Parameterized complexity of some stable matching problems

Ildi Schlotter

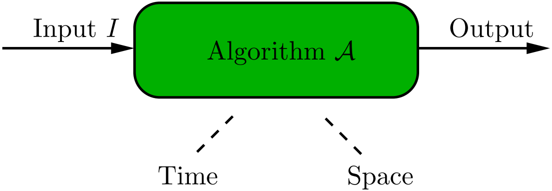
Budapest University of Technology and Economics, Hungary

Summer School on Matching Problems, Markets, and Mechanisms
Budapest, 27 June 2013.

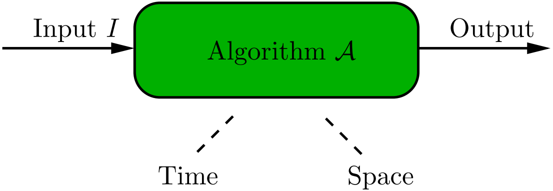
Outline

- Algorithms and complexity: the classical view
- Parameterized complexity:
  - Fixed-Parameter Tractability (FPT)
  - W[1]-hardness
- Parameterizing hard variants of Stable Matching:
  - Stable Matching with Ties and Incomplete Lists
  - Egalitarian, Minimum Regret, and Sex-Equal Stable Matching
  - Hospitals/Residents with Couples: different variants
- Future directions?

Algorithms and complexity



Algorithms and complexity



Running time of $\mathcal{A}$:

- Depends on the input I.
- Measuring the complexity of I: size of I.
- $T(n) =$ maximum number of steps on any input of size $\leq n$.
- Efficient algorithms: $T(n) = n$, $T(n) = n^2$, $T(n) = n^3$, ...
  $\longrightarrow$ $T(n)$ should be a polynomial (having fixed degree).
  $\Longrightarrow$ **P**: Polynomial-time solvable problems.

Hard problems

NP-hard problems:

- No polynomial-time algorithm is known for them.
- There is strong evidence suggesting that NP-hard problems are not in **P**.

Hard problems

NP-hard problems:

- No polynomial-time algorithm is known for them.
- There is strong evidence suggesting that NP-hard problems are not in **P**.

What to do with hard problems in practice?

- “Best effort”: exponential-time algorithms.
- Approximation: finding a sub-optimal solution fast, with a quality guarantee.
  $\longrightarrow$ we want to maximize the value of an objective function f
  $\longrightarrow$ α-approximation: $f(\text{Output}) \geq \dfrac{\text{OPT}}{\alpha}$
- Heuristics: finding a sub-optimal solution fast, no guarantee.
- Parameterized complexity!

Parameterized complexity

Framework for dealing with hard problems [Downey & Fellows, 1999]

- Each input I comes with a parameter k.
- Running time: $T(n,k) \Longrightarrow$ refined complexity measure ($|I| = n$).

Parameterized complexity

Framework for dealing with hard problems [Downey & Fellows, 1999]

- Each input I comes with a parameter k.
- Running time: $T(n, k) \Longrightarrow$ refined complexity measure ($|I| = n$).
- Aim: algorithms with running time

$$f(k)n^{O(1)} \text{ for some function } f.$$

$\longrightarrow$ fixed-parameter tractable (FPT) algorithms

Parameterized complexity

Framework for dealing with hard problems [Downey & Fellows, 1999]

- Each input I comes with a parameter k.
- Running time: $T(n,k) \Longrightarrow$ refined complexity measure ($|I| = n$).
- Aim: algorithms with running time

$$f(k)n^{O(1)} \text{ for some function } f.$$

$\longrightarrow$ fixed-parameter tractable (FPT) algorithms
$\Longrightarrow$ exponential part: restricted to the parameter
dependence on n: fixed-degree polynomial, independent of k
$\longrightarrow$ Example: $2^k n$ is FPT, but n^k is not!

Parameterized complexity

Framework for dealing with hard problems [Downey & Fellows, 1999]

- Each input I comes with a parameter k.
- Running time: $T(n,k) \Longrightarrow$ refined complexity measure ($|I| = n$).
- Aim: algorithms with running time

$$f(k)n^{O(1)} \text{ for some function } f.$$

$\longrightarrow$ fixed-parameter tractable (FPT) algorithms
$\Longrightarrow$ exponential part: restricted to the parameter
dependence on n: fixed-degree polynomial, independent of k
$\longrightarrow$ Example: $2^k n$ is FPT, but n^k is not!

- Motivation: if the parameter k is small in practice, an FPT algorithm can be efficient.

Example: Vertex Cover

Vertex Cover

Input: an undirected graph G and an integer k.
Task: find a vertex cover of size at most k.

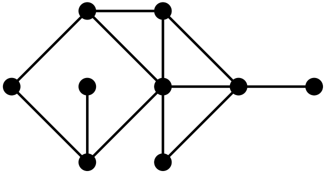
A set $S \subseteq V(G)$ is a *vertex cover* in G, if each edge of G has at least one endpoint in S.

Example: Vertex Cover

Vertex Cover

Input: an undirected graph G and an integer k.
Task: find a vertex cover of size at most k.

A set $S \subseteq V(G)$ is a *vertex cover* in G, if each edge of G has at least one endpoint in S.

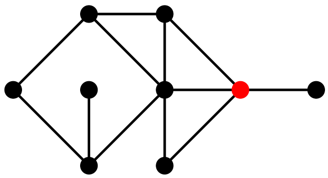


Example: Vertex Cover

Vertex Cover

Input: an undirected graph G and an integer k.
Task: find a vertex cover of size at most k.

A set $S \subseteq V(G)$ is a *vertex cover* in G, if each edge of G has at least one endpoint in S.

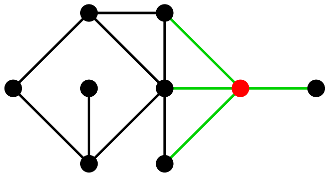


Example: Vertex Cover

Vertex Cover

Input: an undirected graph G and an integer k.
Task: find a vertex cover of size at most k.

A set $S \subseteq V(G)$ is a *vertex cover* in G, if each edge of G has at least one endpoint in S.

