Exact Algorithms for NP-hard problems

Advanced Algorithms: Part 2, Lecture 6

Today: Preprocessing

- Homework on DP over tree decomposition
- Feedback on lab assignment 3
- Preprocessing
  - Kernelization for vertex cover
  - FPT and the Complexity Hierarchy
  - Subset sum & knapsack
- Wrap-up and Q&A

Mathijs de Weerdt

TU Delft

Weighted independent set over nice tree decomposition

Given a graph $G = (V, E)$ with vertex weights $w_v > 0$, we consider the problem of finding an independent set $S \subseteq V$ with the maximum total weight, i.e., $w(S) := \sum_{v \in S} w_v$ is maximized.

Let also a *nice* tree decomposition $(Tr = (T, F), \{V_t : t \in T\})$ of G be given.

(a) (1 point) Define for a leaf node t of Tr the maximum total weight $OPT_t(U)$ for each possible subset U of the bag V_t.

(b) (1 point) Define for a forget node t of Tr the maximum total weight $OPT_t(U)$ for each possible independent subset U of the bag V_t.

(c) (2 points) Define for an introduce node t of Tr the maximum total weight $OPT_t(U)$ for each possible independent subset U of the bag V_t.

(d) (1 point) Define for a join node t of Tr the maximum total weight $OPT_t(U)$ for each possible independent subset U of the bag V_t.

(e) (1 point) Suppose that for all tree nodes t and for all arguments U the values defined in the above recursive function have been stored in a table $M_t[U]$. Explain how to determine the maximum total weight of an independent set of G from this table.

Weighted independent set over nice tree decomposition

Leaf: $|V_t| = 1$

Q. How to define the values for…

$OPT(t,U) = \ldots$

$OPT(t,\emptyset) = \ldots$

as before, OPT(t,U) is maximum value of an independent set *consistent* with U

TU Delft

Weighted independent set over nice tree decomposition

Leaf: $|V_t| = 1$
 $OPT(t,U) = w(U)$
 $OPT(t,\emptyset) = 0$

as before, $OPT(t,U)$ is maximum value of an independent set *consistent* with U

Forget: V_t has one less vertex v than in child t'
Q. How to define the value for every independent set $U \subseteq V_t$?
$OPT(t,U) = \ldots$

Weighted independent set over nice tree decomposition

Leaf: $|V_t| = 1$

$OPT(t,U) = w(U)$

$OPT(t,\emptyset) = 0$

as before, $OPT_t(U)$ is maximum value of an independent set *consistent* with U

Forget: V_t has one less vertex v than in child t'

$OPT(t,U) = \max\{ OPT(t',U), OPT(t',U \cup \{v\}) \}$

Introduce: V_t has one vertex v more than in child t'

$$OPT_t(U) = \begin{cases} \cdots & \text{if } v \notin U \\ \cdots & \text{if } v \in U \text{ but has no neighbors in } U \\ \cdots & \text{if } U \text{ contains } v \text{ and a neighbor} \end{cases}$$

TU Delft

Weighted independent set over nice tree decomposition

as before, $OPT_t(U)$ is maximum value of an independent set *consistent* with U

Leaf: $|V_t| = 1$

$OPT(t,U) = w(U)$

$OPT(t,\emptyset) = 0$

Forget: V_t has one less vertex v than in child t'

$OPT(t,U) = \max\{ OPT(t',U), OPT(t',U \cup \{v\}) \}$

Introduce: V_t has one vertex v more than in child t'

$$OPT_t(U) = \begin{cases} OPT_{t'}(U) & \text{if } v \notin U \\ OPT_{t'}(U \setminus \{v\}) + w(v) & \text{if } v \in U \text{ but has no neighbors in } U \\ -\infty & \text{if } U \text{ contains } v \text{ and a neighbor} \end{cases}$$

Join: two children t_1 and t_2 with $V_t = V_{t1} = V_{t2}$

$OPT(t,U) = \ldots$

Weighted independent set over nice tree decomposition

as before, OPT(t,U) is maximum value of an independent set *consistent* with U

Leaf: $|V_t| = 1$

$OPT(t,U) = w(U)$

$OPT(t,\emptyset) = 0$

Forget: V_t has one less vertex v than in child t'

$OPT(t,U) = \max\{ OPT(t',U), OPT(t',U \cup \{v\}) \}$

Introduce: V_t has one vertex v more than in child t'

$$OPT_t(U) = \begin{cases} OPT_{t'}(U) & \text{if } v \notin U \\ OPT_{t'}(U \setminus \{v\}) + w(v) & \text{if } v \in U \text{ but has no neighbors in } U \\ -\infty & \text{if } U \text{ contains } v \text{ and a neighbor} \end{cases}$$

Join: two children t_1 and t_2 with $V_t = V_{t1} = V_{t2}$

$OPT(t,U) = OPT(t_1, U) + OPT(t_2,U) - w(U)$

Weighted independent set over nice tree decomposition

as before, OPT(t,U) is maximum value of an independent set *consistent* with U

Leaf: $|V_t| = 1$
OPT(t,U) = w(U)
OPT(t,∅) = 0

Forget: V_t has one less vertex v than in child t'
OPT(t,U) = max{ OPT(t',U), OPT(t',U∪{v}) }

Introduce: V_t has one vertex v more than in child t'

$$OPT_t(U) = \begin{cases} OPT_{t'}(U) & \text{if } v \notin U \\ OPT_{t'}(U \setminus \{v\}) + w(v) & \text{if } v \in U \text{ but has no neighbors in } U \\ -\infty & \text{if } U \text{ contains } v \text{ and a neighbor} \end{cases}$$

Join: two children t_1 and t_2 with $V_t = V_{t1} = V_{t2}$
$OPT(t,U) = OPT(t_1, U) + OPT(t_2,U) - w(U)$

Root: $\max_{U \subseteq V_r}$\{ OPT(r,U) \} where no vertices in U are neighbors

TU Delft

Lab Assignment 3: find hitting set H of size at most k

Suppose we are given m minimal conflict sets $B_1,\ldots,B_m$. (All subsets of universe *A*.)

