ALGORITHMICS

Ganian, Nöllenburg, Obszelka, Raidl, Slivovsky, Szeider

Winter term 2020/21

Mirkl Mork der Markur Mer (o)der Tom Poise

9. April 2021

Inhaltsverzeichnis

1 Network Flows and Matchings

1.1 Preliminaries

Definition 1 (Maximum Flow Problem)**.** A *flow network* is a structure $N = (V, E, s, t, c)$, where $G = (V, E)$ is a directed graph[1], s = source and t = sink, $c : E \to \mathbb{R}^+$ capacity. Assumptions: $c(e) \in \mathbb{Z}^+$, s has no incoming edges, t has no outgoing edges and no vertex is isolated.

Definition 2 (s-t flow)**.** An *s-t flow* is a function f that satisfies:

- For each $e \in E$: $0 \le f(e) \le c(e)$ (*capacity*)
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (*conservation*)

Definition 3 (Value)**.** The *value* of a flow f is $v(f) = \sum_{e \text{ out of } s} f(e)$.

Problem 1: Maximum Flow

Find a s-t flow of largest value.

Definition 4 (s-t cut)**.** An *s-t cut* is a partition (A, B) of V with $s \in A$ and $t \in B$.

Definition 5 (Capacity)**.** The *capacity* of a cut (A, B) is $cap(A, B) = \sum_{e \text{ out of } A} c(e)$.

Problem 2: Minimum Cut

Find a s-t cut of smallest capacity.

Lemma 1 (Flow value)**.** *Let f be a flow in a network (V, E, s, t, c), and let (A, B) be an s-t cut. Then*

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f)$$

where the LHS is the net flow sent across the cut and the RHS is the flow leaving s.

Theorem 1 (Weak duality)**.** *Let f be a flow in a network (V, E, s, t, c), and let (A, B) be an s-t cut. Then $v(f) \le cap(A, B)$.*

Corollary 1 (Equivalence)**.** *Let f be a flow in a network (V, E, s, t, c), and let (A, B) be an s-t cut. If $v(f) = cap(A, B)$, then f is a maximum flow and (A, B) is a minimum cut.*

1.2 Ford-Fulkerson Algorithm

Definition 6 (Augmenting Path)**.** Let f be a flow in a network (V, E, s, t, c). An *augmenting path* is a directed path P from s to t such that for every edge e on P we have $f(e) < c(e)$.

Definition 7 (Bottleneck value)**.** The *bottleneck value* of an augmenting path P is $\min_{e \in P} c(e) - f(e)$, denoted by *bottleneck*(P, f).

[1] If not stated otherwise, then G is always a digraph with vertex set V of size n and edge set E of size m.

Method 1 (Augmenting the flow along a path)**.** Given an augmenting path P with bottleneck value b we can find a flow f' which is defined $f'(e) = f(e) + b$ for edges e on P and $f'(e) = f(e)$ for all other edges.

Definition 8 (Residual graph)**.** The *residual graph* $G_f = (V, E_f)$ of a network (V, E, s, t, c) is defined as follows:

- $E_f := \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$ where $e^R := (w, v)$, given $e = (v, w)$.
- Moreover, for flow $f(e)$ and capacity $c(e)$: *Residual capacity* $c_f(e) = c(e) - f(e)$ (*leftover cap*) and $c_f(e^R) = f(e)$ (*undo cap*).

Note, if an edge is full, the residual graph contains only the reversed edge and if an edge is empty, the residual graph contains only the edge in its original direction.

Algorithm 1: Ford-Fulkerson Algorithm

```
Input: $(G, s, t, c)$
foreach $e \in E$: $f(e) \leftarrow 0$
$G_f \leftarrow$ residual graph
while there exists an augmenting path $P$ do
    $f \leftarrow$ Augment$(f, c, P)$
    update $G_f$
end
return $f$
```

where in Augment(f, c, P) for each $e \in P$:

- if $e \in E$, then put $f'(e) = f(e) + \mathit{bottleneck}(f, c, P)$ (forward edge)
- if $e^R \in E$, then put $f'(e^R) = f(e^R) - \mathit{bottleneck}(f, c, P)$ (reverse edge)

Theorem 2 (Augmenting Path)**.** *If there are no augmenting paths P in the residual graph, then flow f is a max flow.*

Theorem 3 (Runtime: Ford-Fulkerson)**.** *For $|V| = n$ and $|E| = m$, Ford-Fulkerson runs in $\mathcal{O}(nmC)$ time.*[2]

Theorem 4 (Max-flow min-cut)**.** *The value of a max flow is equal to the capacity of a min cut.*[3]

Theorem 5 (Integrality)**.** *If all capacities are integers, then there exists a max flow f for which every flow value $f(e)$ is an integer.*

Remark 1. Note, that the runtime of the naive Ford-Fulerson is not yet polynomial (e.g. maximum capacity is C) w.r.t to input size. Optimization via Scaling-Max-Flow(G, s, t, c), where we choose paths with the highest bottleneck value. Runtime: $\mathcal{O}(m^2 \log C)$.

[2]Where we assumed that all capacities c are integers between 1 and C and the flow values $f(e)$ and residual capacities $c_f(e)$ are integers (Invariant).

[3]Given a flow f of maximum value, we can compute an s-t cut of minimum capacity in $\mathcal{O}(m)$ time. Hence, Ford-Fulkerson finds a minimum cut.

1.3 Extensions to Max Flow

Definition 9 (Circulations with demands)**.** Given a directed graph $G = (V, E)$, capacities c and supplies and demands $d : V \to \mathbb{R}$. Vertex v is called

1. *demand node*, if $d(v) < 0$,
2. *supply node*, if $d(v) > 0$,
3. *transshipment node*, if $d(v) = 0$.

A *circulation* f is a function such that

- For each $e \in E$: $0 \leq f(e) \leq c(e)$ (*capacity*)
- For each $v \in V$: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (*conservation*)

Problem 3: Circulation with demands

Given (V, E, c, d), decide whether there exists a circulation.

Method 2 (Reduction to Maximum Flow 1)**.** Add new source s and sink t. For each v with $d(v) < 0$, add edge (s, v) with capacity $-d(v)$ and for each v with $d(v) > 0$, add edge (v, t) with capacity $d(v)$.

Theorem 6. *(V, E, c, d) has a circulation iff the corresponding flow network (V, E, s, t, c) has a maximum flow with value D, where D is defined as*

$$\sum_{v \text{ with } d(v)>0} d(v) = \sum_{v \text{ with } d(v)<0} -d(v)$$

Definition 10 (Circulation with demands and lower bounds)**.** Give G, c, d like before and lower bounds $\ell : V \to \mathbb{R}$. A *circulation* f is a function such that

- For each $e \in E$: $\ell(e) \leq f(e) \leq c(e)$ (*capacity*)
- For each $v \in V$: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (*conservation*)

Problem 4: Circulation with demands and lower bounds

Given (V, E, ℓ, c, d), decide whether there exists a circulation.

Remark 2. Idea: Model lower bounds with demands, i.e. transform an instance (V, E, ℓ, c, d) to an instance (V, E, c, d). Then there exists a circulation in (V, E, ℓ, c, d) iff there exists a circulation in (V, E, c, d)

Application (*Matrix Rounding*). Given a matrix $D_{ij} \in \mathbb{R}^{p \times q}$ with row sums a_i and column sums b_j. We want to round each d_{ij}, a_i and b_j to an integer, such that the sum of rounded elements in each column (row) equals the corresponding column (row) sum.

Problem 5: Circulation with demands and lower bounds

Find a feasible rounding, if one exists.