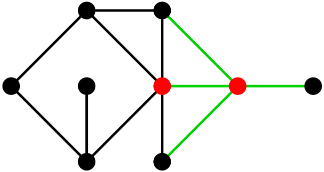


Example: VERTEX COVER

VERTEX COVER

Input: an undirected graph G and an integer k.
Task: find a vertex cover of size at most k.

A set $S \subseteq V(G)$ is a *vertex cover* in G, if each edge of G has at least one endpoint in S.

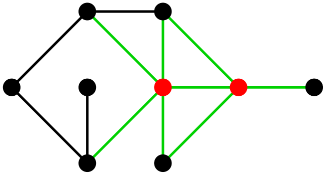


Example: Vertex Cover

Vertex Cover

Input: an undirected graph G and an integer k.
Task: find a vertex cover of size at most k.

A set $S \subseteq V(G)$ is a *vertex cover* in G, if each edge of G has at least one endpoint in S.

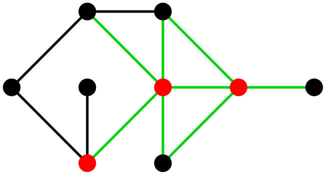


Example: Vertex Cover

Vertex Cover

Input: an undirected graph G and an integer k.
Task: find a vertex cover of size at most k.

A set $S \subseteq V(G)$ is a *vertex cover* in G, if each edge of G has at least one endpoint in S.

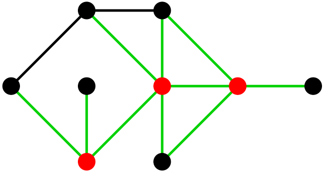


Example: Vertex Cover

Vertex Cover

Input: an undirected graph G and an integer k.
Task: find a vertex cover of size at most k.

A set $S \subseteq V(G)$ is a *vertex cover* in G, if each edge of G has at least one endpoint in S.

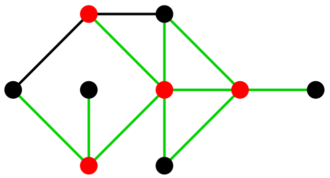


Example: VERTEX COVER

VERTEX COVER

Input: an undirected graph G and an integer k.
Task: find a vertex cover of size at most k.

A set $S \subseteq V(G)$ is a *vertex cover* in G, if each edge of G has at least one endpoint in S.

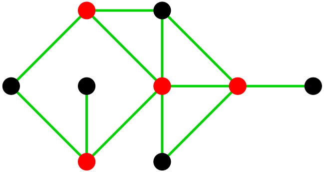


Example: Vertex Cover

Vertex Cover

Input: an undirected graph G and an integer k.
Task: find a vertex cover of size at most k.

A set $S \subseteq V(G)$ is a *vertex cover* in G, if each edge of G has at least one endpoint in S.



Example: Vertex Cover

Vertex Cover

Input: an undirected graph G and an integer k.
Task: find a vertex cover of size at most k.

A set $S \subseteq V(G)$ is a *vertex cover* in G, if each edge of G has at least one endpoint in S.

Classical complexity: NP-hard.
$\Longrightarrow$ no hope for a polynomial-time algorithm.

What can we do in exponential time?

Example: Vertex Cover

Vertex Cover

Input: an undirected graph G and an integer k.
Task: find a vertex cover of size at most k.

A set $S \subseteq V(G)$ is a *vertex cover* in G, if each edge of G has at least one endpoint in S.

Classical complexity: NP-hard.
$\Longrightarrow$ no hope for a polynomial-time algorithm.

What can we do in exponential time?

- Brute force: try each possible set $S \subseteq V(G)$ of size k, check if it covers each edge.
  $\Longrightarrow$ requires $\binom{n}{k} O(|E(G)|) = O(n^{k+1})$ time, where $n = |V(G)|$.
  Slooooow... Does not work for $n = 100$ and $k = 10$.

Example: Vertex Cover

Vertex Cover

Input: an undirected graph G and an integer k.
Task: find a vertex cover of size at most k.

A set $S \subseteq V(G)$ is a *vertex cover* in G, if each edge of G has at least one endpoint in S.

Classical complexity: NP-hard.
$\Longrightarrow$ no hope for a polynomial-time algorithm.

What can we do in exponential time?

- Brute force: try each possible set $S \subseteq V(G)$ of size k, check if it covers each edge.
  $\Longrightarrow$ requires $\binom{n}{k} O(|E(G)|) = O(n^{k+1})$ time, where $n = |V(G)|$.
  Slooooow... Does not work for $n = 100$ and $k = 10$.
- Can we do better?
  Can we get the k out of the exponent of n?

Parameterizing Vertex Cover

k-Vertex Cover

Input: an undirected graph G.
Parameter: an integer k.
Task: find a vertex cover of size at most k.

Question: is this problem FPT?

Parameterizing Vertex Cover

k-Vertex Cover

Input: an undirected graph G.
Parameter: an integer k.
Task: find a vertex cover of size at most k.

Question: is this problem FPT?

Observation.

If xy is an edge, then $x \in S$ or $y \in S$ for any vertex cover S.

Parameterizing Vertex Cover

k-Vertex Cover

Input: an undirected graph G.
Parameter: an integer k.
Task: find a vertex cover of size at most k.

Question: is this problem FPT?

Observation.

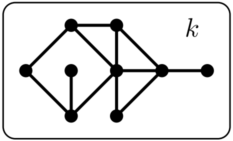
If xy is an edge, then $x \in S$ or $y \in S$ for any vertex cover S.

$\Longrightarrow$ algorithm:

1. Let $S = \emptyset$.
2. While there is an uncovered edge xy:
   if $k = 0$, then reject;
   if $k > 0$, then branch into two directions:
   $\rightarrow$ branch 1: $S := S \cup \{x\}$, $k = k - 1$.
   $\rightarrow$ branch 2: $S := S \cup \{y\}$, $k = k - 1$.
3. Output S.

