exploration is *heuristically* restricted to a selected part of the tree.
  - ◇ The exploration is *potentially exhaustive*.

What does that mean?

- Exploration "heuristically restricted":
  - ◇ Deliberately skip parts of the tree (subtrees).
    - ⟶ The search for a feasible (not to mention optimal) solution may fail even if the solution space is non–empty.
- Exploration "potentially exhaustive":
  - ◇ For each node, we try to construct a certificate that there is no feasible (or optimal) solution among the leaves of the node's subtree.
  - ◇ If we succeed, the subtree is excluded from the exploration as a whole.

Section 3.2: The Greedy Algorithm

What is the greedy algorithm?

- Rough answer: the simplest imaginable strategy for a heuristic restriction of the search space.
  - ◇ Explore exactly one branch of the decision tree.
  - ◇ At each node, choose the option that looks "most promising" at this moment (without foresight = "greedily").
- Due to the second point, this strategy is called the *greedy algorithm* in the literature.
- Sometimes, this algorithmic scheme is also called the A^*–algorithm.
- However, both terms, "greedy algorithm" and "A^*–algorithm", are often used in a more restrictive way: for certain special cases of this general algorithmic scheme only.

Example: TSP

- In the following, we will consider an instance with n points.
- We will define yet another decision tree for the TSP.
- All leaves of this alternative decision tree are on level no. n.
- For $k \in \{0, \ldots, n\}$:
  - ◇ Each node v on the k–th level stands for a set (i_1^v, j_1^v), (i_2^v, j_2^v), $\ldots$, (i_k^v, j_k^v) of k pairwise different pairs.
  - ◇ In turn, each sequence of k pairwise different pairs is represented by some node on the k–th level.

  $\longrightarrow$ One–to–one correspondence.

Example TSP (cont'd)

- For $k \in \{0, \ldots, n-1\}$, each node v on the level no. k has as many as $n \cdot (n-1) - k$ immediate successors:

  Exactly one for each pair (i, j), $i, j \in \{1, \ldots, n\}$ such that $(i, j) \notin \{(i_1^v, j_1^v), (i_2^v, j_2^v), \ldots, (i_k^v, j_k^v)\}$.

  $\longrightarrow$ Each arc leaving v stands for one option how to extend the set of pairs of v by exactly one more pair.

- Thus, consider the node v representing $\{(i_1^v, j_1^v), (i_2^v, j_2^v), \ldots, (i_k^v, j_k^v)\}$ and the arc leaving v that represents the selection of (i, j).

- Such an arc is *eligible* if $\{(i_1^v, j_1^v), \ldots, (i_k^v, j_k^v), (i, j)\}$ can be extended to a round tour on all points, $\{1, \ldots, n\}$.

Example: TSP (cont'd)

- Extensible to a round tour:
  - ◇ A pair (i, j) is definitely not eligible if $i = i^v_\ell$ or $j = j^v_\ell$ for some $\ell \in \{1, \ldots, k\}$.
  - ◇ A pair (i, j) is definitely not eligible, either, if

$$(i^v_1, j^v_1), \ldots, (i^v_k, j^v_k), (i, j)$$

    contains a subtour.
  - ◇ Obviously, in all other cases, (i, j) is eligible.
- Among the eligible arcs, the pair (i, j) with the minimal distance value is chosen by the greedy algorithm.

Greedy algorithm for machine scheduling

- *Input*:
  - ◇ *Jobs* $1, \ldots, N$ with job durations $d_1, \ldots, d_N$.
  - ◇ A number M of *machines* (typically, $M \ll N$).
- *Output*: a machine $m_i \in \{1, \ldots, M\}$ and a *start time* $t_i \geq 0$ for each item $i \in \{1, \ldots, N\}$.
- *Constraint*: disjoint execution times for $i, j \in \{1, \ldots, N\}$ if $m_i = m_j$, that is, either $t_i + d_i \leq t_j$ or $t_j + d_j \leq t_i$.
- *Minimize*: total completion time, $\max\{\, t_i + d_i \mid i = 1, \ldots, N \,\}$.

Greedy procedure

1. For $j = 1, \ldots, M$: $\max_j := 0$.
2. For $i = 1, \ldots, N$ (in this order) do:
   (a) Let $j \in \{1, \ldots, M\}$ such that $\max_j$ is minimal.
   (b) Set $m_i := j$ and $t_i := \max_j$.
   (c) Increase $\max_j$ by d_i.

Analysis of the algorithm

- In general, the total completion time produced by the algorithm is not optimal.
- However, on the next few slides, we will prove that it is not more than *twice* the optimal total completion time.
- In practice, the difference to the optimal total completion time is much smaller except for a few "pathological" instances.

Proof of "at most twice"

- Let $(m_1, t_1), \ldots, (m_N, t_N)$ be a schedule whose total completion time is strictly more than twice the optimal one.
- We have to show that $(m_1, t_1), \ldots, (m_N, t_N)$ cannot have been constructed by our algorithm.
- Let $(\tilde{m}_1, \tilde{t}_1), \ldots, (\tilde{m}_N, \tilde{t}_n)$ be an optimal schedule.
- For $j \in \{1, \ldots, M\}$, let $\tilde{S}_j := \sum_{\substack{i=1 \\ \tilde{m}_i = j}}^{N} d_i$ and $S_j := \sum_{\substack{i=1 \\ m_i = j}}^{N} d_i$.
- Let opt denote the optimal total completion time.
- Obviously, the optimal total completion time cannot be smaller than the average workload of all machines:

$$A := \frac{1}{M} \cdot \sum_{i=1}^{N} d_i \leq opt\,.$$

Proof (cont'd)

- Since the total completion time of $(m_1, t_1), \ldots, (m_N, t_N)$ is at least twice the optimal value, there is a machine $j_1 \in \{1, \ldots, M\}$ such that $S_{j_1} \geq 2 \cdot opt$.
- In particular, it is $S_{j_1} \geq 2A > A$.
- Since A is the average of the values of all S_j, there is another machine $j_2 \in \{1, \ldots, M\}$ such that $S_{j_2} < A$.
- For each $i \in \{1, \ldots, M\}$, it must be $d_i \leq opt$, because $d_i \leq \tilde{S}_{\tilde{m}_i} \leq \max\{\tilde{S}_j \,|\, j = 1, \ldots, M\} = opt$.
- Due to $S_{j_1} \geq 2 \cdot opt$ and $opt \geq A > S_{j_2}$, it is $S_{j_1} - S_{j_2} > opt$.