Theorem 7 (Matrix rounding). *Feasible matrix rounding always exists.*[4]

1.4 Bipartite matching

Definition 11 (Matching). $M \subseteq E$ is called *matching* if each vertex appears in at most one edge in M. A matching is *perfect* if $2|M| = |V|$.

Definition 12 (M-Cover). A vertex $v \in V$ is called M-*covered* if there exists an edge $\{v, w\} \in M$. Otherwise v is called M-*exposed*.

Definition 13 (Bipartite Graph). An undirected graph (V, E) is called *bipartite* if there is a partition of $V = L \dot\cup R$ such that any $e \in E$ connects one vertex from L to one vertex in R.

Definition 14 (Bipartite Matching). $M \subseteq E$ is a matching if each vertex appears in at most edge in M.

Problem 6: Maximum (Bipartite) Matching

Find a matching of maximal cardinality.

Method 3 (Reduction to Maximum Flow 2). Create graph $G' = (L \cup R \cup \{s, t\}, E')$ and direct all edges from L to R with capacities of infinite (or unit) value and direct edges from s to all nodes in L with unit value and direct edges from all nodes in R to t with unit value.

Theorem 8 (Cardinality and Max Flow). *Let M be matching of maximal cardinality. Then $|M| = v(f)$ for maximal flow f in G'.*

Corollary 2 (Runtime: Maximum Bipartite Matching). *With the Ford-Fulkerson algorithm we can find a maximum bipartite matching in $\mathcal{O}(mn)$ time. (Or with shortest-augmenting path: $\mathcal{O}(m\sqrt{n})$)*

Vertex Cover

Definition 15 (Vertex Cover). A *vertex cover* (VC) of a graph (V, E) is a set $C \subseteq V$ such that every edge $e \in E$ has at least one of its ends in C.

Problem 7: Minimum Vertex Cover

Find a vertex cover of minimal cardinality.

Theorem 9 (König-Egerváry Theorem: 1931). *The size of a smallest vertex cover of a bipartite graph equals the size of a maximum matching.*

Corollary 3. *A minimum vertex cover of a bipartite graph can be found in polynomial time.*[5]

[4]Proof: Formulate as a circulation problem with lower bounds.

[5]By the way: Finding a smallest vertex cover of a general graph is **NP**-hard. This is done via reducing 3-SAT to VC.

Perfect Matching and k-regular bipartite graphs

Definition 16 (Bipartite perfect matching)**.** A matching $M \subseteq E$ is perfect if each vertex is incident with exactly one edge in M.

Theorem 10 (Marriage Theorem: Frobenius 1917, Hall 1935)**.** *Let $G = (L \cup R, E)$ be a bipartite graph with $|L| = |R|$. Then, G has a perfect matching iff $|N(S)| \geq |S|$ for all subsets $S \subseteq L$, where $N(S)$ is the set of vertices adjacent to vertices in S.*

Definition 17 (k-regular bipartite Graph)**.** A graph G is *k-regular* if each vertex has the same degree k. A graph has degree d if each vertex has at most d neighbors.

Theorem 11 (König 1916, Frobenius 1917)**.** *Every k-regular bipartite graph has a perfect matching.*

Disjoint Paths

Definition 18 (edge-disjoint)**.** Two paths are edge-disjoint if they have no edge in common.

Problem 8: Disjoint Path

Given a digraph (V, E) and two vertices s and t: Find a maximum number of edge-disjoint s-t paths

Theorem 12. *The maximum number edge-disjoint s-t paths equals the maximum flow value*[6].

Definition 19 (Disconnection)**.** A set of edges $F \subseteq E$ disconnects t from s if every s-t path uses at least one edge in F.

Problem 9: Network Connectivity

Given a digraph (V, E) and two vertices s and t: Find a minimal number of edges whose removal disconnects t from s.

Theorem 13 (Menger 1927)**.** *The maximum number of edge-disjoint s-t paths is equal to the minimum number of edges whose removal disconnects t from s*

2 Parameterized Algorithms

2.1 Fixed Parameter Tractability

Problem 10: (Parameterized) Vertex Cover

Instance: A graph G and an integer k.
Parameter: k.
Task: Choose at most k vertices in G such that each edge is covered by at least one chosen vertex.

[6]In the corresponding transformation/reduction

Problem 11: k-Vertex Cover

Instance: A graph G.
Task: Choose at most k vertices in G such that each edge is covered by at least one chosen vertex.

Remark 3. In the VERTEX COVER we treat k as a parameter, whereas in the k-VERTEX COVER we treat k as a constant. Consequently, one can show that k-VERTEX COVER is in **P**.

Method 4 (Bounded search tree: Vertex Cover)**.**

1. test $k \neq 0$; if $k = 0$ and G contains an edge, output false (for this branch).
2. pick a vertex v incident to some edge.
3. either v is in the vertex cover, or all its neighbors are $\rightsquigarrow$ branching.

[7]

Definition 20 (Fixed-Parameter Algorithm)**.** A *fixed-parameter algorithm* (or *FPT Algorithm*) for a parameterized problem is an algorithm that runs in time:

$$f(k) \cdot poly(n)$$

where f is an arbitrary computable function and *poly* is a polynomial function independent of k. If there exists a FPT algorithm for a given problem, then we call the problem *fixed parameter tractable.*

Example 1. Bounded search tree w.r.t. to VC: Branching tree with 2 children and depth at most k yields runtime $\mathcal{O}(2^k \cdot n)$. Thus, it is a FPT algorithm.

Definition 21 (Independent set)**.** A set $S \subseteq V$ is called *independent* if for any two vertices in S, there is no edge connecting the two.

Problem 12: Independent Set

Instance: A graph G and an integer k.
Parameter: k.
Task: Choose at least k vertices in G which are pairwise independent.

Theorem 14 (Optimality rule)**.** *If there exists an independent set which does not contain a vertex v and any neighbor of v, then adding v results in a new independent set which is larger.*[8]

Definition 22 (Proper c-coloring and k-coloring)**.** A *proper c-coloring* of a graph is a coloring of the vertices by c colors such that no two neighbors have the same color.
A coloring using at most k colors is called a *proper k-coloring.* The smallest number of colors needed to color a graph G is called its chromatic number, and is often denoted $\chi(G)$.

[7]The algorithm tries to build a feasible solution to the problem by making a sequence of decisions on its shape, such as whether to include some vertex into the solution or not. Whenever considering one such step, the algorithm investigates many possibilities for the decision, thus effectively branching into a number of subproblems that are solved one by one. In this manner the execution of a branching algorithm can be viewed as a search tree. If the total size of the search tree is bounded by a function of the parameter alone, and every step takes polynomial time, then such a branching algorithm runs in FPT time.(Parameterized Algorithms, 2015)]

[8]if there exists a solution which does not pick a vertex v and any neighbor of v, then there also exists a solution which picks v.

Problem 13: k-coloring

Instance: A graph G and an integer k.
Parameter: k.
Task: Find a proper k-coloring of G.

Theorem 15 (Runtimes: k-coloring and independent set)**.** *The problems k-coloring and independent set are not FPT, but para**NP**-complete*[9]*.*

Problem 14: Independent set on graphs of bounded degree

Instance: A graph G of degree at most 3 and an integer k.
Parameter: k.
Task: Choose at least k vertices in G which are pairwise independent.

Method 5 (Bounded search tree: Independent set on graphs of bounded degree)**.**

1. if $k = 0$, output true, else if $k > 0$ and the graph is empty, output false (for this branch).
2. pick an arbitrary vertex v.
3. Branch into 4 options as per the previous slide to choose at least one vertex into the independent set.
4. Delete the chosen vertex and all of its neighbors.
5. Restart on the remaining graph with $k := k - 1$.