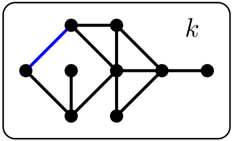
k-Vertex Cover – an $O(2^k kn)$ algorithm

Bounded search tree algorithm:



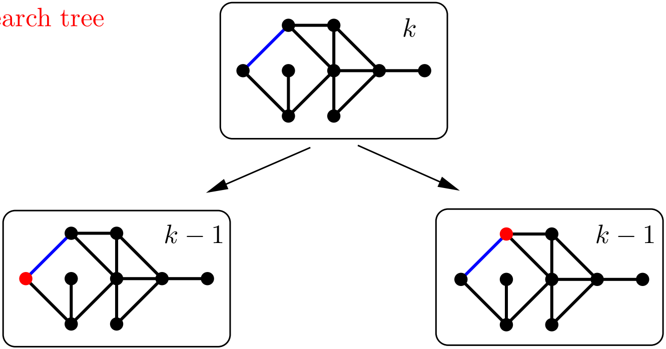
k-Vertex Cover – an $O(2^k kn)$ algorithm

Bounded search tree algorithm:



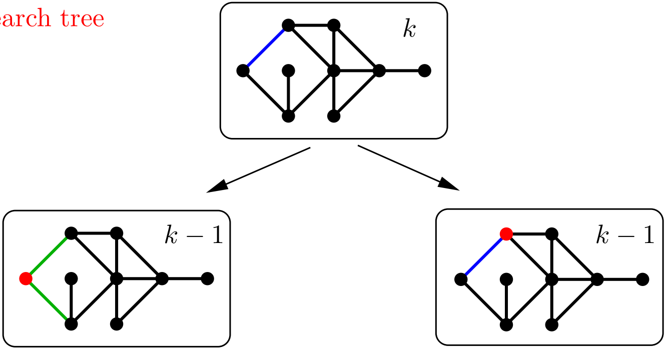
k-Vertex Cover – an $O(2^k kn)$ algorithm

Bounded search tree algorithm:



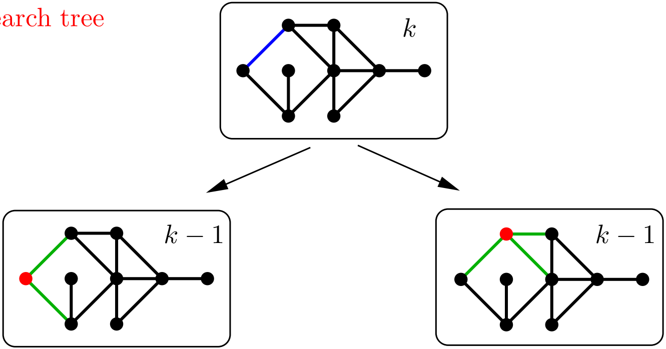
k-Vertex Cover – an $O(2^k kn)$ algorithm

Bounded search tree algorithm:



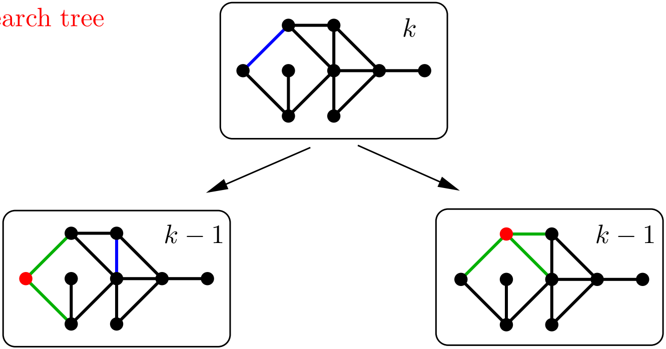
k-Vertex Cover – an $O(2^k kn)$ algorithm

Bounded search tree algorithm:



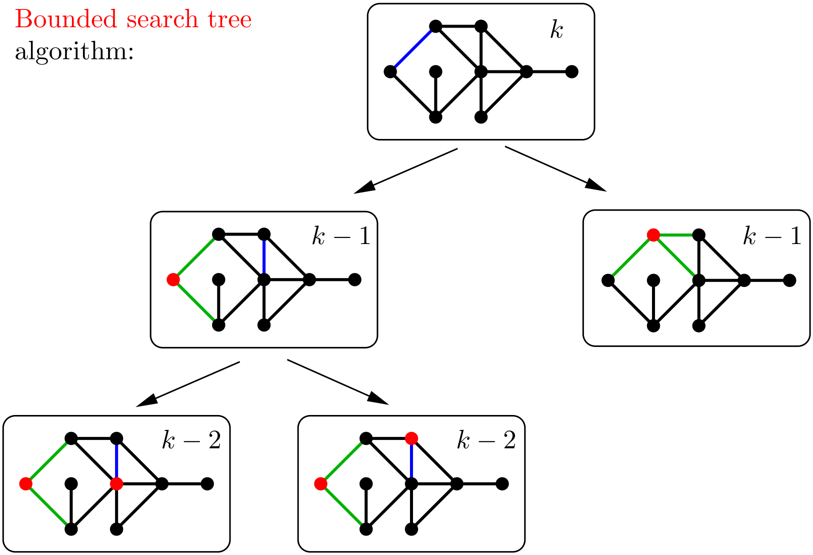
k-Vertex Cover – an $O(2^k kn)$ algorithm

Bounded search tree algorithm:



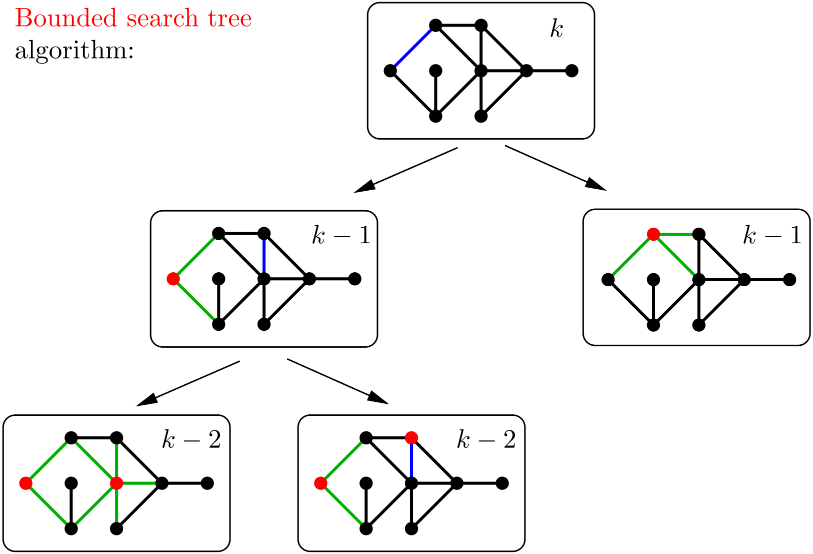
k-Vertex Cover – an $O(2^k kn)$ algorithm