Proof (cont'd)

- Let $i \in \{1, \ldots, M\}$ such that $m_i = j_1$ and t_i is maximal among all of the items placed on machine j_1.

  $\longrightarrow$ Item i is the last item placed on machine j_1, so it is $t_i = S_{j_1} - d_i$.

- Since $d_i \leq opt$ and $S_{j_1} - S_{j_2} > opt$, it is $t_i > S_{j_2} + opt - d_i \geq S_{j_2}$.

- Obviously, S_{j_2} is the last value assumed by variable $\max_{j_2}$, so $\max_{j_2} \leq S_{j_2}$ at any time during the algorithm.

- In particular, it was $\max_{j_2} \leq S_{j_2} < t_i = \max_{j_1}$ at the moment when item i was placed on machine m_i.

- However, this contradicts the procedure of our algorithm: item i had been placed on machine j_2 rather than j_1.

Definition: approximation

In the literature on optimization algorithms, we usually speak of an x-*approximative* algorithm if the value of a solution is never

- more than x times the optimal value in case of a m*in*imization problem,
- less than $1/x$ times the optimal value in case of a m*ax*imization problem.

Approximative and exact greedy

- Machine scheduling is but an example of "approximatively accurate" algorithms.
- For a variety of fundamental, simple problems like that, there are algorithms that provably miss the optimum by not more than a constant factor.
- In most cases in the literature, this is a factor of 2 or worse.
- However, analogously to what we said above for machine scheduling, the observed accuracy in practice is typically much better than the worst-case bound.

Optimal results from the greedy algorithm?

- In Section 2, we have seen that the simple local-search strategy is guaranteed to deliver an optimal solution if the problem fulfills a certain structural property (exact neighborhood).
- So an interesting question is whether the greedy algorithm provably delivers optimal solutions if some structural property is fulfilled.
- The rest of Section 3.2 is devoted to an exhaustive characterization of this for the case of independence systems.
- Unfortunately, the structure and the optimality proof are way more complex than in the case of local search...

Greedy on independence systems:

- From Slide no. 184 recall the notion of independence systems.
- Fundamental optimization problem:
  - ◇ *Input*:
    - ▷ an independence system on a feature set $F = \{f_1, \ldots, f_n\}$,
    - ▷ a weighting $c : F \longrightarrow \mathbb{R}^+$.
  - ◇ *Feasible outputs*: the independent sets $I \subseteq F$.
  - ◇ *Objective*: maximizing $\sum_{f \in I} c[f]$.
- W.l.o.g. assume $c[f_1] \geq c[f_2] \geq \cdots \geq c[f_n]$.

Natural decision tree for greedy on independence systems

- All leaves are on level no. n.
- For $k \in \{0, \ldots, n-1\}$, a node on the level no. k has exactly two immediate successors.
- Each node v on the level no. k stands for a subset of $\{f_1, \ldots, f_k\}$.
- In particular, the root (level 0) stands for $\emptyset$.
- The two immediate successors of a node v stand for the options to insert or not to insert f_{k+1} into the subset of v.

Reduction:

If all inclusion–maximal independent sets happen to have the same cardinality, the problem of

*min*imizing $\sum\{c\,[f] \mid f \in I\}$ over all
inclusion–maximal independent sets I (all $c[\cdot] > 0$)

can be reduced to the m*ax*imization problem
from the last but one slide.

Details of the reduction:

- Let $m := -\min\{c[f] \mid f \in F\} + 1$.
- for all $f \in F$, take $\tilde{c}[f] := -c[f] + m$ instead of $c[f]$.

Analysis of the reduction:

- To see that this reduction is correct, consider the problem of *max*imizing $\sum\{\tilde{c}\,[f] \mid f \in I\}$ over all independent sets I.
- Since $\tilde{c}[f] > 0$ for all $f \in F$, all optimal solutions are inclusion–maximal independent sets.
- Therefore, each optimal solution to this maximization problem is a feasible solution to the original minimization problem as well.

Analysis of the reduction (cont'd):

- Even better, an optimal solution to this maximization problem is also an *optimal* solution to the original minimization problem, because:
  - ◇ Recall that all inclusion-maximal independent sets have the same size k, say.
  - ◇ For every inclusion-maximal set S, this means $\tilde{c}[S] = -c[S] + k \cdot m$.
  - ◇ Therefore, the "lighter" a set is with respect to $c[\cdot]$, the heavier it is with respect to $\tilde{c}[\cdot]$.
- In summary, we can solve the maximization problem to obtain a solution to the minimization problem from the last slide (which was restricted to *inclusion-maximal* independent sets).

Example I: TSP revisited

- For an instance of the TSP, consider the set P of pairs (i, j), $i, j \in \{1, \ldots, n\}$, such that P can be extended to a feasible solution.

  $\longrightarrow$ Formally, "extended" must include the case that nothing is to be done ("empty extension").

- *Obvious facts*:
  - ◇ The set of all of these sets of pairs forms an independence system.
  - ◇ The round tours are exactly the inclusion–maximal elements of this independence system.
  - ◇ All round tours contain exactly n pairs.
- In summary, the problem of finding a round tour of minimal length can be reduced to the problem of finding an independent set of maximal weight (according to Slide 211).

Example II: Minimum Spanning Tree (MST)

- Recall the MST from Section 1.1, Example 2:
  - ◇ *Input*: An undirected, connected graph $G = (V, E)$ and edge lengths $c : E \longrightarrow \mathbb{R}$.
  - ◇ *Feasible output*: A *spanning tree*, that is, a connected, cycle–free subgraph $T = (V, E_T)$ of G on the whole set of nodes of G.
  - ◇ *Objective*: Minimizing $\sum_{e \in E_T} c[e]$.
- In the following, we will identify spanning trees and other subgraphs of G with their edge sets.