Theorem 16 (Runtime: Bounded search tree: Independent set on graphs of bounded degree)**.** *The algorithm above is in $\mathcal{O}(4^k \cdot n)$.*

Definition 23 (Dominating set)**.** A set $X \subseteq V$ is *dominating set* in a graph G if each vertex of G is either in X or has a neighbor in X.

Problem 15: Dominating Set

Instance: A graph G of maximum degree 3 and an integer k.
Parameter: k.
Task: Find a dominating set in G of cardinality at most k.

Method 6 (Bounded search tree: Dominating set on graphs of bounded degree)**.**

1. if $k = 0$ and there is an undominated vertex, output false (for this branch) else if $k > 0$ and there are no undominated vertices, output true.
2. pick an arbitrary undominated vertex v.
3. Branch into 4 options (v or a neighbor of v) to put at least one vertex into the dominating set.

[9] class of parameterized problems that can be solved by a nondeterministic algorithm in time $f(k) \cdot |x|^{\mathcal{O}(1)}$

4. Delete the chosen vertex and mark all of its neighbors as dominated.

5. Restart on the remaining graph with $k := k - 1$.

Theorem 17 (Runtime: Bounded search tree: Dominating set on graphs of bounded degree)**.** *The algorithm above is in* $\mathcal{O}(4^k \cdot n)$*.*

2.2 Kernelization

Definition 24 (Kernelization)**.** A *kernelization* (*kernelization algorithm*) is a polynomial-time algorithm which takes as input an instance (I, k) of $\mathcal{P}$ (parameterized decision problem) and outputs an instance $(I'; k')$ of $\mathcal{P}$ such that:

1. (I, k) is a yes-instance iff $(I'k')$ is a yes-instance of $\mathcal{P}$,

2. there exists a computable function h such that $|I'| \leq h(k)$ and[10] $k' \leq h(k)$. The instance (I', k') is then called the *kernel*. If h is a polynomial function, then we speak of *polynomial kernels*.

Lemma 2 (Kernelization)**.** *Any decidable parameterized problem* $\mathcal{P}$ *is FPT iff it admits a kernelization.*

Method 7 (Reduction rules)**.** Most standard kernelization technique: reduction rules, the idea being to formulate a set of reduction rules such that (1) each rule runs in polytime and (2) each rule slightly modifies the instance to obtain an equivalent instance (typically the new instance is smaller and the parameter value does not increase). To successfully obtain a kernel, it is necessary to show:

1. each rule is safe: it does not change yes-instances to no-instances and vice-versa

2. if no rule can be applied, then we already have a (polynomial) kernel

Method 8 (Reduction rules: : Vertex Cover)**.**

1. Rule 1: If there is a vertex v of degree at least $k + 1$, then delete v and set $k := k - 1$.

2. Rule 2: If there is a vertex v with no neighbors, then delete v.

Theorem 18 (Runtime VC)**.** Vertex Cover *admits a quadratic kernel.*

Definition 25 (Clique)**.** A *Clique* $C \subseteq V$ of an undirected graph, such that every two distinct vertices in C are adjacent; that is, its induced subgraph is complete[11].

Problem 16: Clique

Instance: A graph G and an integer k.
Parameter: k.
Question: Do there exist at least k vertices in G which form a clique?

[10] $|I'|$ denotes the number of vertices and edges of instance I'.
[11] A *complete graph* is a simple undirected graph in which every pair of distinct vertices is connected by a unique edge.

Remark 4 (Observation). A $C \subseteq V$ is independent iff it is a clique in the graph's G complement $\overline{G}$[12], so the two concepts are complementary.

Theorem 19 (Runtime: Clique). *Clique is not FPT, and hence does not admit a kernel.*

Problem 17: Maxsat

Instance: A CNF formula F and an integer k.
Parameter: k.
Question: Is there an assignment satisfying at least k clauses in F?

Theorem 20 (Runtime: Maxsat). *Maxsat admits a $\mathcal{O}(k^2)$ kernel.*

Definition 26 (Hitting set). Let $\mathcal{F}$ be a set of subsets of finite set U. The smallest subset Y of U that meets every member of $\mathcal{F}$ is called the *hitting set.* A set X hits a set $F \in \mathcal{F}$ iff $F \cap X \neq \emptyset$.

Problem 18: Hitting set

Instance: A ground set U, a set $\mathcal{F}$ of subsets of U, and an integer k.
Parameter: k.
Question: Does there exist a set $X \subseteq U$ of cardinality at most k which hits each subset in $\mathcal{F}$?

Problem 19: d-Hitting set

Instance: Set U, set $\mathcal{F}$ of subsets of U each of cardinality at most d, and an integer k.
Parameter: k.
Question: Does there exist a set $X \subseteq U$ of cardinality at most k which hits each subset in $\mathcal{F}$?

Theorem 21 (Runtime: (d-)Hitting set). *Hitting Set is not FPT, whereas d-Hitting Set is FPT.*

Lemma 3 (Sunflower lemma). *Let $\mathcal{F}$ be a set of subsets of a ground set U such that*

1. *each $F \in \mathcal{F}$ has cardinality at most d, and*
2. $|\mathcal{F}| > d! \cdot k^d$

Then there exists a $(k+1)$-sunflower. Moreover, such a $(k+1)$-sunflower can be found in polynomial time.

Method 9 (Reduction: Sunflower lemma). For an instance $(U, \mathcal{F}, k)$ of d-Hitting Set:

1. Remove elements of U that do not intersect anything in $\mathcal{F}$.
2. If $|\mathcal{F}| \leq d! \cdot k^d$, then we already have a $\mathcal{O}(k^d)$ kernel, otherwise we apply the Sunflower lemma and

   find a $(k+1)$-sunflower.

[12]A complement (of a graph G) is the graph obtained by complementing (reversing) all edges, i.e., an edge is in the complement iff it is not there in the original graph.

3. Apply sun ower reduction rule to remove $k+1$ elements of $\mathcal{F}$ and add the core C to $\mathcal{F}$; this reduces the size of $\mathcal{F}$ by k[13].

4. Restart.

Theorem 22. *d-Hitting Set admits a $\mathcal{O}(k^d)$ kernel.*

2.3 Structural Parameters

Definition 27 (Hamiltonian Cycle)**.** A cycle which goes through each vertex in G is called *Hamiltonian*.

Problem 20: Hamiltonian Cycle using vertex cover

Instance: A graph G and a vertex cover X of G.
Parameter: $k = |X|$.
Question: Does there exist a Hamiltonian cycle in G?

Problem 21: Chromatic Number using vertex cover

Instance: A graph G, an integer q and a vertex cover X of G.
Parameter: $k = |X|$.
Question: Does there exist a proper q-coloring of G?

Problem 22: Independent Set using vertex cover

Instance: A graph G, an integer p and a vertex cover X of G.
Parameter: $k = |X|$.
Question: Does there exist an independent set of size at least p in G?

Theorem 23 (Runtimes)**.** Hamiltonian Cycle using vertex cover *admits a polynomial kernel.* Chromatic Number using vertex over *is FPT, but does not admit a polynomial kernel.* Independent Set using vertex cover *admits a polynomial kernel.*

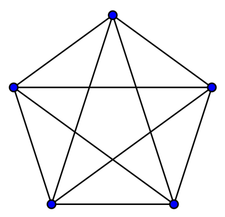
3 Planarity, A* Search, and Randomized Algs.

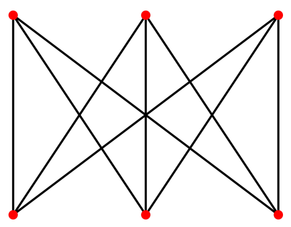
3.1 Planar Graphs

Definition 28 (Planar Graph)**.** A graph $G = (V, E)$ is called *planar*, if it can be drawn (*planar embedding*)in the plane in a way such that no two edges intersect geometrically except at a vertex to which both are incident.