Bounded search tree algorithm:



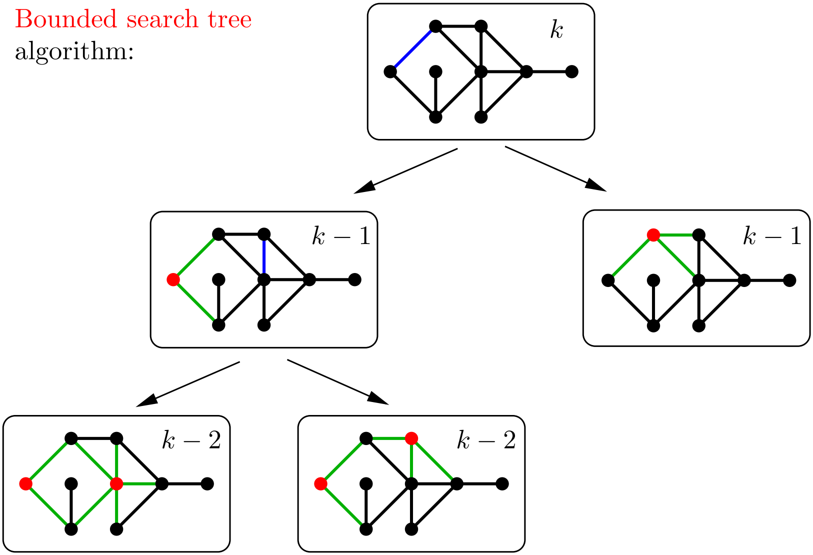
k-Vertex Cover – an $O(2^k kn)$ algorithm

Bounded search tree algorithm:



k-Vertex Cover – an $O(2^k kn)$ algorithm

Bounded search tree algorithm:



k-Vertex Cover – an $O(2^k kn)$ algorithm

Bounded search tree algorithm:

- Each branching decreases the parameter.
  $\Longrightarrow$ Search tree has depth at most k
  $\Longrightarrow$ at most 2^k leaves.
- Computations at one node: $O(|E(G)|) = O(kn)$ time.
- Overall running time: $O(2^k kn)$
  $\Longrightarrow$ fixed-parameter tractable with parameter k;
  $\longrightarrow$ much better than $O(n^{k+1})$!

k-Vertex Cover – an $O(2^k kn)$ algorithm

Bounded search tree algorithm:

- Each branching decreases the parameter.
  $\Longrightarrow$ Search tree has depth at most k
  $\Longrightarrow$ at most 2^k leaves.
- Computations at one node: $O(|E(G)|) = O(kn)$ time.
- Overall running time: $O(2^k kn)$
  $\Longrightarrow$ fixed-parameter tractable with parameter k;
  $\longrightarrow$ much better than $O(n^{k+1})$!

Currently the fastest algorithm: $O(1.2738^k + kn)$ [Chen et al.]

$\Longrightarrow$ Vertex Cover is solvable for $n = 10^6$ and $k = 40$.

Clique

Clique

Input: an undirected graph G and an integer k.
Question: is there a clique of size at least k in G?

A *clique* is a set of mutually adjacent vertices.

Clique

Clique

Input: an undirected graph G and an integer k.
Question: is there a clique of size at least k in G?

A *clique* is a set of mutually adjacent vertices.
Complexity:

- NP-hard $\Longrightarrow$ no hope for a poly-time algorithm.

Clique

Clique

Input: an undirected graph G and an integer k.
Question: is there a clique of size at least k in G?

A *clique* is a set of mutually adjacent vertices.
Complexity:

- NP-hard $\Longrightarrow$ no hope for a poly-time algorithm.
- Brute force (exponential-time) algorithm: $O(n^k)$
  Try each possible subset S of k vertices, and check it.

Clique

Clique

Input: an undirected graph G and an integer k.
Question: is there a clique of size at least k in G?

A *clique* is a set of mutually adjacent vertices.
Complexity:

- NP-hard $\Longrightarrow$ no hope for a poly-time algorithm.
- Brute force (exponential-time) algorithm: $O(n^k)$
  Try each possible subset S of k vertices, and check it.
- Can we get k out of the exponent?
  Is Clique FPT if the parameter is k?
  $\Rightarrow$ no FPT-algorithm is known... But can we prove it?

Parameterized hardness

W[1]-hardness:

- Analogous to NP-hardness.
- Hardness hierarchy: $\mathrm{FPT} \subseteq \underbrace{W[1] \subseteq W[2] \subseteq \cdots \subseteq WP}_{\text{intractable classes}}$
- W[1]-hard problems are unlikely to admit an FPT-algorithm.
  An FPT-algorithm for a W[1]-hard problem would yield an FPT-algorithm for *all* problems in W[1].
- Defining W[1]-hardness:
  we need parameterized or FPT-reductions.

Parameterized hardness

W[1]-hardness:

- Analogous to NP-hardness.
- Hardness hierarchy: $\mathrm{FPT} \subseteq \underbrace{W[1] \subseteq W[2] \subseteq \cdots \subseteq WP}_{\text{intractable classes}}$
- W[1]-hard problems are unlikely to admit an FPT-algorithm. An FPT-algorithm for a W[1]-hard problem would yield an FPT-algorithm for *all* problems in W[1].
- Defining W[1]-hardness: we need parameterized or FPT-reductions.

Theorem [Downey and Fellows].

CLIQUE is W[1]-hard with parameter k (size of the aimed solution).

Parameterized hardness

W[1]-hardness:

- Analogous to NP-hardness.
- Hardness hierarchy: $\mathrm{FPT} \subseteq \underbrace{W[1] \subseteq W[2] \subseteq \cdots \subseteq WP}_{\text{intractable classes}}$
- W[1]-hard problems are unlikely to admit an FPT-algorithm.
  An FPT-algorithm for a W[1]-hard problem would yield an FPT-algorithm for *all* problems in W[1].
- Defining W[1]-hardness:
  we need parameterized or FPT-reductions.