  $\longrightarrow$ A spanning tree is then a cycle–free edge set such that each node is hit by at least one edge in this set.

Simple observations

- The spanning trees are exactly the inclusion-maximal cycle-free subsets of E.

- The cycle-free subsets of E form an independence system.

Definition:

For an undirected graph $G = (V, E)$ and $E' \subseteq E$, the *connected components* of E' in G are the inclusion-maximal subsets $V' \subseteq V$ such that for any two nodes $v, w \in V'$ there is a path in G solely consisting of edges in E'.

Easy consequences:

- If a node is not incident to any edge in E', it forms a connected component of its own.
- If E' is a spanning tree, V is the (only) connected component.
- A cycle-free edge set E' consists of one spanning tree for each of its connected components.

Crucial fact:

All spanning trees contain exactly $\#V - 1$ edges.

Proof of the crucial fact: on the next slide.

Significance of the crucial fact:

The problem of finding a minimal spanning tree may be reduced to the problem of finding a maximal-weight independent set (=cycle-free edge set).

Proof of the crucial fact

- By induction on $\#V > 0$.
- For $\#V = 1$, the claim is obvious.
- So consider a tree $T = (V, E)$ with $\#V > 1$ nodes.
- As T is cycle-free, there is an inclusion-maximal path p in T.
- Since p is inclusion-maximal, the two endnodes of p must be leaves (=must have degree one).
- Removing one of these two nodes along with the incident edge yields a tree T' with $\#V - 1$ nodes and $\#E - 1$ edges.
- Therefore, the induction hypothesis applies to T'.
- In summary, this proves the claim for T.

Exchange property:

- Let
  - $F = \{f_1, \ldots, f_n\}$ be a set of features and
  - $\mathcal{I}$ an independence system on F.
- We will say that this independence system has the *exchange property* if the following holds:
  - Let F_1, $F_2 \in \mathcal{I}$ be arbitrarily chosen such that $\#F_1 > \#F_2$.
  - Then there must be an element $f \in F_1 \setminus F_2$ such that $F_2 \cup \{f\} \in \mathcal{I}$.

Example I: MST has the exchange property

- Let $G = (V, E)$ be an undirected graph.
- Consider two cycle-free subsets of E, F_1 and F_2, and suppose $\#F_1 > \#F_2$.
- Let $C_1, \ldots, C_k$ denote the connected components of F_2.
- For $j \in \{1, \ldots, k\}$, let $F_{1j} \subseteq F_1$ and $F_{2j} \subseteq F_2$ denote the sets of edges in F_1 and F_2, respectively, that connect two nodes in C_j.
- According to Slide 217, each F_{2j} is a spanning tree in C_j.
- Of course, each F_{1j} is cycle-free, $j \in \{1, \ldots, k\}$.

MST and the exchange property (cont'd)

- Since the spanning trees are the cycle-free sets of maximal size, we have $\#F_{1j} \leq \#F_{2j}$ for all $j \in \{1, \ldots, k\}$.
- Thus, the assumption $\#F_1 > \#F_2$ implies that there is at least one edge $e \in F_1$ that is not in any of $C_1, \ldots, C_k$.
- In particular, this implies $e \notin F_2$, so $e \in F_1 \setminus F_2$.
- Since e links two components C_j to each other, inserting e in F_2 does not close a cycle.

Example II: linearly independent vectors

- Consider some vector space V over some field.
- Let $v_1, \ldots, v_n \in V$.
- *Observation*:

  The set of linearly independent subsets of $\{v_1, \ldots, v_n\}$ forms an independence system.

- Now let $F_1 := \{u_1, \ldots, u_k\} \subseteq \{v_1, \ldots, v_n\}$ and $F_2 := \{w_1, \ldots, w_\ell\} \subseteq \{v_1, \ldots, v_n\}$, $k > \ell$.
- Since F_1 and F_2 are linearly independent,
  - ◇ the span of F_1 has dimension k and
  - ◇ the span of F_2 has dimension $k - 1$.

Linearly independent vectors (cont'd)

- In particular, at least of of the vectors $u_1, \ldots, u_k$ is not in the span of F_2.
- This vector can be added to F_2 without destroying linear independence of F_2.
- In summary, this independence system has the exchange property, too.

Remark on Examples I and II:

- Example I (MST) may be viewed as a special case of Example II (independent vectors).
- In fact, consider the *incidence matrix* I_G of an undirected graph $G = (V, E)$:
  - ◇ There is a row for each node $v \in V$.
  - ◇ There is a column for each edge $e \in E$.
  - ◇ All entries are 0 or 1.
  - ◇ For $v \in V$ and $e \in E$, it is $I_G[v, e] = 1$ if, and only if, e is incident to v.

Claim:

A set of columns of I_G (viewed as vectors) is linearly independent.

$$\Longleftrightarrow$$

The set of edges corresponding to these columns is cycle–free.

Proof: left as an exercise.

Examples lacking the exchange property

- TSP: the set of all subsets of tours (incl. tours as a whole).
- The matchings in a graph.
- The Steiner tree problem (Section 1.1, Example 2).

$\longrightarrow$ Left as an exercise.

Terminology:

- An independence system that has the exchange property is called a *matroid* in the literature.

- Origin of the name:

  The independent sets of columns of a matrix form a matroid (cf. Slide no. 223).

  $\longrightarrow$ The independent subsets of finite vector sets are called *matrix matroids* in the literature.

Theorem:

The following two statements about an independence system are equivalent:

- The independence system is a matroid.
- The greedy algorithm from Slide no. 209 computes an optimal solution for *every*(!) choice of the objective function c.

Proof: on the next ten slides.

Remark: if the independence system is not a matroid, the greedy algorithm will anyway compute an optimal solution for *some* choices of c (e.g. for $c \equiv 0$ in any case).

Proof of the theorem, direction 1 of the equivalence:

- Suppose that the independence system is a matroid.
- In direction 1, we have to show that the greedy algorithm computes an optimal solution for any objective function c.
- Let F denote the set of features.
- The proof of this direction is broken down into two steps:
  - ◇ First an auxiliary claim is proved:

    For all $F' \subseteq F$ and $I_1, I_2 \subseteq F'$ two inclusion–maximal independent subsets of F', we have $\#I_1 = \#I_2$.
  - ◇ Then we prove the main claim from this auxiliary claim.