These represent m different situations where each conflict set represents a possible explanation of a malfunction.

Find a small set of possible problematic components from *A* which explains all malfunctions.

Formally, the question is whether there exists a subset of components $H \subseteq A$ of size k such that each minimal conflict set has at least one element in common with this H, and the size of H is at most k.

Lab Assignment 3: find hitting set H of size at most k

Given conflict sets $B_1,\ldots,B_m$, each is a subset of A, and of size at most c
Decide whether $H \subseteq A$ of size at most k exists with a nonempty intersection with each B_i

Q. What is a basic decision towards a bounded search tree algorithm?
A. include an item in *H*. Take one from one of the conflict sets (of course!)

Q. What are the options for this decision?
A. select a conflict set B_i, branch for each of the c items in this conflict set

Q. How to express solution of whole problem as solution to subproblems?
A. Recursive definition on m remaining conflict sets with k left to choose:

1. return true if $m=0$; return false if $m>0$ and $k=0$
2. compute each subproblem $(m-1, k-1)$ for each item from B_m
3. return true if one of these returns true

Q. What is an upper bound on the runtime?
A. $O^*(c^k)$

TU Delft

Preprocessing NP-hard problems

- *kernelization* for vertex cover
- *preprocessing* for subset sum and knapsack

Preprocessing NP-hard problems

General idea

- analyze and restructure (reduce) input to solve problem more efficiently
- provide bounds on input size and thereby better bounds on the runtime

This idea will be put forward by giving examples on

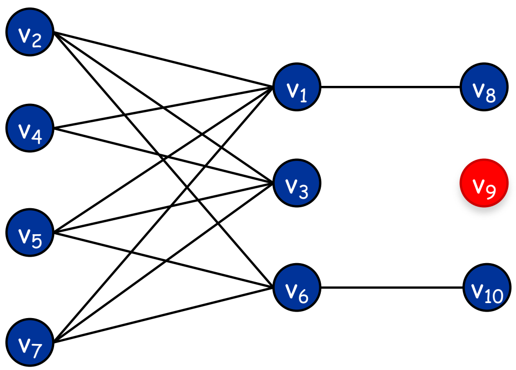
- Vertex cover (kernelization) from Huffner, Niedermeier, Wernicke '07
- Sub-set sum
- Knapsack

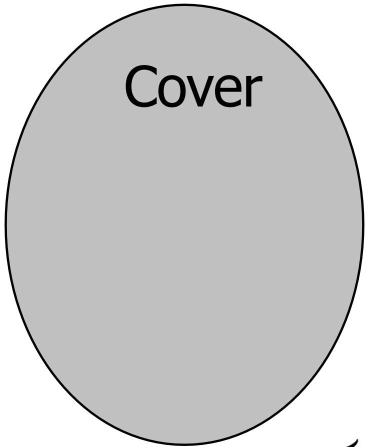
from Woeginger survey, p197-198

Preprocessing Vertex cover (decision problem)

Q. What do we know about vertex 9?
A. Rule 1: isolated vertices will never be part of the cover

Q. What do we know about vertices 8 and 10?
A. Rule 2: put the neighbors of degree=1 vertices in the cover (k' :=k-1)

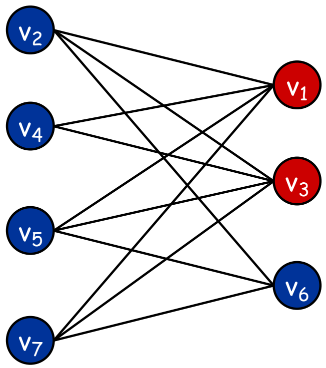




Preprocessing Vertex cover

Q. When looking for a vertex cover of size at most k, what do we know about vertices with *more than k* neighbors (e.g. for k=3, vertex 1 and 3)?
A. Rule 3: vertices with degree k+1 or greater should be in the cover (otherwise all k+1 neighbors should be in the cover) (k' :=k-1)

Q. What is the maximum possible size of a remaining graph after applying these rules *when a vertex cover exists of size at most k'*? (3 min)
Q_1. What is the maximum degree of vertices in the remaining graph?
A. k' , because of rule 3



Analysis of vertex cover rules

So: max degree in remaining graph is k'…
Q_2. Give an upper bound on the number of *edges* in the *remaining* graph (if a vertex cover of size at most k' exists, how many edges can we cover)?
A. at most k'^2 edges, since each vertex has degree at most k' and we can have *at most k' vertices in the cover*

Q_3. Give an upper bound on the number of vertices in the remaining graph.
A. Kernel has also at most k'^2 vertices, because every vertex has degree 2 or more. (Because of Rule 2.)
So, Rule 4: if kernel has more than k'^2 vertices, return "No".

Size of the remaining graph (kernel) can thus solely be expressed in $k' \leq k$ (instead of n and m), namely k'^2, so runtime of *trivial* algorithm becomes now $O^*(2^{k' \cdot k'})$ instead of $O^*(2^n)$
… or $O^*(2^{k'})$ or $O^*(1.47^{k'})$ if combined with bounded search trees.

Kernelizable

Def. A decision problem with input (I,k) where

- I is the instance
- k is the parameter

is *kernelizable* if every such input can be reduced to an instance (I',k') s.t.:

instance size is bounded by k

1. $k' \leq k$
2. $|I'|$ is smaller or equal to g(k) for some function g only depending on k
3. (I',k') has a solution if and only if (I,k) has one, and
4. the reduction from (I,k) to (I',k') must be computable in poly-time

Q. Given the previous analysis, what is $|I'|$ here?
A. k'^2 (which is indeed smaller than $g(k)=k^2$)

Summary: Kernelization

General idea

- find (polynomial-time) rules to reduce input:
  - remove irrelevant parts
  - solve the “easy” parts of the problem and remove those
  - bound size of reduced input (*kernel*)
- run costly algorithms (e.g., search trees) only on kernel

Rules can also be used for other types of algorithms (than exact)