Theorem [Downey and Fellows].

CLIQUE is W[1]-hard with parameter k (size of the aimed solution).

$\Longrightarrow$ No FPT-algorithm, unless FPT $\neq$ W[1].

Parameterized hardness

W[1]-hardness:

- Analogous to NP-hardness.
- Hardness hierarchy: $\mathrm{FPT} \subseteq \underbrace{W[1] \subseteq W[2] \subseteq \cdots \subseteq WP}_{\text{intractable classes}}$
- W[1]-hard problems are unlikely to admit an FPT-algorithm. An FPT-algorithm for a W[1]-hard problem would yield an FPT-algorithm for *all* problems in W[1].
- Defining W[1]-hardness: we need parameterized or FPT-reductions.

Theorem [Downey and Fellows].

Clique is W[1]-hard with parameter k (size of the aimed solution).

$\Longrightarrow$ No FPT-algorithm, unless FPT $\neq$ W[1].
No $n^{o(k)}$ time algorithm, unless ETH fails.

Parameterized hardness

W[1]-hardness:

- Analogous to NP-hardness.
- Hardness hierarchy: $\mathrm{FPT} \subseteq \underbrace{W[1] \subseteq W[2] \subseteq \cdots \subseteq WP}_{\text{intractable classes}}$
- W[1]-hard problems are unlikely to admit an FPT-algorithm.
  An FPT-algorithm for a W[1]-hard problem would yield an FPT-algorithm for *all* problems in W[1].
- Defining W[1]-hardness:
  we need parameterized or FPT-reductions.

Theorem [Downey and Fellows].

Clique is W[1]-hard with parameter k (size of the aimed solution).

$\Longrightarrow$ No FPT-algorithm, unless FPT $\neq$ W[1].
No $n^{o(k)}$ time algorithm, unless ETH fails.

$\Longrightarrow$ $O(n^k)$ seems optimal.

Parameterized reductions

Let Q and Q' be two parameterized problems.
We assume that Q and Q' are decision problems (answer: yes / no).

Parameterized reductions

Let Q and Q' be two parameterized problems.
We assume that Q and Q' are decision problems (answer: yes / no).

Definition: FPT-reduction.

An FPT- (or parameterized) reduction from Q to Q' is a function that, given an input (I, k) for Q, computes an input (I', k') for Q' in FPT time such that

- (I, k) is a yes-instance for $Q \iff (I', k')$ is a yes-instance for Q', and
- $k' \leq g(k)$ for some function g.

Parameterized reductions

Let Q and Q' be two parameterized problems.
We assume that Q and Q' are decision problems (answer: yes / no).

Definition: FPT-reduction.

An FPT- (or parameterized) reduction from Q to Q' is a function that, given an input (I, k) for Q, computes an input (I', k') for Q' in FPT time such that

- (I, k) is a yes-instance for $Q \iff (I', k')$ is a yes-instance for Q', and
- $k' \leq g(k)$ for some function g.

FPT is closed under parameterized reductions:
If Q can be reduced to Q', and Q' is FPT, then Q is FPT as well.

Parameterized reductions: example I.

k-Independent Set

Input: an undirected graph G.
Parameter: an integer k.
Question: is there an independent set of size k in G?

An *independent set* is a set of pairwise non-adjacent vertices.

Parameterized reductions: example I.

k-Independent Set

Input: an undirected graph G.
Parameter: an integer k.
Question: is there an independent set of size k in G?

An *independent set* is a set of pairwise non-adjacent vertices.

A reduction from k-Clique to k-Independent Set:
$(G, k) \longrightarrow (G', k')$ where

- G' is the complement of G: $\forall e : e \in E(G') \iff e \notin E(G)$;
- $k' = k$.

Parameterized reductions: example I.

k-Independent Set

Input: an undirected graph G.
Parameter: an integer k.
Question: is there an independent set of size k in G?

An *independent set* is a set of pairwise non-adjacent vertices.

A reduction from k-Clique to k-Independent Set:
$(G, k) \longrightarrow (G', k')$ where

- G' is the complement of G: $\forall e : e \in E(G') \iff e \notin E(G)$;
- $k' = k$.

Is this an FPT-reduction?

Parameterized reductions: example I.

k-Independent Set

Input: an undirected graph G.
Parameter: an integer k.
Question: is there an independent set of size k in G?

An *independent set* is a set of pairwise non-adjacent vertices.

A reduction from k-Clique to k-Independent Set:
$(G, k) \longrightarrow (G', k')$ where

- G' is the complement of G: $\forall e : e \in E(G') \iff e \notin E(G)$;
- $k' = k$.

Is this an FPT-reduction?

- G has a clique of size $k \iff G'$ has k independent vertices.

Parameterized reductions: example I.

k-Independent Set

Input: an undirected graph G.
Parameter: an integer k.
Question: is there an independent set of size k in G?

An *independent set* is a set of pairwise non-adjacent vertices.

A reduction from k-Clique to k-Independent Set:
$(G, k) \longrightarrow (G', k')$ where

- G' is the complement of G: $\forall e : e \in E(G') \iff e \notin E(G)$;
- $k' = k$.

Is this an FPT-reduction?

- G has a clique of size $k \iff G'$ has k independent vertices. ✓

Parameterized reductions: example I.

k-Independent Set

Input: an undirected graph G.
Parameter: an integer k.
Question: is there an independent set of size k in G?

An *independent set* is a set of pairwise non-adjacent vertices.

A reduction from k-Clique to k-Independent Set:
$(G, k) \longrightarrow (G', k')$ where

- G' is the complement of G: $\forall e : e \in E(G') \iff e \notin E(G)$;
- $k' = k$.

Is this an FPT-reduction?

- G has a clique of size $k \iff G'$ has k independent vertices. ✓
- The reduction runs in FPT time.

Parameterized reductions: example I.

k-Independent Set

Input: an undirected graph G.
Parameter: an integer k.
Question: is there an independent set of size k in G?

An *independent set* is a set of pairwise non-adjacent vertices.

A reduction from k-Clique to k-Independent Set:
$(G, k) \longrightarrow (G', k')$ where

- G' is the complement of G: $\forall e : e \in E(G') \iff e \notin E(G)$;
- $k' = k$.