Definition 29 (Face)**.** A *face* of the graph is a region bounded by a set of edges and vertices in the embedding. The unbounded region is called the *outer/external face* of G.

[13]If $C = \emptyset$ we'll be adding $\emptyset$ to $\mathcal{F}$, which correctly results in a clear no-instance (can't hit $k+1$ petals without using a core)


$K5$


$K_{3,3}$

These graphs are often called *Kuratowski graphs*, they cannot be drawn in the plane without crossing edges.

Theorem 24 (Euler). *Let $G = (V, E)$ be a connected, planar graph, and let f denote the number of faces. Then $|V| - |E| + f = 2$ holds for any planar embedding of G.*

Definition 30 (Simple graph). A *simple graph* is an unweighted, undirected graph containing no graph loops or multiple edges.

Corollary 4.

1. *If G is a simple, planar graph with $|V| \geq 3$, then the number of edges is $|E| \leq 3|V| - 6$.*
2. *For any simple, planar, bipartite graph with $|V| \geq 3$ vertices and $|E|$ edges, $|E| \leq 2|V| - 4$ holds.*
3. *Each simple, planar graph contains a vertex v of degree $d(v) \leq 5$.*

Definition 31 (Subdivision). *Subdividing* an edge $(u, v) \in E$ of a graph $G = (V, E)$ is the operation of deleting (u, v) and adding a path $P = (u, w_1, \ldots, w_k, v)$, $w_i \notin V$, $1 \leq i \leq k$ to G. A graph G is called a *subdivision* of a graph G' if it is obtained by subdividing some of the edges of G'.

Theorem 25 (Kuratowski). *A graph is planar iff it does not contain a subdivision of K_5 or $K_{3,3}$.*

Definition 32 (Graph Minor). A graph H is a *minor* of G if H is obtained from G by deleting and contracting edges and/or deleting vertices.

Theorem 26 (Wagner). *A graph is planar iff it has no minor isomorphic to K_5 or $K_{3,3}$.*

Method 10 (Randomized algorithms). Depend on uniformly random numbers as an auxiliary input to guide behavior, usually implemented by a pseudo-random number generator. Two variants:

1. *Monte Carlo algorithms*: always time-efficient (e.g. polynomial), but correct output only with high probability (e.g. Miller-Rabin Primality Test)
2. *Las Vegas algorithms*: always correct output, but time-efficient only in expectation (e.g. Randomized Quicksort)

Remark 5. One can always turn a Las Vegas into a Monte Carlo (stoping the algorithm). The other direction is not always true.

Problem 23: Primality Test

Given a positive integer number n (typically very large), determine whether or not n is prime.

Theorem 27 (Fermat's Little Theorem)**.** *If n is prime, $a^{n-1} \equiv 1 \mod n$ holds for all $a \in \{1, \dots, n-1\}$.*

Corollary 5. *If for some basis a:*

- *If $a^{n-1} \not\equiv 1 \mod n$ then n is not prime and a is called a witness for the compositeness.*
- *If $a^{n-1} \equiv 1 \mod n$then n is prime with some probability.*

```
Algorithm 1: MILLER-RABIN PRIMALITY TEST
Input: $n$
for $i \leftarrow 1, \dots, s$
if $Witness\ (random(2, n-1), n)$ then
|  return not prime
end
return prime
```

```
Algorithm 2: WITNESS
Input: $a, n$
$result \leftarrow 1 \qquad (c \leftarrow 0)$
for $i = k-1, \dots, 0$
    $result \leftarrow (result \cdot result) \mod n$                    $(c \leftarrow 2c)$
            if $b_i = 1$ then $result \leftarrow (result \cdot a) \mod n$     $(c \leftarrow c+1)$;
if $result \neq 1$ then
|  return true ($a$ is witness for $n$ not prime)
else
|  return false ($a$ is not a witness)
end
```

Theorem 28 (Runtime: Miller-Rabin)**.** MILLER-RABIN PRIMALITY TEST *is in $\mathcal{O}(s \cdot k^3)$.*

Problem 24: Max 3-SAT

Given a set of clauses $C_1, \dots, C_k$, each of length 3, over a set of binary variables $X = x_1, \dots, x_n$, find a variable assignment satisfying as many clauses as possible.

Lemma 4. *Given a* MAX-3SAT *instance with k clauses, the expected number of clauses satisfied by a random assignment is $7/8k$.*

Method 11. Repeatedly generate random assignments until one satisfies $\geq 7/8k$ clauses. (This yields a 7/8-approximation algorithm)

Theorem 29 (Runtime: Johnson)**.** JOHNSON *is in $\mathcal{O}(k^2 + kn)$*

3.2 A^* Algorithm

Method 12 (A^* Algorithm)**.** Priority function: $f(x) = g(x) + h(x)$, $x \in V$, where

- $g(x)$: best cost so far to reach x
- $h(x)$: estimated cost from x to goal t (cost-to-go), e.g. $d_1(x,t)$ or $d_2(x,t)$

I.e., $f(x)$ estimates the total cost from s to t via node x.

Algorithm 3: A^* ALGORITHM

```
Input: digraph $G = (V, A)$ with $w_{ij} > 0\ \forall(i,j) \in A$, source node $s$, target node $t$
$g(s) = 0; g(x) = \infty\ \forall x \in V - \{s\}$
priority queue $Q \leftarrow \{(s, f(s) = h(s))\}$
while $Q \neq \emptyset$ do
    $x \leftarrow$ Q.getMin( )                                  remove node with min. f(x)
    if $x$=$t$ then
        return f(t), path given by predecessor list pred(t);
    end
    forall $v$ adjacent to $x$ do
    $g' \leftarrow g(x) + w_{xv}$
    if $g' < g(v)$ then
        $g(v) \leftarrow g'$; $pred(v) \leftarrow x$                    new/better path to v;
        Q.put($v, g' + h(v)$)
    end
end
return no path to target $t$;
```

Theorem 30. *A^* will always terminate in limited time. If a path from s to t exists, A^* will always find one.*

Definition 33 (Admissible heuristic)**.** Let $h^*(x)$ be a true minimal cost from x to t. A heuristic $h(x)$ is *admissible* if it represents a lower bound, i.e. $h(x) \leq h^*(x)\ \forall x \in V$.[14]

Theorem 31. *If $h(x)$ is admissible, A^* is guaranteed to find a least-cost path.*

Definition 34 (Monotonic heuristic)**.** A heuristic$h(x)$ is monotonic (consistent) if $h(x) \leq w_{xy} + h(y)\ \forall(x,y) \in A$ and $h(t) = 0$.[15]

4 Structural Decompositions and Algorithms

4.1 Definitions and basic properties

Definition 35 (Elimination ordering)**.** Let $G = (V, E)$ be a graph. An *elimination ordering* of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

Method 13. Upon eliminating v_i create (fill-in) edges between its neighbors.

Definition 36 (Width of an elimination ordering)**.** The *width* of σ is the maximum degree of a vertex upon elimination.

[14] An admissible heuristic is *optimistic.* $h(x) = 0$ is an admissible heuristic.

[15] Monotonicity implies admissibility of the heuristic.

Definition 37 (Treewidth)**.** Let G be a graph. The *treewidth* of G is the minimum width of an elimination ordering of G.[16]

Definition 38 (Tree decomposition)**.** Let G be a graph. A *tree decomposition* of G is a pair (T, χ) consisting of a tree $T = (V', E')$ and a mapping $\chi : V' \to \mathcal{P}(V')$ (*bag*) such that

1. $\bigcup_{t \in V'} \chi(t) = V$
2. For each $(v, w) \in E$ there is a $t \in V'$ with $v, w \in \chi(t)$.
3. $T[\{t \in V' | v \in \chi(t)\}]$ is a connected subtree for each $t \in V'$.