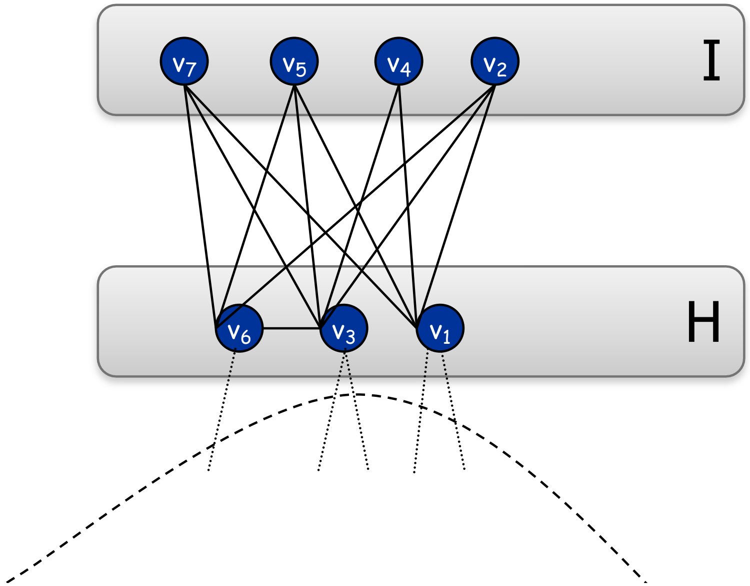
- Approximation algorithms
- Local search techniques
- Randomized algorithms

Kernelization using crown structures (for vertex cover)

Idea. Generalize Rule 2, about vertices with one edge

- an independent set of vertices I (no two vertices in I are connected)
- the set H of all adjacent vertices

Q. How can we cover edges in $I \cup H$?



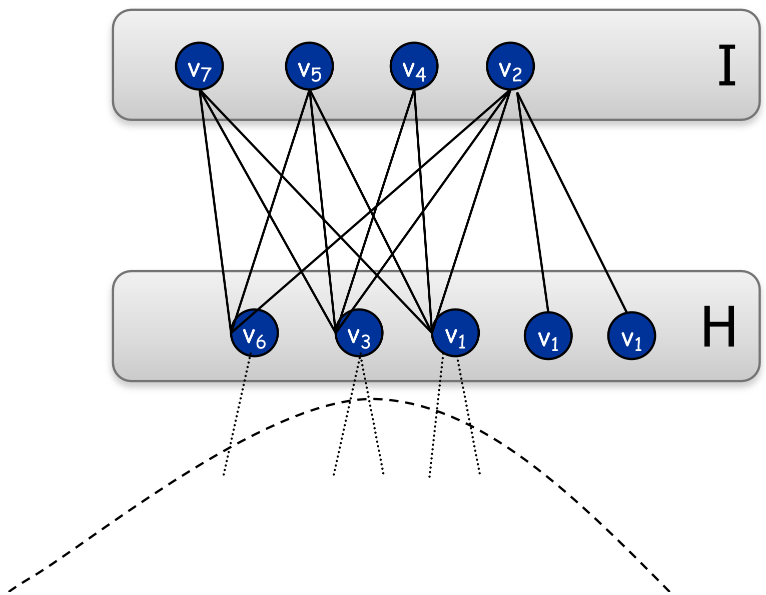
Kernelization using crown structures (for vertex cover)

Idea. Generalize Rule 2, about vertices with one edge

- an independent set of vertices I (no two vertices in I are connected)
- the set H of all adjacent vertices

Q. How can we cover edges in I∪H?

A. Here we cannot know for sure what to do… H seems too large.



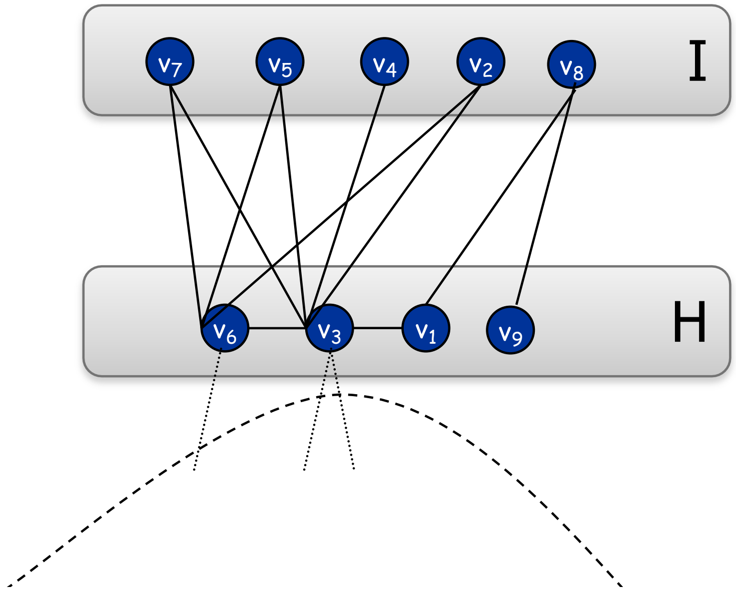
Kernelization using crown structures (for vertex cover)

Idea. Generalize Rule 2, about vertices with one edge

- an independent set of vertices I (no two vertices in I are connected)
- the set H of all adjacent vertices

Q. How can we cover edges in I∪H?

A. H is small, but v_8 has more neighbors than v_9.

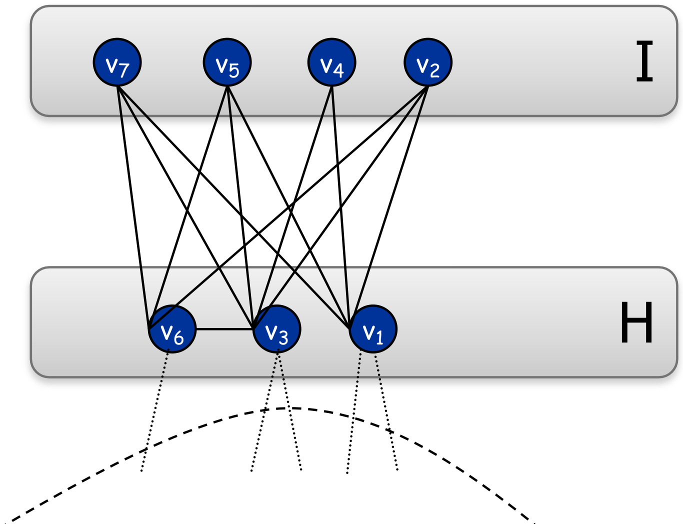


Kernelization using crown structures (for vertex cover)

Idea. Generalize Rule 2, about vertices with one edge
Identify a so-called *crown structure*:

- an independent set of vertices I (no two vertices in I are connected)
- the set H of all adjacent vertices
- the maximum matching in the bipartite graph I∪H should be size |H|

Claim. There is a min cover that includes all vertices H.