Is this an FPT-reduction?

- G has a clique of size $k \iff G'$ has k independent vertices. ✓
- The reduction runs in FPT time. ✓

Parameterized reductions: example I.

k-Independent Set

Input: an undirected graph G.
Parameter: an integer k.
Question: is there an independent set of size k in G?

An *independent set* is a set of pairwise non-adjacent vertices.

A reduction from k-Clique to k-Independent Set:
$(G, k) \longrightarrow (G', k')$ where

- G' is the complement of G: $\forall e : e \in E(G') \iff e \notin E(G)$;
- $k' = k$.

Is this an FPT-reduction?

- G has a clique of size $k \iff G'$ has k independent vertices. $\checkmark$
- The reduction runs in FPT time. $\checkmark$
- k' is a function of k only.

Parameterized reductions: example I.

k-INDEPENDENT SET

Input: an undirected graph G.
Parameter: an integer k.
Question: is there an independent set of size k in G?

An *independent set* is a set of pairwise non-adjacent vertices.

A reduction from k-CLIQUE to k-INDEPENDENT SET:
$(G, k) \longrightarrow (G', k')$ where

- G' is the complement of G: $\forall e : e \in E(G') \iff e \notin E(G)$;
- $k' = k$.

Is this an FPT-reduction?

- G has a clique of size $k \iff G'$ has k independent vertices. ✓
- The reduction runs in FPT time. ✓
- k' is a function of k only. ✓

Parameterized reductions: example II.

A reduction from k-Independent Set to k-Vertex Cover:

$(G, k) \longrightarrow (G', k')$ where

- $G' = G$ and
- $k' = n - k$ $(n = |V(G)|)$.

Parameterized reductions: example II.

A reduction from k-INDEPENDENT SET to k-VERTEX COVER:

$(G, k) \longrightarrow (G', k')$ where

- $G' = G$ and
- $k' = n - k$ $(n = |V(G)|)$.

Is this an FPT-reduction?

Parameterized reductions: example II.

A reduction from k-Independent Set to k-Vertex Cover:

$(G, k) \longrightarrow (G', k')$ where

- $G' = G$ and
- $k' = n - k$ $(n = |V(G)|)$.

Is this an FPT-reduction?

- G has an indep. set of size $k \iff G$ has a vertex cover of size $n - k$?

Parameterized reductions: example II.

A reduction from k-INDEPENDENT SET to k-VERTEX COVER:

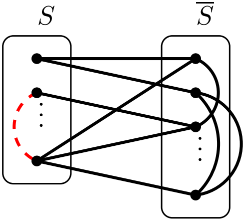
$(G, k) \longrightarrow (G', k')$ where

- $G' = G$ and
- $k' = n - k$ $(n = |V(G)|)$.

Is this an FPT-reduction?

- G has an indep. set of size $k \iff G$ has a vertex cover of size $n - k$?

  S is an independent set in $G \iff \bar{S} = V(G) \setminus S$ is a vertex cover.



Parameterized reductions: example II.

A reduction from k-Independent Set to k-Vertex Cover:

$(G, k) \longrightarrow (G', k')$ where

- $G' = G$ and
- $k' = n - k$ $(n = |V(G)|)$.

Is this an FPT-reduction?

- G has an indep. set of size $k \iff G$ has a vertex cover of size $n - k$?

  S is an independent set in $G \iff \bar{S} = V(G) \setminus S$ is a vertex cover.

  ✓

Parameterized reductions: example II.

A reduction from k-Independent Set to k-Vertex Cover:

$(G, k) \longrightarrow (G', k')$ where

- $G' = G$ and
- $k' = n - k$ $(n = |V(G)|)$.

Is this an FPT-reduction?

- G has an indep. set of size k $\iff$ G has a vertex cover of size $n - k$?
  S is an independent set in G $\iff$ $\bar{S} = V(G) \setminus S$ is a vertex cover.
  $\checkmark$
- The reduction runs in FPT time.

Parameterized reductions: example II.

A reduction from k-Independent Set to k-Vertex Cover:

$(G, k) \longrightarrow (G', k')$ where

- $G' = G$ and
- $k' = n - k$ $(n = |V(G)|)$.

Is this an FPT-reduction?

- G has an indep. set of size k $\iff$ G has a vertex cover of size $n - k$?
  S is an independent set in G $\iff$ $\bar{S} = V(G) \setminus S$ is a vertex cover.
  ✓
- The reduction runs in FPT time. ✓

Parameterized reductions: example II.

A reduction from k-Independent Set to k-Vertex Cover:

$(G, k) \longrightarrow (G', k')$ where

- $G' = G$ and
- $k' = n - k$ $(n = |V(G)|)$.

Is this an FPT-reduction?

- G has an indep. set of size $k \iff G$ has a vertex cover of size $n - k$?
  S is an independent set in $G \iff \bar{S} = V(G) \setminus S$ is a vertex cover.
  $\checkmark$
- The reduction runs in FPT time. $\checkmark$
- k' is a function of k only.

Parameterized reductions: example II.

A reduction from k-INDEPENDENT SET to k-VERTEX COVER:

$(G,k) \longrightarrow (G',k')$ where

- $G' = G$ and
- $k' = n - k$ $(n = |V(G)|)$.

Is this an FPT-reduction?

- G has an indep. set of size k $\iff$ G has a vertex cover of size $n-k$?
  S is an independent set in G $\iff$ $\bar{S} = V(G) \setminus S$ is a vertex cover.
  ✓
- The reduction runs in FPT time. ✓
- k' is a function of k only. ↯

Proving W[1]-hardness

How to prove W[1]-hardness?

Theorem.

If Q is W[1]-hard, and Q can be FPT-reduced to some problem Q', then Q' is W[1]-hard as well.

$\Longrightarrow$ We can prove W[1]-hardness of Q' by giving an FPT-reduction from any known W[1]-hard problem.

Proving W[1]-hardness

How to prove W[1]-hardness?

Theorem.

If Q is W[1]-hard, and Q can be FPT-reduced to some problem Q', then Q' is W[1]-hard as well.

$\Longrightarrow$ We can prove W[1]-hardness of Q' by giving an FPT-reduction from any known W[1]-hard problem.