Proof of the auxiliary claim:

- Let I_1, $I_2 \subseteq F'$ be two inclusion–maximal independent subsets of $F' \subseteq F$.
- Suppose $\#I_1 \neq \#I_2$ for a contradiction, say $\#I_1 > \#I_2$.
- Since, by assumption, the independence system is a matroid, there is $f \in I_1 \setminus I_2$ such that $I_2 \cup \{f\}$ is independent.
- This immediately contradicts the assumption that I_2 is an inclusion-maximal independent set.

Proof of the theorem, direction 1 (cont'd):

- For the first direction, it remains to show:
  - ◇ If an independence system satisfies the statement of the auxiliary claim,
  - ◇ then the greedy algorithm constructs an optimal solution for every objective function c.
- Let $I = \{f_1, \ldots, f_k\}$ be the solution constructed by the greedy algorithm.
- Suppose for a contradiction that there is an independent set $I' = \{f'_1, \ldots, f'_\ell\}$ such that

$$\sum_{i=1}^{k} c[f_i] < \sum_{j=1}^{\ell} c[f'_j] \,.$$

- Since $c[\cdot] \geq 0$, we may assume that I' is an inclusion-maximal independent set.

Proof of the theorem, direction 1 (cont'd):

- W.l.o.g. assume
  - $c[f_1] \geq c[f_2] \geq \cdots \geq c[f_k]$ and
  - $c[f'_1] \geq c[f'_2] \geq \cdots \geq c[f'_\ell]$.
- Since $c \geq 0$, $I = \{f_1, \ldots, f_k\}$ is inclusion–maximal.
- As $I' = \{f'_1, \ldots, f'_\ell\}$ is inclusion-maximal, too, the auxiliary claim implies $k = \ell$.
- Therefore, it suffices to show $c[f_i] \geq c[f'_i]$ for all $i \in \{1, \ldots, k\}$.
- We will prove this by induction on $i = 1, 2, 3, \ldots$.
- For $i = 1$, the claim follows immediately from the nature of the greedy strategy: the first element to be chosen is always the overall maximal independent singleton.
- So consider the case $i > 1$ in the following.

Proof of the theorem, direction 1 (cont'd):

- Induction hypothesis: $c[f_j] \geq c[f'_j]$ for all $j \in \{1, \ldots, i-1\}$.
- Let $F' = \{f \in F \mid c[f] \geq c[f'_i]\}$.
- First we have to understand that $\{f_1, \ldots, f_{i-1}\}$ is an inclusion-maximal independent subset of F':
  - ◇ Clearly, $\{f_1, \ldots, f_{i-1}\}$ is independent and a subset of F' due to $c[f_j] \geq c[f'_j] \geq c[f'_i]$ for all $j \in \{1, \ldots, i-1\}$.
  - ◇ To see that $\{f_1, \ldots, f_{i-1}\}$ is inclusion–maximal independent in F', assume for a contradiction that $\{f_1, \ldots, f_{i-1}, f\}$ is independent for some $f \in F' \setminus \{f_1, \ldots, f_{i-1}\}$:

    Since $c[f] \geq c[f'_i] > c[f_i]$, f had then been chosen instead of f_k (INDEX!???) by the greedy algorithm.

    $\longrightarrow$ Contradiction!

Proof of the theorem, direction 1 (cont'd):

- On the other hand, due to $c[f'_1] \geq c[f'_2] \geq \cdots \geq c[f'_i]$, it is $\{f'_1, \ldots, f'_i\} \subseteq F'$ as well.
- Let $I'' \supseteq \{f'_1, \ldots, f'_i\}$ be an inclusion-maximal independent subset of F' that contains $\{f'_1, \ldots, f'_i\}$ ($I'' = \{f'_1, \ldots, f'_i\}$ possible).

  $\longrightarrow$ Exists since $\{f'_1, \ldots, f'_i\} \subseteq \{f'_1, \ldots, f'_\ell\}$ and $\{f'_1, \ldots, f'_\ell\}$ is independent by definition.
- Then we have $\#I'' \geq i > \#\{f_1, \ldots, f_{i-1}\}$.
- This contradicts the auxiliary claim because $\{f_1, \ldots, f_{i-1}\}$ is an inclusion-maximal independent subset of F' (cf. last slide).

Proof of the theorem, direction 2:

- In this direction,
  - ◇ we assume that the independence system is not a matroid
  - ◇ and have to show that the greedy algorithm will fail to produce an optimal solution for at least one choice of c.
- So suppose there are independent I_1, I_2 such that $\#I_1 > \#I_2$ but $I_2 \cup \{f\} \notin \mathcal{I}$ is *not* independent for any $f \in I_1 \setminus I_2$.
- We will construct a particular objective function c such that the greedy algorithm will fail to produce an optimal solution for c.

Proof of the theorem, direction 2 (cont'd):

- Idea: define c such that
  - $\diamond$ the elements of I_2 have higher c-values than any other elements of F,
  - $\diamond$ but I_1 has a higher total value than I_2.

  $\longrightarrow$ The choice of c must be "well balanced" to satisfy these two conflicting goals.
- Implementation of this idea:

$$c[f] := \begin{cases} \#(I_2 \setminus I_1) + 2 & \text{for } f \in I_1 \setminus I_2\,. \\ \#(I_2 \setminus I_1) + \#I_1 - \#I_2 + 2 & \text{for } f \in I_2\,. \\ 0 & \text{otherwise}\,. \end{cases}$$

Proof of the theorem, direction 2 (cont'd):

- Since $\#I_1 > \#I_2$, the elements of I_2 indeed have higher c-values than any elements of $F \setminus I_2$.

  $\longrightarrow$ The greedy algorithm is forced to produce I_2 or a superset of I_2.

- By our choice of I_1 and I_2, $I_2 \cup \{f\}$ is not independent for any $f \in I_1 \setminus I_2$.