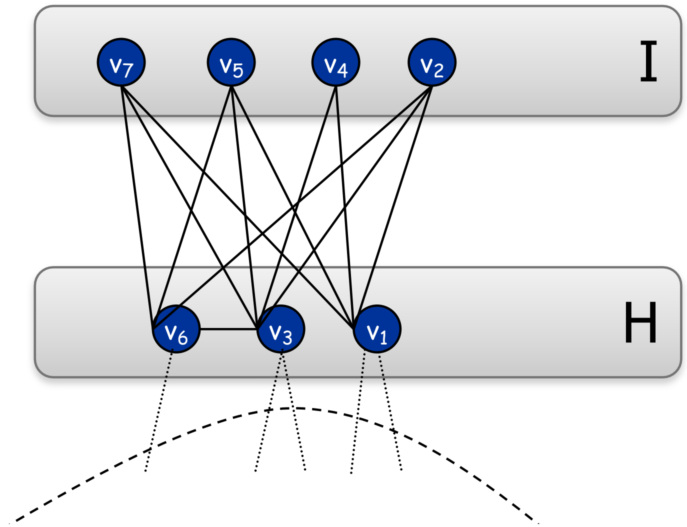
maximum matching = largest set of edges where no two edges share an endpoint

Kernelization using crown structures (for vertex cover)

Idea. Generalize Rule 2, about vertices with one edge
Identify a so-called *crown structure*:

- an independent set of vertices I (no two vertices in I are connected)
- the set H of all adjacent vertices
- the maximum matching in the bipartite graph I∪H should be size |H|

Claim. There is a min cover that includes all vertices H.
Pf. At least |H| to cover all edges in crown, H is sufficient as I is independent set. |H| is *minimum* cover because max matching size |H|.

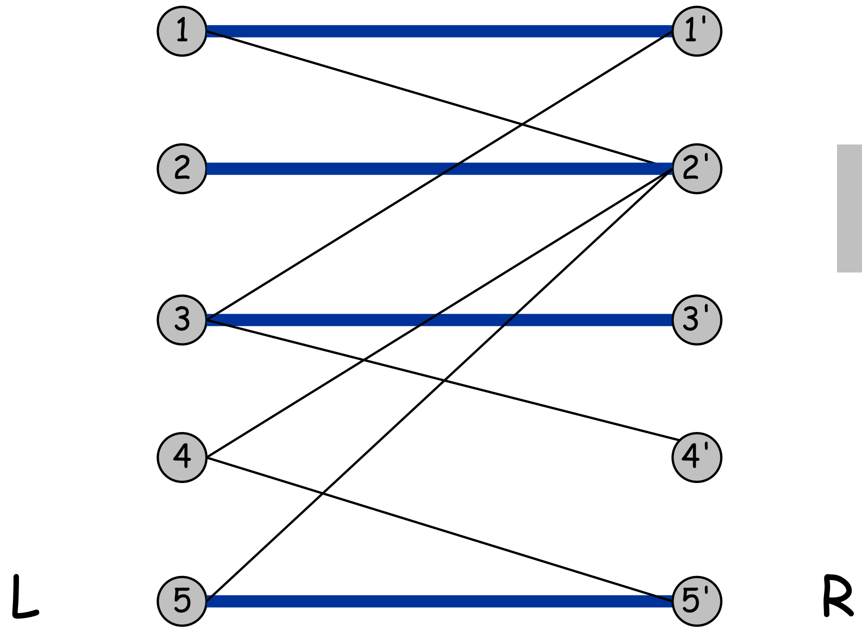


Recapture: Bipartite Matching

Bipartite matching.

- Input: undirected, bipartite graph $G = (L \cup R, E)$.
- $M \subseteq E$ is a matching if each node appears in *at most* one edge in M.
- Max matching: find a max cardinality matching.