Example:

- We know that k-Clique is W[1]-hard.
- We just gave an FPT-reduction from k-Clique to k-Independent Set.

$\Longrightarrow$ We proved that k-Independent Set is W[1]-hard.

Many parameters

Extension of the model: multiple parameters.

- Each input I has multiple parameters $k_1, k_2, \ldots, k_d \in \mathbb{N}$.
- Easy to extend the notation.
  An algorithm is FPT with combined parameters $(k_1, \ldots, k_d)$, if it runs in time

$$f(k_1, \ldots, k_d)|I|^{O(1)} \text{ for some function } f.$$

Many parameters

Extension of the model: multiple parameters.

- Each input I has multiple parameters $k_1, k_2, \ldots, k_d \in \mathbb{N}$.
- Easy to extend the notation.
  An algorithm is FPT with combined parameters $(k_1, \ldots, k_d)$, if it runs in time

$$f(k_1, \ldots, k_d)|I|^{O(1)} \text{ for some function } f.$$

- Very useful in practice!
  There can be many important parameters in a problem.
- Multi-dimensional view on the complexity of the problem
  $\Longrightarrow$ yields a more detailed insight.

How to choose the parameter?

A good parameter ...

- has small value in practice,
- makes the problem FPT.

$\Longrightarrow$ If the parameter is small, the problem must be easy!

How to choose the parameter?

A good parameter ...

- has small value in practice,
- makes the problem FPT.

  $\Longrightarrow$ If the parameter is small, the problem must be easy!

Typical parameters:

- size of the solution
- some natural, simple, problem-specific property of the input
- distance from triviality
  $\longrightarrow$ Which special cases are easy?
- search radius in local search problems

How to choose the parameter?

A good parameter ...

- has small value in practice,
- makes the problem FPT.

  $\Longrightarrow$ If the parameter is small, the problem must be easy!

Typical parameters:

- size of the solution
- some natural, simple, problem-specific property of the input
- distance from triviality
  $\longrightarrow$ Which special cases are easy?
- search radius in local search problems

Multivariate analysis: the more parameters we examine, the more knowledge we obtain about the problem.

Hard variants of some stable matching problems

NP-hard problems for which the parameterized complexity has been studied:

- Maximum Stable Matching with Ties and Incomplete Lists
- Minimum Regret Stable Matching
  Egalitarian Stable Matching
  Sex-Equal Stable Matching
- Hospitals/Residents with Couples (HRC)
  Matching with Couples
  Special HRC with master list
- Socially stable matchings for Hospitals/Residents
- Housing Markets with Duplicate Houses

SMTI

Stable Matching with Ties and Incomplete Lists:

- Input: A set W of women and a set U of men, and a set $\mathcal{L} = \{L_p \mid p \in W \cup U\}$ of preference lists.
- L_p may be incomplete: contains only acceptable partners.
  L_p may contain ties: possible partners equally good for p.

SMTI

Stable Matching with Ties and Incomplete Lists:

- Input: A set W of women and a set U of men, and a set $\mathcal{L} = \{L_p \mid p \in W \cup U\}$ of preference lists.
- L_p may be incomplete: contains only acceptable partners. L_p may contain ties: possible partners equally good for p.
- A matching M is a set of acceptable man-women pairs, containing each person at most once.
  $\longrightarrow$ It is a matching in the underlying bipartite graph.
  $\longrightarrow$ $M(p)$ is the person matched to p by M.
- $(w, m) \in W \times U$ is a blocking pair w.r.t. a matching M, if
  - w is unmatched, or strictly prefers m to $M(w)$; and
  - m is unmatched, or strictly prefers w to $M(m)$.
- M is stable $\iff$ there is no blocking pair for M.

MaxSMTI

The Gale-Shapley algorithm can find *some* stable matching in linear time. But what about its size?

MaxSMTI

Input: An SMTI instance $I = (W, U, \mathcal{L})$ as described above.
Task: Find a stable matching for I of maximum size.

MaxSMTI

The Gale-Shapley algorithm can find *some* stable matching in linear time. But what about its size?

MaxSMTI

Input: An SMTI instance $I = (W, U, \mathcal{L})$ as described above.
Task: Find a stable matching for I of maximum size.

Theorem [D. Manlove et al.]

MaxSMTI is NP-hard, even if:

- each tie has length 2;
- each tie is at the end of the preference list of a woman.

MaxSMTI

The Gale-Shapley algorithm can find *some* stable matching in linear time. But what about its size?

MaxSMTI

Input: An SMTI instance $I = (W, U, \mathcal{L})$ as described above.
Task: Find a stable matching for I of maximum size.

Theorem [D. Manlove et al.]

MaxSMTI is NP-hard, even if:

- each tie has length 2;
- each tie is at the end of the preference list of a woman.

Easy special cases: no ties OR preference lists are complete:
$\Longrightarrow$ all stable matchings have the same size,
$\Longrightarrow$ the Gale-Shapley algorithm is optimal.

Parameterized complexity of MaxSMTI

Possible parameters:

- the maximum length of ties;
  Recall: MaxSMTI is NP-hard, even if each tie has length 2!
  $\Longrightarrow$ not a good parameter.
- the number of ties;
- the total length L of ties.

Parameterized complexity of MaxSMTI

Possible parameters:

- the maximum length of ties;
  Recall: MaxSMTI is NP-hard, even if each tie has length 2!
  $\Longrightarrow$ not a good parameter.
- the number of ties;
- the total length L of ties.

Theorem [D. Marx and I. S.]

MaxSMTI is FPT if the parameter is the total length of ties.

Parameterized complexity of MaxSMTI

Possible parameters:

- the maximum length of ties;
  Recall: MaxSMTI is NP-hard, even if each tie has length 2!
  $\Longrightarrow$ not a good parameter.
- the number of ties;
- the total length L of ties.

Theorem [D. Marx and I. S.]

MaxSMTI is FPT if the parameter is the total length of ties.

Theorem [D. Marx and I. S.]

MaxSMTI is W[1]-hard if the parameter is the number of ties, even if ties are only on the women's side.