The *width* of (T, χ) is $\max_{t \in V'} |\chi(t)| - 1$.

Definition 39 (Treewidth and Pathwidth)**.** Let G be a graph. The *treewidth* of G is the minimum width of a tree decomposition of G. A *path decomposition* of G is a tree decomposition (T, χ) of G where T is a path. The *pathwidth* of a graph G is the minimum width of a path decomposition of G.

Theorem 32.

1. *If $H \leq G$ (subgraph), then $\boldsymbol{tw}(H) \leq \boldsymbol{tw}(G)$.*
2. *If $G = G_1 \dot\cup G_2$ (disjoint graphs), then $\boldsymbol{tw}(G) = \max\{\mathbf{tw}(G_1), \mathbf{tw}(G_2)\}$.*
3. *The treewidth of a graph G is the maximum treewidth of a connected component of G.*
4. *If G has minimum degree d then $d \leq \boldsymbol{tw}(G)$.*
5. *A graph of treewidth k has at most kn edges.*

Definition 40 (Small tree decomposition)**.** A tree decomposition (T, χ) is *small* if there is no pair t, t' of adjacent nodes such that $\chi(t) \subseteq \chi(t')$.[17]

Graph Minors and Treewidth

Definition 41 (Graph minors)**.** Let G be a graph. A graph H is called a *minor* of G if it can be obtained from by successive applications of the following operations:

1. Deleting an edge.
2. Deleting an isolated vertex.
3. Contracting an edge.

Theorem 33. *If H is a minor of G, then $\boldsymbol{tw}(H) \leq \boldsymbol{tw}(G)$.*

[16]Observation: The treewidth of an n-vertex graph is at most $n - 1$.
[17]A small tree decomposition of an n-vertex graph has at most n nodes.

4.2 Courcelles's Theorem

Definition 42 (MSO: Syntax)**.** Fix a relational model signature $\mathbb{M}$ with finitely many relations $R \in \mathbb{M}$ (of arity ≥ 1) and a set of individual variables $x, y, \ldots$ and a set of set variables $X, Y \ldots$ (altogether: signature Σ). Then the following formulas are formulas of MSO:

1. $x = y$ (equality)
2. $x \in X$ (or Xx) (membership)
3. $R(x_1, \ldots, x_n)$ for each $R \in \mathbb{M}$ (atomic relational formulas)

Moreover, if φ, ψ are MSO formulas, then the following formulas are formulas of MSO:

1. $(\neg\varphi)$ (negation)
2. $(\varphi \vee \psi)$ (disjunction)
3. $(\exists x)[\varphi]$ (existential quantification for individual)
4. $(\exists X)[\varphi]$ (existential quantification for set of individual)

Nothing else is a MSO formulas[18].

Definition 43 (MSO: Semantics)**.** Given a Σ-structure $\mathcal{M}$ with assignment I from variables to individuals (in M) and $\varphi, \psi \in \text{MSO}[\Sigma]$ we say[19]:

1. $\mathcal{M}, I \models x = y \iff I(x) = I(y)$
2. $\mathcal{M}, I \models Xx \iff I(x) \in I(X)$
3. $\mathcal{M}, I \models R(x_1, \ldots, x_n) \iff (I(x_1), \ldots, I(x_n)) \in R$
4. $\mathcal{M}, I \models (\neg\varphi) \iff \neg(\mathcal{M}, I \models \varphi)$
5. $\mathcal{M}, I \models (\varphi \vee \psi) \iff \mathcal{M}, I \models \varphi$ or $\mathcal{M}, I \models \psi$
6. $\mathcal{M}, I \models (\exists x)[\varphi] \iff (\exists s \in M)[\mathcal{M}, I \models \varphi(x/s)]$
7. $\mathcal{M}, I \models (\exists X)[\varphi] \iff (\exists S \subseteq M)[\mathcal{M}, I \models \varphi(X/S)]$

Problem 25: MSO Model Checking

Given a structure $\mathcal{M}$ and an MSO[Σ] sentence φ decide whether $\mathcal{M} \models \varphi$.

Theorem 34 (Courcelle)**.** *Let $\mathcal{M}$ be an n-element structure and $\varphi \in MSO[\Sigma]$. There exists an algorithm $\mathbb{A}$ that, given a tree decomposition of of width k, determines whether $\mathcal{M} \models \varphi$ in time $f(|\varphi|, k) \cdot n + \mathcal{O}(||\mathcal{M}||)$ for some computable f.*

Theorem 35 (Runtime: MSO Model Checking)**.** MSO Model Checking *is FPT parameterized by the treewidth of $\mathcal{M}$ and $|\varphi|$.*

[18] As usual, $(\forall X)[\varphi] = (\neg\exists X)[\neg\varphi]$ etc . . . bla bla.
[19] An expression of the form $\varphi(a/b)$ denotes the formula that results from φ after applying $a \mapsto_I b$.

Problem 26: LinEMSO Maximization

Input: A graph $G = (V, E, f_1^G, \dots f_m^G, r_1, \dots, r_t)$, an MSO formula $\varphi(X_1, \dots, X_l)$, and a linear EMS evaluation term $g(x_{11}, \dots, x_{ml}, y_1, \dots, y_l)$.
Output: $\max_{\substack{A_1,\dots A_l \subseteq V \cup E \\ G \models \varphi[A_1,\dots,A_l]}} g\left(\sum_{a\in A_1} f_1^G(a), \dots, \sum_{a \in A_l} f_m^G(a), r_1, \dots, r_t\right)$

where $X_1, \dots, X_l$ are free set variables in φ, g is a linear function, $f_i^G : V \cup E \to \mathbb{Q}$ for $i = 1, \dots, m$ and $r_1, \dots, r_t \in \mathbb{Q}$.

Theorem 36 (Arnborg, Lagergren, Seese)**.** *There is a computable function $f(\,.\,,.\,)$ such that* LinEMSO Maximization *can be solved in time*[20] $f(|\varphi| + |g|, k) \cdot ||G||$ *given a width k tree decomposition of G.*

5 Linear and Mixed Integer Linear Programming

5.1 Linear Programming

Traditional approach:

1. identify the problem
2. learn about the problem's properties
3. design an algorithm
4. implement the algorithm
5. compute a solution

Model-based approach:

1. identify the problem
2. learn about the problem's properties
3. design a model
4. feed the model to a solver
5. obtain a solution from the solver

Example 2. Consider the Maximum Flow problem. Then we write the linear program:

$$\begin{aligned}
\max \; & \sum_{(s,i)\in E} f_{si} & \\
s.t. \; & \sum_{(i,v)\in E} f_{iv} = \sum_{(v,j)\in E} f_{vj}, & \forall v \in V - \{s,t\} \\
& 0 \le f_e \le c_e & \forall e \in E
\end{aligned}$$

Definition 44 (Linear program)**.** A *linear program* (LP) is the problem of minimizing a linear cost function subject to linear equality and inequality constraints, that is

$$\begin{aligned}
\min \; & \boldsymbol{c'x} & \\
s.t. \; & \boldsymbol{a_i'x} \ge b_i, & \forall i \in M_1 \\
& \boldsymbol{a_i'x} \le b_i, & \forall i \in M_2 \\
& \boldsymbol{a_i'x} = b_i, & \forall i \in M_3 \\
& x_j \ge 0, & \forall j \in N_1 \\
& x_j \le 0, & \forall j \in N_2
\end{aligned}$$

[20] where $|\varphi|$ is the size of formula φ, $|g|$ is the size of the term g (linear function) and $||G||$ is the size of graph G (including $|V| + |E|$).