Q. How can a max matching be found (efficiently)?



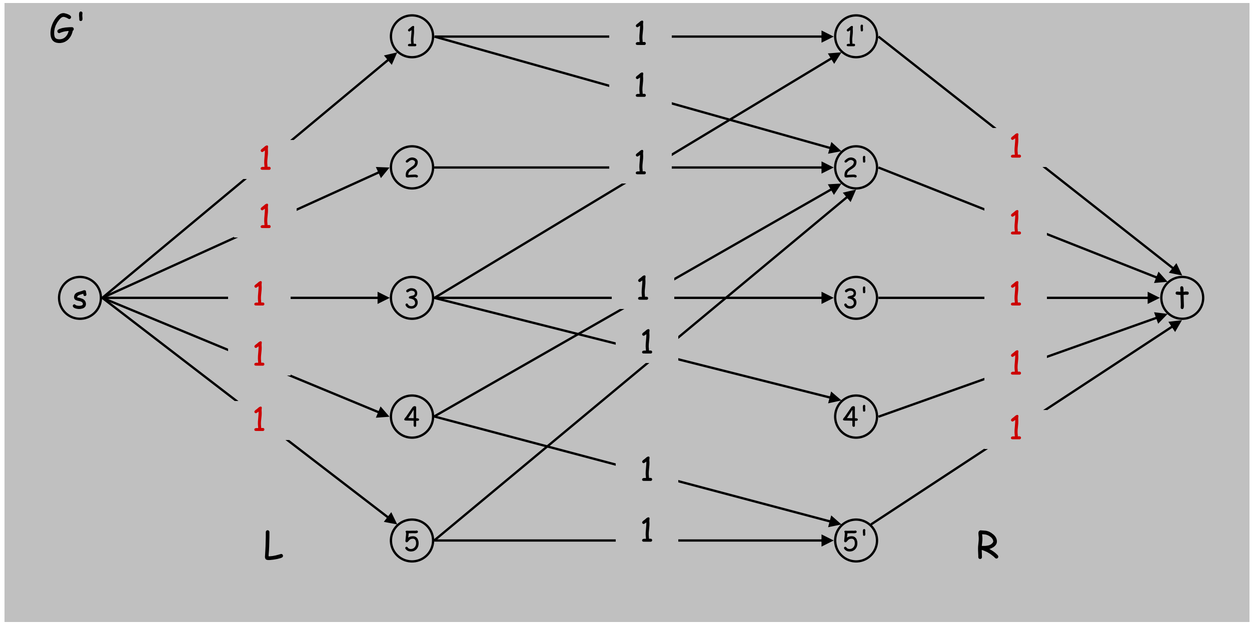
max matching

1-1', 2-2', 3-3' 4-4'

Recapture: Bipartite Matching

Max flow formulation.

- Create digraph G' = (L ∪ R ∪ {s, t}, E').
- Direct all edges from L to R, and assign unit (or infinite) capacity.
- Add source s, and unit capacity edges from s to each node in L.
- Add sink t, and unit capacity edges from each node in R to t.
- Max flow represents max matching



Kernelization using crown structures

Crowns can be found in polynomial time by maximum matchings.

Generalized Rule 2′

1. Remove crown ($I \cup H$) from graph.
2. Reduce k by $|H|$: $k' = k - |H|$.

Thm. A graph that is crown-free and has vertex cover of size at most k', can contain at most 3k' vertices. (Proof is a bit complicated, not part of course.)

Consequently, kernel (size of problem instance) now *linear* in parameter k', so runtime of *trivial* algorithm is $O^*(2^{3k'})$, but often better in practice.

Note. Using Nemhauser-Trotter '75, a kernel of size 2k' can be found by computing maximum matchings.
It is very unlikely that there exists a smaller kernel (Khot, Regev, 2003).

Kernelization, Fixed-Parameter Tractability, and the Complexity Hierarchy

Kernelizable and FPT

Def. A decision problem with input (I,k) where, I is the instance, k is the parameter is *kernelizable* if every such input can be reduced to an instance (I',k') s.t.:

1. $k' \leq k$
2. |I'| is smaller or equal to g(k) for some function g only depending on k
3. (I',k') has a solution if and only if (I,k) has one, and
4. the reduction from (I,k) to (I',k') must be computable in poly-time

A problem instance (I,k) for vertex cover is thus kernelizable to (I',k') where |I'| is O(3k') and $k' \leq k$.

Q. Is it FPT?

Thm. A problem is *fixed-parameter tractable* if and only if it is *kernelizable*.

Not very useful theorem:

- upper bounds obtained from kernelizations are often not very good
- given an FPT-algorithm, no *constructive* way of obtaining a kernelization