  $\longrightarrow$ The value of the solution produced by the greedy algorithm is just the total weight of I_2, that is,

$$c(I_2) := \#I_2 \cdot (\#(I_2 \setminus I_1) + \#I_1 - \#I_2 + 2)$$

Proof of the theorem, direction 2 (cont'd):

- However, the value of the independent set I_1 is

$$c(I_1) := \#(I_1 \cap I_2) \cdot (\#(I_2 \setminus I_1) + \#I_1 - \#I_2 + 2) + \#(I_1 \setminus I_2) \cdot (\#(I_2 \setminus I_1) + 2)\,.$$

- It is not too hard to see that $c(I_1) > c(I_2)$.

  $\longrightarrow$ Left as an exercise.

- This concludes the proof of direction 2 and thus the proof of the theorem.

Remarks:

- For the example of linearly independent vectors (cf. Slide 223), the auxiliary claim is an obvious consequence of standard facts from basic linear algebra (*Steinitz Exchange Lemma*):
  - $\diamond$ F' is a set of vectors.
  - $\diamond$ The inclusion–maximal (linearly) independent subsets are bases of the span of F'.
  - $\diamond$ As is well known, all bases of a linear subspace are of equal size.
- From the logical function of the auxiliary claim as a "bridge" between two equivalent statements, it follows immediately that an independence system is a matroid if, and only if, the statement of the auxiliary claim is satisfied.

Alternative formulation of the greedy algorithm:

Typically, the greedy algorithm on independent sets is formulated in the literature as follows:

1. Sort all features $f_1, \ldots, f_n$ such that $c[f_1] \geq c[f_2] \geq \cdots \geq c[f_n]$.
2. Set $I = \emptyset$.
3. For $i = 1, \ldots, n$ do:
   (a) If $I \cup \{f_i\}$ is independent, insert f_i into I.
   (b) Otherwise, do not change I in the i–th iteration.
4. Deliver the final I as the output of the algorithm.

Greedy algorithm for start solutions:

- The greedy algorithm is often useful to generate a first start solution for local search.
- *Example*: the matching problem.
- *Greedy matching*:

  1. Set $M = \emptyset$.
  2. For each edge $e \in E$ (in an arbitrary order) do:
     (a) If $M \cup \{e\}$ is a matching, insert e into M.
     (b) Otherwise, do not change M in this iteration.
  3. Deliver the final M.

Comments

- Very often, the greedy matching is not much smaller than a maximal matching.
- Obviously, the number of search steps required by the local–search scheme from Slide 47 is just

  the size of a maximal solution

  minus

  the size of the start solution.
- Matching is but an example of greedy start solution; we can use greedy on any independence system to compute a (hopefully) good inclusion-maximal independent set.

Approximate maximum through greedy:

- Consider a finite set E and an independence system on E.
- The auxiliary claim formulated on Slide no. 230 says that, for each subset $E' \subseteq E$, the size of an inclusion–maximal independent subset $I \subseteq E'$ is a constant.
- On Slide 240 we have seen that, for matrix matroids, this property follows immediately from basic linear algebra.

Approximate maximum through greedy (cont'd):

- In conformance to linear algebra, the size of the inclusion-maximal independent subsets of E' is called the *rank* of E' in the literature.

  $\longrightarrow$ A generalization of the rank of a matrix
  (matrix = set of its column vectors).

- For an arbitrary independence system on E, not necessarily a matroid, and for a subset $E' \subseteq E$ we can define:

  $\diamond$ $\min(E')$ = the m*in*imal size of an inclusion-maximal independent subset of E' and

  $\diamond$ $\max(E')$ = the m*ax*imal size of an inclusion-maximal independent subset of E'.

Approximate maximum through greedy (cont'd):

- Let $\mathcal{E}$ be the set of subsets $E' \subseteq E$ such that E' contains at least one non-empty independent set.
- For $E' \in \mathcal{E}$, $q(E') = \dfrac{\max(E')}{\min(E')}$ is the *rank quotient* of E'.
- The rank quotient of $(E, \mathcal{I})$ is defined as $q(E, \mathcal{I}) = \max\limits_{E' \in \mathcal{E}} q(E')$.
- The following fact has been proved in the literature (proof omitted here):

  For an independence system $(E, \mathcal{I})$, the objective value of the result produced by the greedy algorithm is at most a factor of $q(E, \mathcal{I})$ off the maximal objective value.

  $\longrightarrow$ A generalization of the Theorem on Slide 228.

Remark

The cyclic proof incl. the auxiliary claim on Slide no. 230 implies that the following two statements are equivalent:

- The independence system is a matroid.
- It is $\min(E') = \max(E')$ for all $E' \subseteq E$.

Concluding remark on the history:

- The greedy algorithm for the MST problem is usually called the *algorithm of Kruskal* in the literature.
- This algorithm is named after its inventor, J.B. Kruskal.
- Kruskal also pioneered the mathematical research on the greedy algorithm and its connection to matroids.
- However, matroids have been introduced a few decades before to attack the so–called *four–color conjecture*:

  Every imaginable planar map of "countries" can be colored by at most four colors such that any two adjacent countries are assigned different colors.

Section 3.3:

Decision Tree Traversal Orders

Introductory comments

- In the greedy approach, the rules for the selection and order of the explored tree nodes was simple:

  One single branch — from the root to a leaf.
- If more than one branch is to be explored, the question indeed arises
  - ◇ which subtrees shall be excluded from the exploration and
  - ◇ in which order the remaining tree nodes shall be explored.
- We will see that both questions are intimately related:
  - ◇ The benefit from a decision in one question depends on the decision in the other question.
  - ◇ However, not so strongly that one decision would completely dictate the other one.
- To be more systematic, we will focus on the second question on the next few slides.

Tree traversal

- Obviously, a tree node (except for the root) can only be visited when its immediate predecessor has been visited before.

  $\longrightarrow$ For each arc (v, w) of the decision tree, w enters the state "visited" only after v.

- Thus, a "reasonable" tree traversal order can be viewed as propagating a "frontier line" through the tree:

  $\diamond$ In each step, one arc (v, w) with v already visited and w not yet visited is chosen.