- $x_1, \dots, x_n$ are decision variables,
- $\boldsymbol{c'x} = \sum_{i=1}^n c_i x_i$ is the linear (objective) cost function,
- M_1, M_2, M_3 finite index sets for the constraints,
- N_1, N_2 are subsets of $\{1, \dots, n\}$ for variables constraints,
- $x_j, j \notin N_1 \cup N_2$ free (unrestricted) variables.

or, more compactly, for m constraints, $\boldsymbol{A} \in \mathbb{R}^{n \times m}$, $\boldsymbol{b} \in \mathbb{R}^m$:

$$\begin{aligned} \min \;& \boldsymbol{c'x} \\ s.t. \;& \boldsymbol{Ax} \geq \boldsymbol{b} \\ & \boldsymbol{x} \in \mathbb{R}^n \end{aligned}$$

Definition 45 (Feasible solution)**.** A vector $\boldsymbol{x}$ satisfying all constraints is called a *feasible solution* or *feasible vector.* A set of feasible solutions is called *feasible set (region)*, i.e. $\{x \in \mathbb{R}^n | \boldsymbol{Ax} \geq \boldsymbol{b}\}$.

Definition 46 (Optimal feasible solution)**.** A feasible solution $\boldsymbol{x}^*$ that minimizes the objective function ($\boldsymbol{c'x}^* \leq \boldsymbol{c'x}$ for all feasible solutions $\boldsymbol{x}$) is called an *optimal feasible solution.* In this case $\boldsymbol{c'x}^*$ is called *optimal cost.*

Method 14 (Transformation rules)**.**

1. $\max \boldsymbol{c'x} \Longleftrightarrow -\min \boldsymbol{c'x}$
2. $\boldsymbol{a_i'x} = b_i \Longleftrightarrow \boldsymbol{a_i'x} \leq b_i$ and $\boldsymbol{a_i'x} \geq b_i$
3. $\boldsymbol{a_i'x} \leq b_i \Longleftrightarrow -\boldsymbol{a_i'x} \geq -b_i$

Definition 47 (Standard form)**.** A LP of the following form is said to be in *standard form*:

$$\begin{aligned} \min \;& \boldsymbol{c'x} \\ s.t. \;& \boldsymbol{Ax} = \boldsymbol{b} \\ & \boldsymbol{x} \in \mathbb{R}^n \end{aligned}$$

Method 15 (Transformation)**.** Transformation a LP into a LP in standard form via the following reduction rules:

1. Unrestricted variable: $x_j \Longleftrightarrow x_j^+ - x_j^-$ where $x^+, x^- \geq 0$.
2. Introduction of slack (surplus) variables: Given $\sum_{j=1}^n a_{ij} x_j \leq b_j$ (or $\geq b_j$), introduce $s_i \geq 0$ and rewrite $\sum_{j=1}^n a_{ij} x_j + s_i = b_j$ (or $-s_i$)

Theorem 37 (Solutions)**.** *Given a LP and feasible set $P = \{x \in \mathbb{R}^n | \boldsymbol{Ax} \geq \boldsymbol{b}\}$ the following possibilities exist:*

1. *$P = \emptyset$: No solutions exists.*
2. *$\boldsymbol{c'x}^* = -\infty$, i.e. $\inf\{\boldsymbol{c'x}^* | \boldsymbol{x} \in P\}$ does not exist: LP is unbounded and no solutions is optimal.*
3. *A unique optimal solution exists, that is, P is not empty and bounded (geometrically speaking: there exists an optimal solution which is an extreme point.)*
4. *Multiple optimal solutions exist and set of optimal solutions may be bounded or unbounded.*

5.2 Duality theory

Definition 48 (Primal problem, relaxed problem and dual problem)**.** A LP in standard form such that

$$\begin{aligned} \min\ & \boldsymbol{c}'\boldsymbol{x} \\ s.t.\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq 0 \end{aligned}$$

with optimal solution $\boldsymbol{x}^*$ is called *primal problem* (within duality theory). The following relaxed variant

$$\begin{aligned} g(\boldsymbol{p}) = \min & \, \boldsymbol{c}'\boldsymbol{x} + \boldsymbol{p}'(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \\ & \boldsymbol{x} \geq 0 \end{aligned}$$

with price vector $\boldsymbol{p}$ and penalty $\boldsymbol{p}'(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$ yields the *dual problem*

$$\begin{aligned} \max\ & \boldsymbol{p}'\boldsymbol{b} \\ s.t.\ & \boldsymbol{p}'\boldsymbol{A} \geq \boldsymbol{c}' \\ & \boldsymbol{x} \geq 0 \end{aligned}$$

Scheme.

$$\begin{array}{lll} \min \boldsymbol{c}'\boldsymbol{x} & & \\ \text{s.t. } \boldsymbol{a}_i'\boldsymbol{x} \geq b_i, & i \in M_1, \\ \quad \boldsymbol{a}_i'\boldsymbol{x} \leq b_i, & i \in M_2, \\ \quad \boldsymbol{a}_i'\boldsymbol{x} = b_i, & i \in M_3, \\ \quad x_j \geq 0, & j \in N_1, \\ \quad x_j \leq 0, & j \in N_2, \\ \quad x_j \text{ free}, & j \in N_3, \end{array} \qquad \begin{array}{lll} \max \boldsymbol{p}'\boldsymbol{b} & & \\ \text{s.t. } p_i \geq 0, & i \in M_1, \\ \quad p_i \leq 0, & i \in M_2, \\ \quad p_i \text{ free}, & i \in M_3, \\ \quad \boldsymbol{p}'\boldsymbol{A}_j \leq c_j, & j \in N_1, \\ \quad \boldsymbol{p}'\boldsymbol{A}_j \geq c_j, & j \in N_2, \\ \quad \boldsymbol{p}'\boldsymbol{A}_j = c_j, & j \in N_3. \end{array}$$

PRIMAL	minimize	DUAL	maximize
constraints	$\geq b_i$ $\leq b_i$ $= b_i$	variables	≥ 0 ≤ 0 free
variables	≥ 0 ≤ 0 free	constraints	$\leq c_j$ $\geq c_j$ $= c_j$

Example.

Primal		Dual	
min	$x_1 + 2x_2 + 3x_3$	max	$5p_1 + 6p_2 + 4p_3$
s.t.	$-x_1 + 3x_2 = 5$	s.t.	p_1 free
	$2x_1 - x_2 + 3x_3 \geq 6$		$p_2 \geq 0$
	$x_3 \leq 4$		$p_3 \leq 0$
	$x_1 \geq 0$		$-p_1 + 2p_2 \leq 1$
	$x_2 \leq 0$		$3p_1 - p_2 \geq 2$
	x_3 free,		$3p_2 + p_3 = 3.$

Theorem 38 (Duality)**.** *The optimal cost in the dual problem is equal to the optimal cost in the primal problem.*

Theorem 39. *The dual of the dual is the primal.*

Theorem 40 (Weak duality)**.** *If $\boldsymbol{x}$ is a feasible solution to the primal problem an $\boldsymbol{p}$ is a feasible solution to the dual problem, then $\boldsymbol{p'b} \leq \boldsymbol{c'x}$*

Theorem 41 (Strong duality)**.** *If a linear programming problem has an optimal solution, so does its dual, and the respective optimal costs are equal.*

Theorem 42 (Complementary slackness)**.** *Let $\boldsymbol{x}$ and $\boldsymbol{p}$ be feasible solutions to the primal and the dual problem, respectively. They are optimal if and only if:*

$$\begin{aligned} p_i(\boldsymbol{a}_i'\boldsymbol{x} - b_i) &= 0 \qquad \forall i \\ (c_j - \boldsymbol{p}'\boldsymbol{A}_j)\boldsymbol{x}_j &= 0 \qquad \forall j \end{aligned}$$