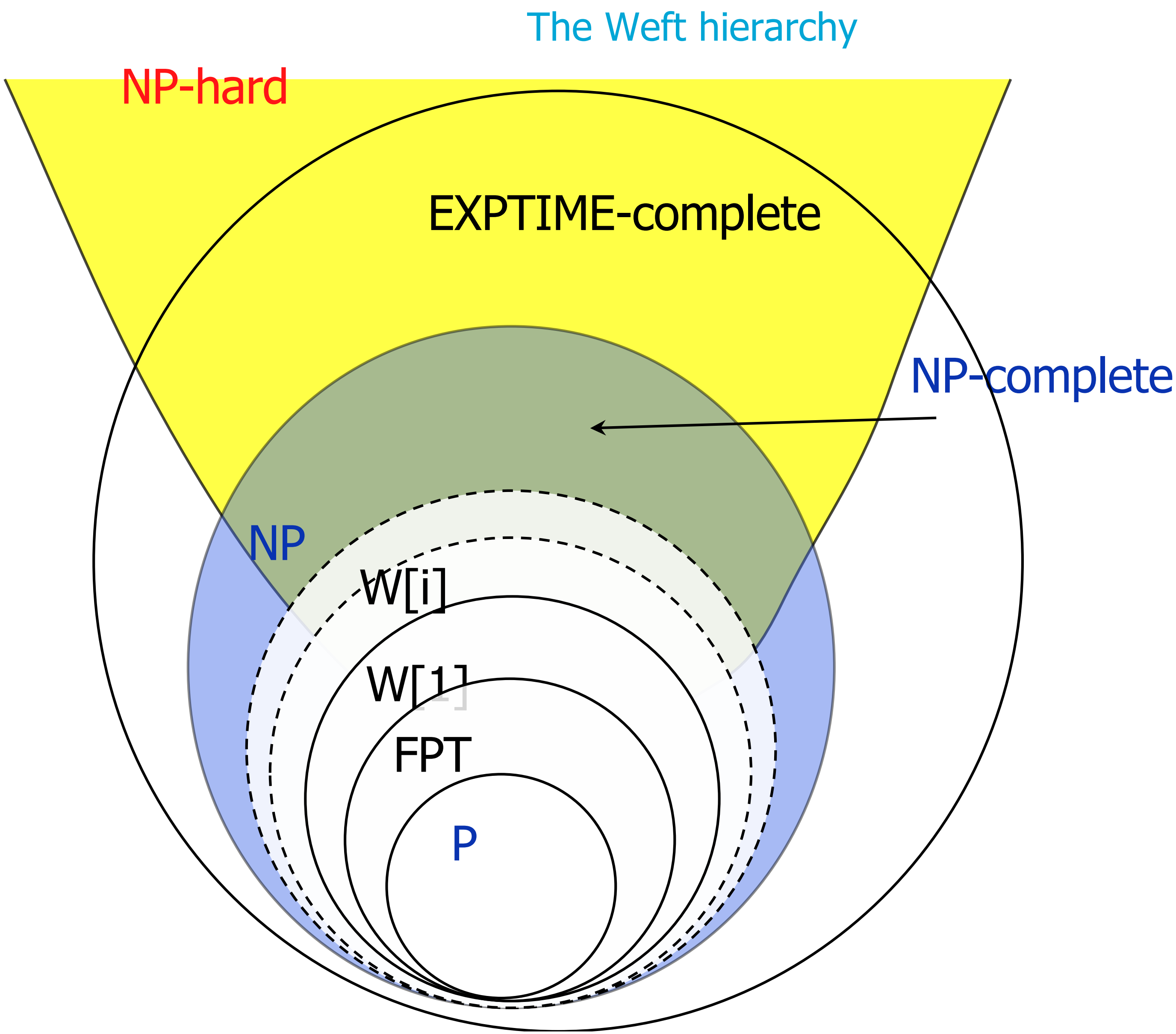
Fixed parameter tractable and NP

Q. Which problems are FPT (and not in P)?

- Vertex Cover
- Max Sat
- Dominating Set for planar graphs (see paper [2])
- Feedback Vertex Set

Q. Which problems are *not* FPT?

Answered by introducing new complexity classes: the Weft hierarchy W[i]


The Weft hierarchy
NP-hard
EXPTIME-complete
NP-complete
NP
W[i]
W[1]
FPT
P

Fixed parameter tractable and NP

Problems that are *not* FPT:

- W[1]-complete problems:
  - Weighted 3SAT (Given a 3SAT formula, does it have a satisfying assignment of Hamming weight k (i.e. with k variables set to true)?)
  - Clique (does the graph G=(V,E) contain a clique of at least size k?)
  - Independent Set (does G contain an independent set of at least size k?)
- W[2]-complete problems:
  - Dominating Set (is there a set $S \subseteq V$ of at most size k such that each vertex is in S or has a neighbor in S?)

Membership proven by *FPT reductions*:

A problem A with parameter *k is FPT reducible* to a problem B with parameter k' if there is a reduction such that $k' \leq g(k)$ for a computable function g().

- No one has been able to prove that W[1]-hard problems are FPT, or that W[i+1] =W[i] for any i, so it is believed that

$$P \subset FPTAS \subset FPT \subset W[1] \subset W[2] \subset \ldots \subset W[P] \subset NP$$

Preprocessing of Subset Sum and Knapsack

Preprocessing can take many forms

- Subset Sum
- Knapsack (exact algorithm with the best known upper bound!)

Recapture: Knapsack Problem

Knapsack problem.

- Given n objects and a "knapsack."
- Item i weighs $w_i > 0$ kilograms and has value $v_i > 0$.
- Knapsack has limit of W kilograms.
- Goal: fill knapsack so as to maximize total value.

- Best we can do here is select { 3, 4 }
  - attains 40

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

TU Delft

Recapture: Dynamic Programming for Knapsack

Recursively define value of optimal solution:
Def. OPT(i, w) = max profit subset of items 1, …, i with weight limit w.

- Case 1: OPT does not select item i.
  - OPT selects best set out of { 1, 2, …, i-1 } using weight limit w

- Case 2: OPT selects item i.
  - new weight limit = $w - w_i$
  - OPT selects best set out of { 1, 2, …, i–1 } using this new weight limit

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{\, OPT(i-1,w),\ \ v_i + OPT(i-1,w-w_i)\,\} & \text{otherwise} \end{cases}$$

$\Theta(n\ W) = \Theta(n\ 2^{\log W}) = O^*(2^{|x|})$ where $|x| = \log W$ = input size
"Pseudo-polynomial."

Subset Sum

CD-fitting problem. Given a set of songs. Select a subset of songs that fits exactly on a CD.

Is also used in Merkle-Hellman cryptosystem, and as a subproblem of TSP, bin-packing, and knapsack.

Subset Sum
Given
- integers $a_1, \ldots, a_n$, and
- an integer S

Decide
- whether there exists a subset of the integers a_i that sum up to S

Subset Sum

Given

- integers $a_1, \ldots, a_n$, and an integer S

Decide

- whether there exists a subset of the integers a_i that sum up to S

Q. E.g., given 17, 3, 5, 11, 13, 7 and S=21, does such a subset exists?
A. Yes, {3, 5, 13} or {3, 11, 7}

Q. What is the runtime of a trivial algorithm?
A. Try each subset, so $O^*(2^n)$

Towards Preprocessing Subset Sum: Toy Problem 1

Given

- *two* integer sequences $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$, and
- an integer S

Decide

- whether there exist x_i and y_j such that $x_i+y_j = S$

Q. E.g., given 17, 3, 5 and 11, 13, 7 and S=18, does such a pair exists?
A. Yes, $x_3=5$ and $y_2=13$

Q. What is the runtime of a trivial algorithm?
A. Try each pair, so $O(n^2)$

Q*. Any ideas for preprocessing the data (useful when n is large)?
A. sorting: if one of the arrays is sorted, O(n log n):
for each x_i search for $S-x_i$ in the other sequence

Preprocessing: Subset Sum

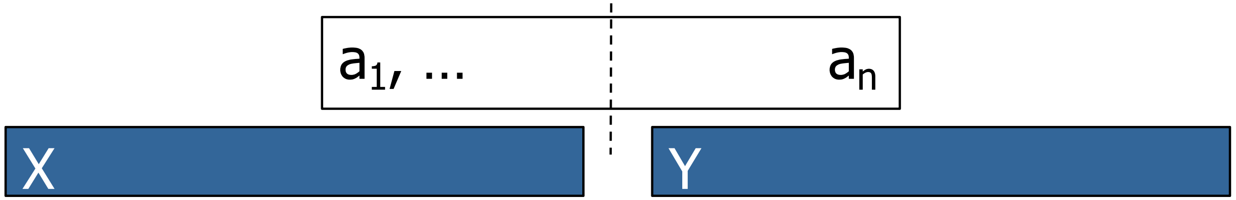
Given

- integers $a_1, \ldots, a_n$, and an integer S

Decide

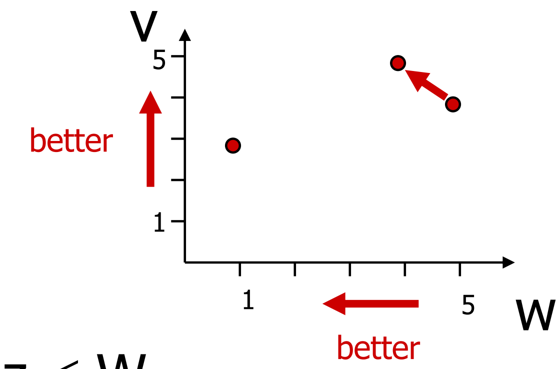
- whether there exists a subset of the a_i that sum up to S

Idea. Divide (once) & preprocess by sorting some subproblems.