  $\diamond$ The node w is visited.

- The individual tree traversal strategies differ in the selection rule for the arc (v, w).

Canonical tree traversal orders

Three kinds of traversal order are quite obvious and used in practice almost exclusively:

- *depth-first search*,
- *breadth-first search*,
- *best-first search*.

$\longrightarrow$ To be discussed in more detail on the next few slides.

Depth–first search

- Consider a stage of the tree traversal after visiting k nodes, say.
- Let $v_1, \ldots, v_k$ be the nodes visited so far (v_i visited before v_{i+1} for $i \in \{1, \ldots, k-1\}$).
- Let $i \in \{1, \ldots, k\}$ be maximal such that there is an arc (v_i, w) with w not yet visited.
- Choose one of the arcs (v_i, w).

Use a LIFO (last-in first-out) stack for depth-first search

- A node v is inserted in the stack when the search descends from v's parent to v.
- The next node to be visited is one of the children of the top element of the stack.
- A node v is removed from the stack once all immediate children of v have been visited and the search ascends back from v to v's parent.
- So, at any time, the nodes in the stack form the (unique) path from the root to the current top element.

$\longrightarrow$ If the next arc from v is always the leftmost one not yet processed, the nodes in the stack form a "frontier line" that passes from left to right through the tree.

Breadth–first search

- Let $(v_1, w_1), \ldots, (v_k, w_k)$ be the arcs of the current frontier line at some stage of the tree traversal ($v_i = v_j$ possible for $i \neq j$).

- For a node v, let $h(v)$ be the height level of v in the decision tree.

  $\longrightarrow h(w) = h(v) + 1$ for every arc (v, w) of the decision tree.

- Choose an arc (v_i, w_i) such that $\ell(v_i)$ is minimal.

Use a FIFO (first-in first-out) queue for breadth-first search

- A node v is appended to the queue (added to the back) when the search descends from v's parent to v.
- The next node to be visited is one of the children of the first element of the queue.
- A node v is removed from the queue when all immediate children of v have been visited.

$\longrightarrow$ At any time, the nodes in the queue form a "frontier line" in the decision tree, which passes "horizontally" through the tree (the first few elements one height level deeper than the other elements).

Best–first search

- For the tree traversal, a "goodness function" $g : V \longrightarrow \mathbb{R}$ is defined on the nodes of the decision tree.
- In each stage, choose an arc $(v, w) \in A$ with $g(w)$ maximal among all arcs (v, w) such that v has been visited but not yet w.
- Typically, the goodness function value of a node v is an estimate of the best possible leaf (=solution) in the subtree rooted at v.
- *Simple example*: complete the partial solution corresponding to v by a greedy-like algorithm and take the objective value of the result as $g(v)$.

Use a priority queue for best-first search

- For example, may be implemented as a *heap*.
- Much like a FIFO queue.
- Difference: the top element is not the one that was inserted first but the "best" one according to some criterion that can be expressed as a numerical value for each element.
- Here the numerical value of a node v is just $g(v)$.

$\longrightarrow$ At any time, the nodes in the queue form a "frontier line" in the decision tree, which does not follow any particular pattern (as opposed to breadth-first search).

Tree traversal and exclusion of subtrees

- Whenever the decision tree is explored exhaustively, the choice of the tree traversal order does not make a significant difference.
- However, we said that we want to exclude subtrees from the traversal to cut down the run time.
- Technically, each tree traversal order goes with each subtree exclusion strategy.
- However, we will see that certain "matings" of a tree traversal order and a subtree exclusion strategy are more natural and more promising than others.

Section 3.4: Exhaustive Enumeration

Why not enumerate exhaustively

- As mentioned on Slide no. 193, the decision tree is typically way too large to be explored exhaustively.
- For $\mathcal{NP}$–hard problems (the overwhelming majority of all practically relevant problems!), even the existence of a decision tree of polynomial size would be very surprising, since $\mathcal{NP}$-hard problems are (most likely) not solvable in polynomial time.
- However, sometimes a relatively small decision tree is possible for a specific *application* of an algorithmic problem.

$\longrightarrow$ Details on the next few slides.

Decision tree for a specific application

- In the examples considered so far,
  - ◇ the decision tree to an instance and,
  - ◇ in particular, the size of this tree,

  were basically exponential (or worse) in the size of the instance.
- *Example*: the number of points in the TSP.
- However, the decision tree (and thus its size) may alternatively depend in a more involved fashion on the characteristics of the instance.

Decision tree for a specific application (cont'd)

- For an $\mathcal{NP}$–hard problem and any definition of decision tree for this problem, at least *some* instances of moderate sizes *must* exist whose decision trees are intractably large.
- However, sometimes, one may (more or less) safely expect that this kind of instance does not occur in a given, concrete application.
- The definition of such a decision tree is often not obvious from the problem definition.
- In fact, it usually depends on the characteristics of the application.