5.3 Mixed Linear Integer Programming

Definition 49 (Mixed Linear Integer Programming)**.** Recollect the notion of LP. A *mixed linear integer program* is of the following form:

$$\begin{aligned} \max \quad & \boldsymbol{c_1x_1} + \boldsymbol{c_2x_2} \\ s.t. \quad & \boldsymbol{A_1x_1} + \boldsymbol{A_1x_2} \leq \boldsymbol{b} \\ & \boldsymbol{x_1} \geq 0 \\ & \boldsymbol{x_2} \geq 0, \boldsymbol{x_2} \in \mathbb{Z} \end{aligned}$$

- $\boldsymbol{c_1}(\boldsymbol{c_2})$ is a n-dimensional (p-dimensional) row vector.,
- $\boldsymbol{x_1}(\boldsymbol{x_2})$ is a n-dimensional (p-dimensional) column vector,
- $\boldsymbol{A_1} \in \mathbb{R}^{m\times n}$, $\boldsymbol{A_2} \in \mathbb{R}^{m\times p}$ and $\boldsymbol{b} \in \mathbb{R}^{m\times 1}$.

Problem 27: Assignment Problem

Given n jobs, n persons and cost c_{ij} for assigning person i to job j, find an assignment with minimum cost.

Method 16 (MILP Formulation: AP)**.**

1. Variables: $x_{ij} \in \{0,1\}$, $\forall i,j \in \{1,\dots,n\}$

$$x_{ij} = \begin{cases} 1 & \text{if person } i \text{ is assigned to job } j, \\ 0 & \text{else} \end{cases}$$

2. Constraints: Each person i is assigned exactly one job:

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad \forall i \in \{1,\dots,n\}$$

Each job j is done by exactly one person:

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad \forall j \in \{1,\dots,n\}$$

3. Objective function: minimize the total assignment costs

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

Problem 28: The 0-1 Knapsack Problem

Given n possible projects with costs a_j for project j, estimated profit c_j for selecting project j and available budget B, find a subset of projects that maximizes the total profit while not exceeding the budget

Method 17 (MILP Formulation: 0-1KP)**.**

1. Variables: $x_j \in \{0,1\}, \forall j \in \{1,\ldots,n\}$

$$x_j = \begin{cases} 1 & \text{if project } j \text{ is selected,} \\ 0 & \text{else} \end{cases}$$

2. Constraints: The budget must not be exceeded:

$$\sum_{j=1}^{n} a_j x_j \leq B$$

3. Objective function: maximize the total profit

$$\max \sum_{j=1}^{n} c_j x_j$$

Problem 29: The Traveling Salesman Problem

Given a set $N = \{1,\ldots,n\}$ of cities and costs c_{ij} for traveling from city i to city j, find a tour with minimum traveling cost; where each city has to be visited exactly once and the salesman has to return to his starting city at the end.

Method 18 (MILP Formulation: TSP)**.**

1. Variables: $x_{ij} \in \{0,1\}, \forall i,j \in N, i \neq j$

$$x_{ij} = \begin{cases} 1 & \text{if the salesman travels directly from city } i \text{ to } j, \\ 0 & \text{else} \end{cases}$$

2. Constraints: The salesman leaves each city i exactly once:

$$\sum_{j \in N, j \neq i} x_{ij} = 1, \qquad \forall i \in N$$

The salesman arrives at each city j exactly once:

$$\sum_{i \in N, i \neq j} x_{ij} = 1, \qquad \forall j \in N$$

3. Objective function: minimize the total traveling costs

$$\min \sum_{i\in N} \sum_{\substack{j\in N\\ j\neq i}} c_{ij}x_{ij}$$

Treating subtours via

- Sequential Formulation [Miller, Tucker and Zemlin (1960)]
- Single-Commodity Flow Formulation [Gavish and Graves (1978)]
- Multi-Commodity Flow Formulation [Wong (1980)][21]

5.4 Branch-and-bound

Definition 50 (Primal bounds and dual bounds)**.** Lower bounds in maximization problems and upper bounds in minimization problems are called *primal bounds*. Upper bounds in maximization problems and lower bounds in minimization problems are called *dual bounds*[22].

Theorem 43. *Let $S = S_1 \cup \ldots \cup S_K$ be a decomposition of S into smaller sets, and let $z^k = \max\{\boldsymbol{cx}|\boldsymbol{x} \in S_k\}, \forall k \in \{1,\ldots,K\}$. Then $z = \max_k z^k$.*

Theorem 44. *Let $S = S_1 \cup \ldots \cup S_K$ be a decomposition of S into smaller sets, and let $z^k = \max\{\boldsymbol{cx}|\boldsymbol{x} \in S_k\}, \forall k \in \{1,\ldots,K\}$, $\overline{z}^k$ be an upper bound and $\underline{z}^k$ be a lower bound on z^k. Then $\overline{z} = \max_k \overline{z}^k$ is an upper bound and $\underline{z} = \max_k \underline{z}^k$ is a lower bound on z.*

6 Geometric Algorithms

6.1 Sweep Line

Problem 30: Segment Intersections

Set S of n line segments in the plane $\mathbb{R}^2$ decide whether any two segments in S intersect?

Method 19 (Sweep line: Segment Intersections)**.**

1. Observation: two segments separated by a (vertical) line cannot intersect
2. Observation: two segments stabbed by a (vertical) line may intersect iff there is such a line on which they are neighbors
3. sweep a vertical line ℓ (*sweep line*) across the plane from left to right and perform the intersection tests when necessary

Definition 51. Two segments s_1 and s_2 are *comparable* for a given sweep line ℓ if they both intersect ℓ but not in the (potential) intersection point of s_1 and s_2. In this case we define $s_1 <_\ell s_2$ if s_1 lies below s_2 on ℓ (or vice versa).

[21]Btw.: $P_{MCF} \subset P_{SCF} \subset P_{MTZ}$

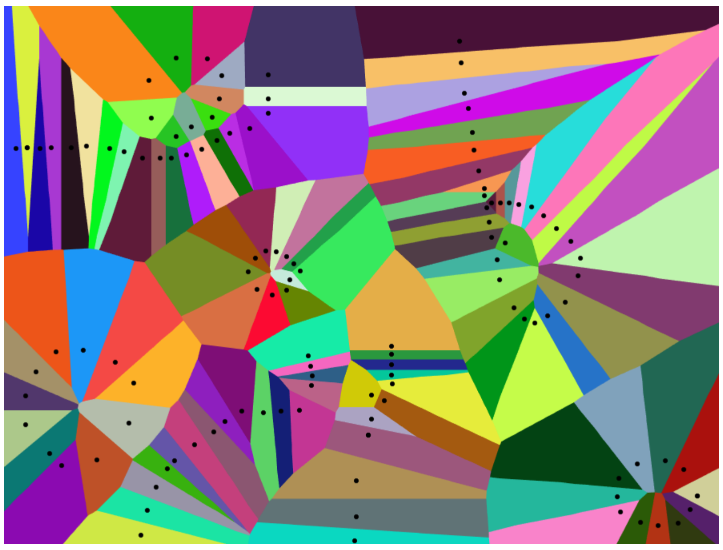
[22]In most cases dual bounds are obtained by relaxations: a difficult (maximization) IP is replaced by a simpler optimization problem with an optimal value at least as large as $\boldsymbol{z^*}$, e.g., by removing some constraints.