1. split $a_1, \ldots, a_n$ in two sets of size $n/2$
2. let X be set of all $2^{n/2}$ subset sums (brute force) of first half, and Y of second half
3. apply $O(k \log k)$ algorithm for Toy Problem 1:
   1. sort Y in $O^*(2^{n/2} \cdot \log 2^{n/2})$
   2. search for each $S-x_i$ in Y also in $O^*(2^{n/2} \cdot \log 2^{n/2})$
   3. $O^*(2^{n/2} \cdot \log 2^{n/2}) = O^*(2^{n/2} \cdot n) = O^*(2^{n/2}) = O^*((2^{1/2})^n) = O^*(1.415^n)$

Towards preprocessing Knapsack: Toy Problem 2 (in P)



Given
- n (weight,value) points $(w_1, v_1), \ldots, (w_n, v_n)$
- n numbers $z_1, \ldots, z_n$
- a number $W>0$

Decide
- for every z_j the largest value v_i such that $w_i+z_j \leq W$

Q. E.g., given (1,3), (4,5) and (5,4) with z: 2, 4, 5 and W=6. Give for every z_j the largest value v_i such that $w_i \leq W-z_j$.
A. For z_1: $v_2=5$ and for z_2: $v_1=3$ and for z_3: $v_1=3$
Q. What is the runtime of a trivial algorithm?
Q. Any ideas for preprocessing the data?
A. 1. throw away dominated points, such as (5,4) in the example above
2. sort remaining points on w coordinate
3. search in this array for the largest w_i less than or equal to $W-z_j$

This takes O(n log n).

Knapsack Problem: Running Time

Runtime for dynamic programming solution.
$\Theta(n\ W) = \Theta(n\ 2^{\log W}) = O^*(2^{|x|})$ where $|x| = \log W$ = input size

- "Pseudo-polynomial."
- Not polynomial in input size, same worst-case as trivial algorithm
- Decision version of Knapsack is NP-complete.

Idea. Use a similar approach to preprocess the data (algorithm $O^*(1.415^n)$)

1. split items in two sets of size n/2
2. let X be set of all $2^{n/2}$ (brute force) compound items I of first half with combined value v_I and combined weight w_I, and
3. let Y be set of $2^{n/2}$ compound items J of second half
4. remove dominated points from Y and sort on w_J as in Toy Problem 2
5. for each $(w_I, v_I) \in X$ search for $W - w_I$ in Y; take maximal $v_I + v_J$
6. $O^*(2^{n/2} \cdot \log 2^{n/2}) = O^*(2^{n/2} \cdot n) = O^*(2^{n/2}) = O^*((2^{1/2})^n) = O^*(1.415^n)$

Q*. Can an exact solution can be found more efficiently?

Knapsack Problem: Running Time

Note. It is an open problem whether an algorithm of $O^*(c^n)$ with $c < 1.415$ exists. (Since 1974 when Horowitz and Sahni proposed the previous preprocessing trick.)

Removing dominated points subroutine

```
Input: list L of (weight,value) points (w₁, v₁),…,(wₙ,vₙ)
Output: subset L′ of undominated points i.e.:
no other point i in L′ has higher/equal value vᵢ with
  lower/equal weight wᵢ

 1. w:= ∞
 2. sort L decreasing on v (first), and for equal v increasing
    on w (second)
 3. for each (wᵢ,vᵢ) in L
   1. if wᵢ < w then
     1. w := wᵢ
     2. add (wᵢ,vᵢ) to L′
 4. return L′
```

1-Slide Summary on Kernelization and Preprocessing

General idea kernelization

- find (polynomial-time) rules to reduce input:
  - remove irrelevant parts
  - solve the "easy" parts of the problem and remove those
  - bound size of reduced input (*kernel*)
- run costly algorithms (e.g., search trees) only on kernel

Examples

- Kernelization for Vertex cover:
  - 3 "easy" rules: kernel of size k'^2
  - crown structure: kernel of size $3k'$
- Preprocessing for knapsack:
  - sort input and use domination

- FPT if and only if kernelizable.

TU Delft

Wrap-Up

- Summary
- Q&A
- Next?

High-level overview on Part 2: Techniques for solving NP-hard problems exactly

- (complete/bounded) search trees (Ch.10.1 + [2,3])
  - independent set: $O^*(1.38^n)$
  - 3-SAT:$O^*(1.84^n)$
  - vertex cover: $O^*(1.47^k)$
- dynamic programming (Ch.10.2-10.3 + [2])
  - traveling salesman: $O^*(2^n)$
  - scheduling with precedences: $O^*(2^n)$
- tree decomposition (Ch.10.4)
  - (weighted) independent set, vertex cover: $O^*(4^{w+1})$
  - nice tree decomposition, treewidth, properties
- decision diagrams: weighted independent set
- preprocessing:
  - knapsack (sorting): $O^*(1.415^n)$
  - kernelization: rules: $O^*(2^{k'*k'})$ with crown structure: $O^*(2^{3k'})$