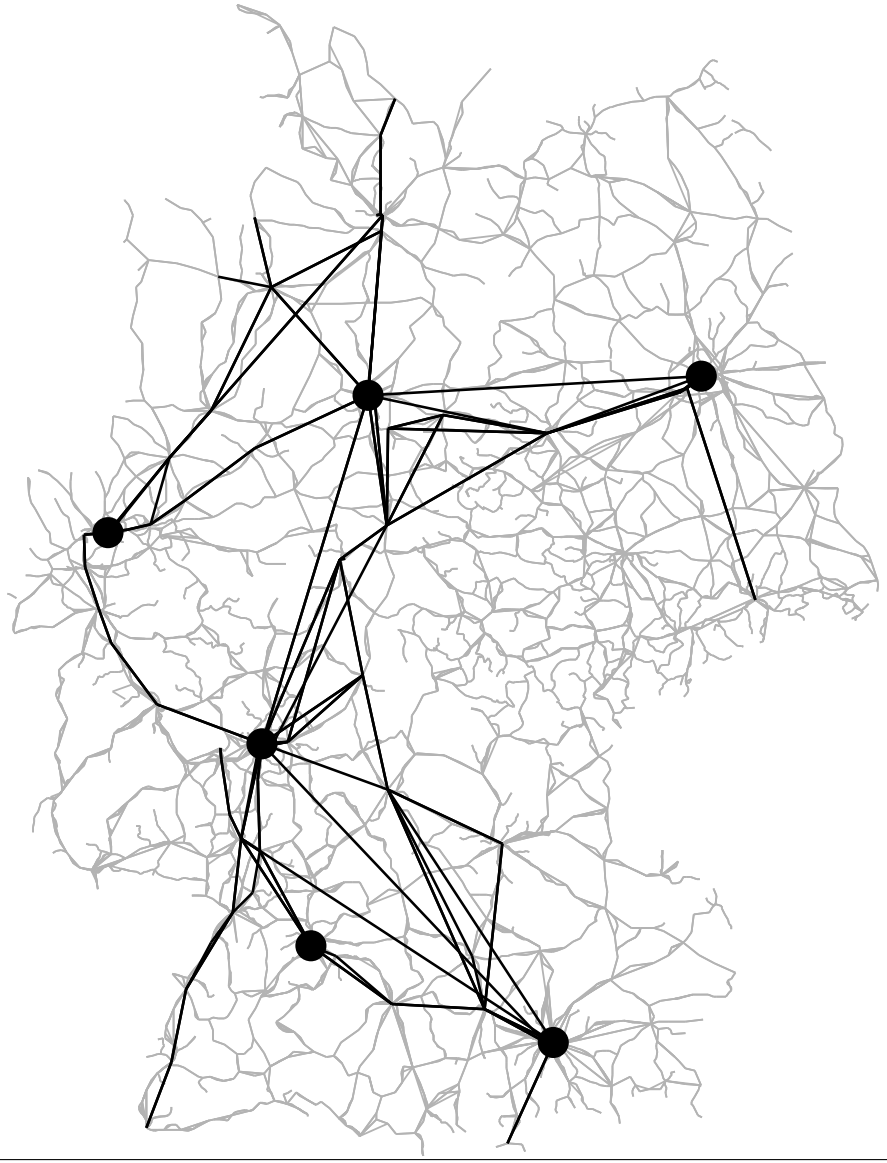
Case study: hitting set

- *Input*: a ground set $\mathcal{F}$ and a collection $\mathcal{S}$ of subsets of $\mathcal{F}$.
- *Feasible output*: a subset $\mathcal{F}' \subseteq \mathcal{F}$ of $\mathcal{F}$ such that $s \cap \mathcal{F}' \neq \emptyset$ for every $s \in \mathcal{S}$.
- *Objective*: minimizing $\#\mathcal{F}'$.

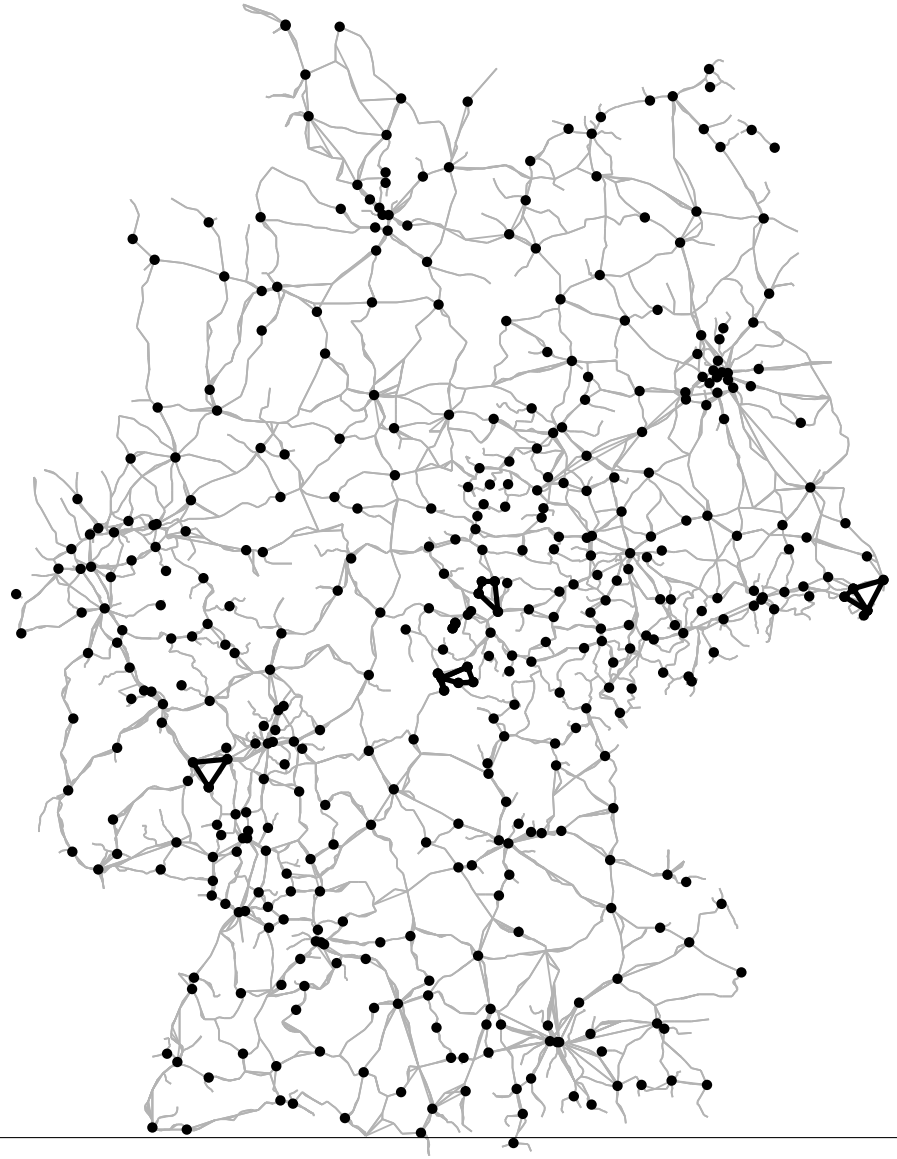
Example from the real world

- $\mathcal{F}$ is the set of all stations of a railroad system.
- There is an element of $\mathcal{S}$ for each train in the schedule.
- An element $s \in \mathcal{S}$ consists of all stations where the corresponding train stops.
- The problem is to find a set $\mathcal{F}' \subseteq \mathcal{F}$ of *service stations* such that each train stops at at least one service station.