Algorithm 4: SWEEP LINE ALGORITHM: INTERSECT

```
$T \leftarrow \emptyset$                                              (sweep-line status)
$Q \leftarrow$ sorted list of endpoints from left to right                (event queue)
if two endpoints coincide then
    return true
end
foreach endpoint $p$ in $Q$ do
    if $p$ left endpoint of segment $s$ then
        $Insert(T, s)$
        if $s$ intersects $Above(T, s)$ then
            return true
        end
        if $s$ intersects $Below(T, s)$ then
            return true
        end
    end
    if $p$ right endpoint of segment $s$ then
        if $Above(T, s)$ intersects $Below(T, s)$ then
            return true
        end
        $Delete(T, s)$
    end
end
return false
```

where *sweep-line status* is data structure storing all objects intersected by the current sweep line (and their possible relationships) and *event queue* priority queue of all sweep-line events, i.e., x-coordinates at which the sweep-line status changes combinatorially.[23]

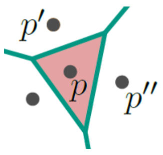
6.2 The Voronoi Diagram



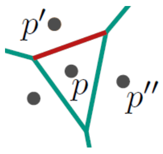
Definition 52 (Voronoi Diagram)**.** Let P be a set of points in the plane and let $p, p', p'' \in P$.

[23] $Above(T, s)$ and $Below(T, s)$ yields the segment directly above (below) s in T.

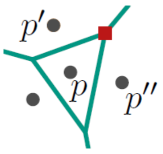
- *Voronoi cell*: $\mathcal{V}(\{p\}) = \mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq|, \forall q \in P - \{p\}\} = \bigcap_{q \neq p} h(p,q)$ where $h(p,q) = x \in \mathbb{R}^2 : |xp| < |xq|$ $h(q,p) = x \in \mathbb{R}^2 : |xq| < |xp|$.



- *Voronoi edge*: $\mathcal{V}(\{p,p'\}) = \{x : |xp| = |xp'| \text{ and } |xp| < |xq| \ \forall q \neq p, p'\}$.



- *Voronoi vertices*: $\mathcal{V}(\{p,p',p''\}) = \partial\mathcal{V}(p) \cap \partial\mathcal{V}(p') \cap \partial\mathcal{V}(p'')$ where ∂X denotes the boundary of X.



Theorem 45. *Let $P \in \mathbb{R}^2$ be a set of n points. The size of $Vor(P)$ is linear in $n = |P|$: It has at most $2n - 5$ vertices and at most $3n - 6$ edges.*

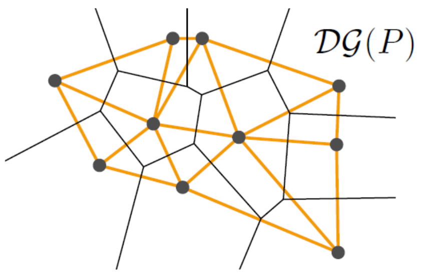
Definition 53. Let q be a point. Define $C_P(q)$ to be the largest empty circle (no points from P in the interior) with center q.

Theorem 46.

1. *A point $q \in \mathbb{R}^2$ is a Voronoi vertex iff $|\partial C_P(q) \cap P| \geq 3$.*
2. *The bisector[24] $b(p_i, p_j)$ of $p_i, p_j \in P$ defines a Voronoi edge iff $\exists q \in b(p_i, p_j)$ with $\partial C_P(q) \cap P = \{p_i, p_j\}$.*

Theorem 47 (Runtime: Compute $Vor(P)$)**.** *The $Vor(P)$ of a set P of n points in the plane can be computed in $\mathcal{O}(n \cdot log\, n)$ time.*

Definition 54 (Dual of Voronoi graph)**.** The graph $G = (P, E)$ with $E = \{pq | \ \mathcal{V}(p) \text{ and } \mathcal{V}(q) \text{ adjacent}\}$ is called the *dual graph of $Vor(P)$* and its embedding (straight line drawing) is called the *Delaunay graph $DG(P)$*.



[24] $b(p,q) = \{x \in \mathbb{R}^2 : |xp| = |xq|\}$

Remark 6. If we have $Vor(P)$ we can easily compute $DG(P)$ in time $\mathcal{O}(n)$. If all Voronoi vertices have degree 3, then $DG(P)$ is a *(Delaunay) triangulation* of P.

Corollary 6.

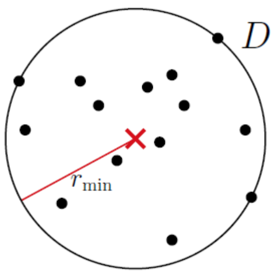
1. *$DG(P)$ is planar and crossing free.*
2. *Points $p, q \in P$ are connected by an edge in $DG(P)$ iff there exists an empty circle C through p and q, i.e. $p, q \in \partial C$ and $C^\circ = \emptyset$.*
3. *Points p, q, r are vertices of the same face in $DG(P)$ iff the circle through p, q, r is empty.*

6.3 Smallest enclosing disk

[25]

Problem 31

Given set P of n points in $\mathbb{R}^2$, find a disk D with minimum radius r that contains P.



Method 20 (Incremental algorithm: SED)**.** Let $P = \{p_1, p_2, \ldots, p_n\}$ and define $P_i := \{p_1, \ldots, p_i\}$ and let D_i be the smallest enclosing disk for P_i. Idea: Incrementally compute all disks D_i for $i = 2, \ldots, n$ by adding the points one-by-one and obtain the final solution $D = D_n$ where we make use of

1. SEDWithTwoPoints(P, q, r)
2. SEDWithPoint(P, q)
3. SEDisk(P)

Theorem 48 (Runtime: SED)**.**

1. SED *is in $\mathcal{O}(n)$ (best case) if never calling the else-case.*
2. SED *is in $\mathcal{O}(n^3)$ (worst case) if always calling the else-case.*
3. *The randomized incremental algorithm for* SED *is in $\mathcal{O}(n)$.*

[25]Assume your village needs to build a new fire station and the time needed to reach any house should be smallest possible. What is the best location?

Problem	**Kernel**	**Poly Kernel**	**FPT**	**Parameterized by**	**Based on graphs**
Parameterized Vertex Cover	Yes	$\mathcal{O}(k^2)$	Yes $\mathcal{O}(2^k \cdot n)$	k	Yes
Independent Set	Yes		Yes	Treewidth	Yes
Independent Set	No	–	No	–	Yes
Independent Set	Yes		Yes	Treewidth	Yes
Parameterized Independent Set	No	–	No	k	Yes
Parameterized Independent Set	Yes	$poly(k)$	Yes $\mathcal{O}(4^k \cdot n)$	Vertex Cover	Yes
Parameterized Independent Set (Deg ≤ 3)	Yes	–	Yes $\mathcal{O}(4^k \cdot n)$	k	Yes
Parameterized Dominating Set	No	–	No	–	Yes
Parameterized Dominating Set (Deg ≤ 3)	Yes		Yes $\mathcal{O}(4^k \cdot n)$	k	Yes
Parameterized Clique	No	–	No	k	Yes
Parameterized MaxSat	Yes	$\mathcal{O}(k^2)$	Yes	k	No
Hitting Set	No	–	No	k	No
d-Hitting Set (for fixed d)	Yes	$\mathcal{O}(k^d)$	Yes	k	No
Hamiltonian Cycle	Yes	$poly(k)$	Yes	Vertex Cover	Yes
Chromatic Number	Yes	–	Yes	Vertex Cover	Yes
SAT Using Deletion to Unit Clauses	Yes	–	Yes	k	No

A little overview

»And how many hours a day did you do lessons?« said Alice, in a hurry to change the subject.

»Ten hours the first day,« said the Mock Turtle: »nine the next, and so on.«

»What a curious plan!« exclaimed Alice.

»That's the reason they're called lessons,« the Gryphon remarked: »because they lessen from day to day.«