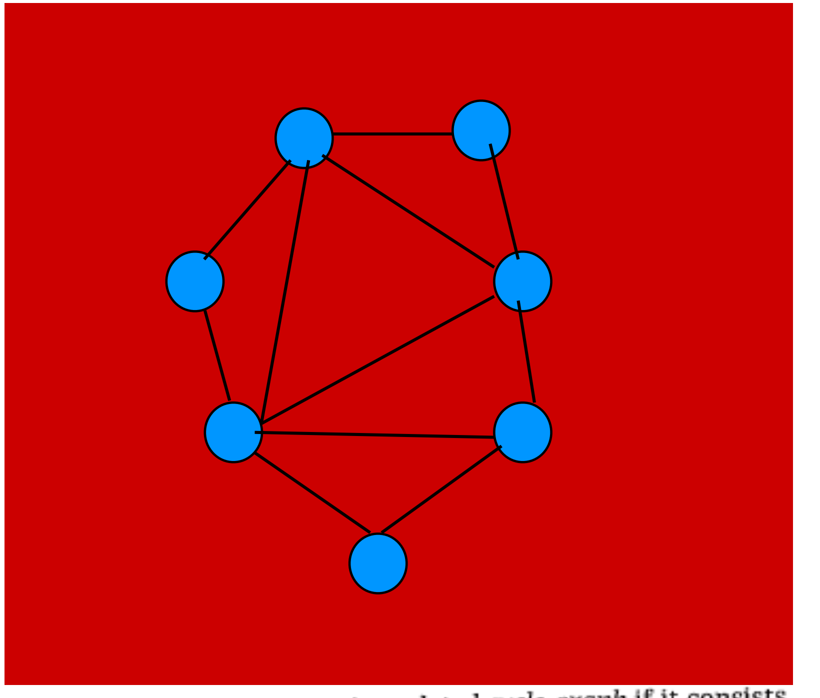
TU Delft

Q: triangulated cycle graph

Q. How is this triangulated property defined here?

A.

1. S = all points in the plane that are also on vertex or edge
2. Remove S
3. Consider bounded components: each is bordered by exactly three edges.



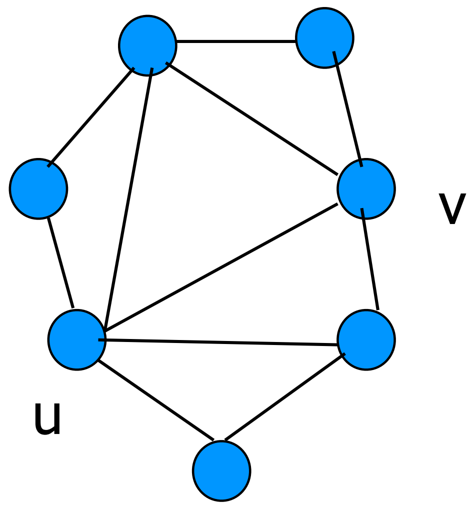
Q. How to construct a tree decomposition?

4. We say that a graph $G = (V, E)$ is a *triangulated cycle graph* if it consists of the vertices and edges of a triangulated convex n-gon in the plane—in other words, if it can be drawn in the plane as follows.

The vertices are all placed on the boundary of a convex set in the plane (we may assume on the boundary of a circle), with each pair of consecutive vertices on the circle joined by an edge. The remaining edges are then drawn as straight line segments through the interior of the circle, with no pair of edges crossing in the interior. We require the drawing to have the following property. If we let S denote the set of all points in the plane that lie on vertices or edges of the drawing, then each bounded component of the plane after deleting S is bordered by exactly three edges. (This is the sense in which the graph is a "triangulation.")

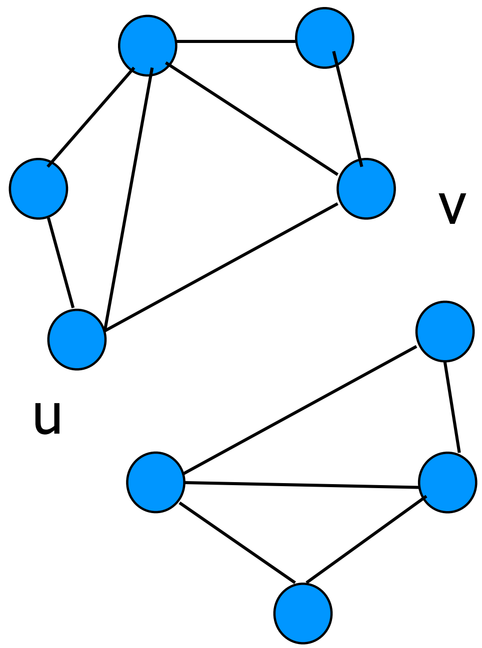
Q: triangulated cycle graph

Q. How to construct a tree decomposition?



Q: triangulated cycle graph

Q. How to construct a tree decomposition?
A.
- If G is a triangle, create a single node.
- Otherwise consider an internal edge (u,v).
- Removing u and v gives two unconnected components A and B.
- Consider two subgraphs of G,
- induced by A∪{u,v} and B∪{u,v}.
- Both are triangulated cycle graphs.
- Recursively compute subtrees, and connect these.



Next step: Research on Algorithms in Delft

Research. Planning and Scheduling algorithms (problems are NP-hard)

"New" aspects

- dynamic environments: problems change continuously, uncertainty
- automatically detect patterns (learning)

Example projects

- scheduling in the smart grid
- energy system investment support
- train shunting and maintenance scheduling at NS

see also *Intelligent Decision Making* courses in Q3 and Q4

Approach. Both theoretical (complexity/run-time analysis, proving game theoretic properties) as well as experimental (comparing on benchmark problems, apply to realistic settings)

TU Delft

Study Advice

Please read (about 12 pages)

1. Gerhard Woeginger, Exact algorithms for NP-hard problems: A survey, *Combinatorial Optimization*, LNCS 3570, pp 187-207, 2003: Section 1-2 (intro)

2. Falk Hueffner, Rold Niedermeier and Sebastian Wernicke, Techniques for Practical Fixed-Parameter Algorithms, *The Computer Journal*, 51(1):7–25, 2008: Section 1 (background), Section 2 (Kernelization) and Section 3 (search trees)

Homework

- Reduce hitting set
- 3 partition

Exam on November 6 in the morning