For the German ICEs:



For all German trains:



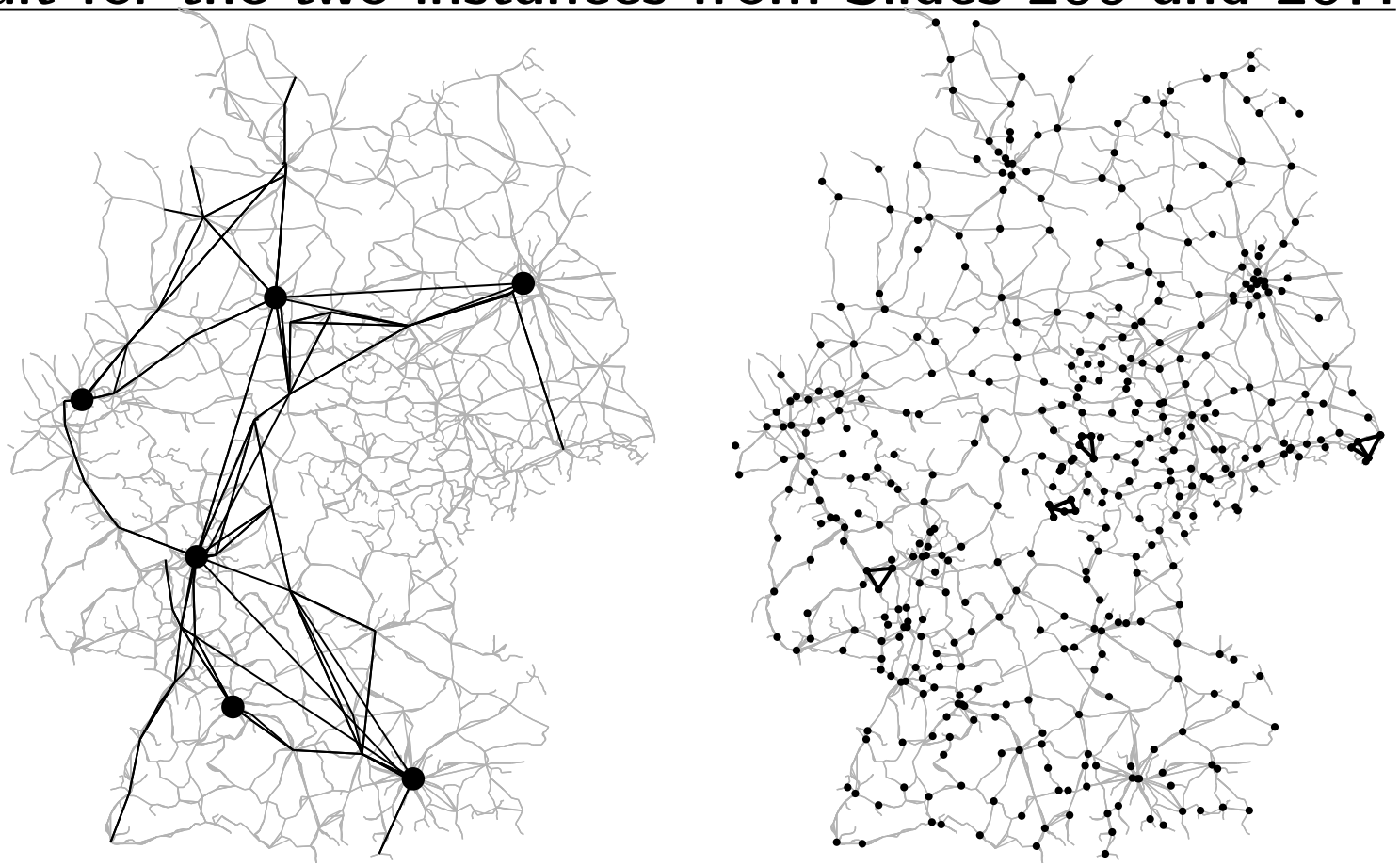
Complexity of the hitting–set problem:

- This problem is well known to be $\mathcal{NP}$–hard.
- In general, there are $2^{\#\mathcal{F}}$ possible subsets $\mathcal{F}' \subseteq \mathcal{F}$.
- *Consequently*:
  - ◇ If $\#\mathcal{F}$ is *very* small, total enumeration is feasible.
  - ◇ However, there might not be too many applications where $\#\mathcal{F}$ is that small.
  - ◇ For instance, in our concrete train example, $\mathcal{F}$ is prohibitively large.

Data reduction:

- Simple reduction techniques apply:
  - ◇ If all trains that stop at station A also stop at station B, then A may be removed from the set of stations (and from the list of stops of each train).
  - ◇ If train A stops at all stations where train B stops, then A may be removed from the set of trains (and from the list of trains at every station).
- These two techniques may be applied as often as possible, and every optimal solution to the reduced instance is still an optimal solution to the original instance.
- *Observation*: If a station becomes isolated, but is not removed, it belongs to every feasible solution to the reduced instance.

Result for the two instances from Slides 266 and 267:



What's going on:

- The data reduction maintains the optimal value.
- If the reduction decomposes the graph into connected components, we get an optimal solution to the entire instance by computing an optimal solution to each connected component and concatenating all of them.
- If a connected component is an isolated node, this node is the optimal solution to this connected component.
- The repeated application of the two reduction techniques has simplified the ICE instance to a set of isolated stations.

  $\longrightarrow$ Optimal solution found through reduction only!
- The all–German–trains instance was simplified to a set of isolated stations and a few, *very* small connected components.

  $\longrightarrow$ For each connected component, the decision tree is small enough to be searched exhaustively.

Interesting observation:

Exactly this phenomenon occurred in all tested instances (taken from all over Europe).

Morale of this story:

- Sometimes, total enumeration is efficient.
- Do not overlook the simple things that you can do!