FPT algorithm for MAXSMTI (par.: $\sum$ length of ties)

FPT algorithm for MAXSMTI:

1. Break ties in all possible ways for the given SMTI instance I. Let $I_1, I_2, \ldots, I_t$ be the obtained instances (without ties).
2. For each I_j, $j = 1, \ldots, t$, compute a stable matching M_i.
3. Output the largest among $M_1, M_2, \ldots, M_t$.

FPT algorithm for MaxSMTI (par.: $\sum$ length of ties)

FPT algorithm for MaxSMTI:

1. Break ties in all possible ways for the given SMTI instance I. Let $I_1, I_2, \ldots, I_t$ be the obtained instances (without ties).
2. For each I_j, $j = 1, \ldots, t$, compute a stable matching M_i.
3. Output the largest among $M_1, M_2, \ldots, M_t$.

Observation.

Any stable matching M for I is also stable for *some* instance obtained by breaking ties.

FPT algorithm for MaxSMTI (par.: $\sum$ length of ties)

FPT algorithm for MaxSMTI:

1. Break ties in all possible ways for the given SMTI instance I.
   Let $I_1, I_2, \ldots, I_t$ be the obtained instances (without ties).
2. For each I_j, $j = 1, \ldots, t$, compute a stable matching M_i.
3. Output the largest among $M_1, M_2, \ldots, M_t$.

Observation.

Any stable matching M for I is also stable for *some* instance obtained by breaking ties.

- Suppose T is a tie in w's preference list.
- If w is matched to some $m \in T$, then we break T such that m becomes the most preferred men in T.
  Otherwise, we can break T arbitrarily.
  $\Longrightarrow$ No blocking pair can appear, M remains stable.

FPT algorithm for MaxSMTI (par.: $\sum$ length of ties)

FPT algorithm for MaxSMTI:

1. Break ties in all possible ways for the given SMTI instance I. Let $I_1, I_2, \dots, I_t$ be the obtained instances (without ties).
2. For each I_j, $j = 1, \dots, t$, compute a stable matching M_i.
3. Output the largest among $M_1, M_2, \dots, M_t$.

Observation.

Any stable matching M for I is also stable for *some* instance obtained by breaking ties.

We can break ties in at most $L!$ possible ways. (L: $\sum$ length of ties)
$\Longrightarrow$ running time: $O(L!|I|)$
$\Longrightarrow$ FPT with parameter L.

Fair stable matchings

Maximality vs. fairness?

- We want a stable matching that is *fair* (not necessarily maximal).

Fair stable matchings

Maximality vs. fairness?

- We want a stable matching that is *fair* (not necessarily maximal).
- Many different notions are in use.
- The *cost* of a person p in a matching M:

$$c_M(p) = \begin{cases} \text{the rank of } M(p) \text{ in } L_p, & \text{if } p \text{ is matched in } M; \\ |L_p| + 1, & \text{if } p \text{ is unmatched in } M. \end{cases}$$

Fair stable matchings

Maximality vs. fairness?

- We want a stable matching that is *fair* (not necessarily maximal).
- Many different notions are in use.
- The *cost* of a person p in a matching M:

$$c_M(p) = \begin{cases} \text{the rank of } M(p) \text{ in } L_p, & \text{if } p \text{ is matched in } M; \\ |L_p| + 1, & \text{if } p \text{ is unmatched in } M. \end{cases}$$

- M is egalitarian, if it minimizes $\sum c_M(p)$.
  (≈ optimizing total happiness)
- M is minimum regret, if it minimizes $\max c_M(p)$.
  (≈ optimizing on the least satisfied person)
- M is sex-equal, if it minimizes

$$\delta(M) = \left| \sum_{m \in U} c_M(m) - \sum_{w \in W} c_M(w) \right|.$$

$\longrightarrow \delta(M) \approx$ difference between men's and women's happiness.

Egalitarian and minimum regret stable matchings

Egal SMTI

Input: an SMTI instance I.
Task: find an egalitarian stable matching for I.

MinReg SMTI

Input: an SMTI instance I.
Task: find a minimum regret stable matching for I.

Complexity:

- If no ties are involved $\Longrightarrow$ both can be solved in polynomial time. [Irving et al.], [Gusfield]
- If ties can occur, then both problems become NP-hard, and even hard to approximate. [Halldórsson et al.]

Parameterizing Egal SMTI and MinReg STMI

Possible parameters:

- the maximum length of ties;
- the number t of ties;
- the total length L of ties.

Parameterizing Egal SMTI and MinReg STMI

Possible parameters:

- the maximum length of ties;
- the number t of ties;
- the total length L of ties.

Theorem [Manlove et al.]

If P $\neq$ NP and $\varepsilon > 0$, then no polynomial time algorithm can approximate the Egal SMTI or the MinReg SMTI problem within a factor of $N^{1-\varepsilon}$, where N is the number of men, even if ties are only present on women's side, and each tie has length 2.

$\Longrightarrow$ maximum length of ties: not a good choice.

Parameterizing EGAL SMTI and MINREG STMI

Theorem [D. Marx and I. S.]

Both the EGAL SMTI and the MINREG SMTI problems can be solved by an FPT algorithm, with the parameter being the total length of ties.

Parameterizing EGAL SMTI and MINREG STMI

Theorem [D. Marx and I. S.]

Both the EGAL SMTI and the MINREG SMTI problems can be solved by an FPT algorithm, with the parameter being the total length of ties.

Simple FPT-algorithm:

1. Break ties in all possible ways.
2. For each obtained instance, apply the standard poly-time algorithm for finding an egalitarian or a minimum regret matching.

Important: break ties in a cost-preserving way!
$\longrightarrow$ use explicit *ranking functions* (instead of precedence lists).

FPT-inapproximability

FPT-approximation algorithm:

- Approximates a given optimization problem within some guaranteed factor, and
- runs in FPT time (instead of polynomial time).

FPT-inapproximability

FPT-approximation algorithm:

- Approximates a given optimization problem within some guaranteed factor, and
- runs in FPT time (instead of polynomial time).

Theorem [D. Marx and I. S.]

If $\varepsilon > 0$ and W[1] $\neq$ FPT, then there is no FPT algorithm with the parameter being the number of ties, that can approximate MinReg SMTI or Egal SMTI within a factor of $N^{1-\varepsilon}$, even if ties are only present on women's side.

Sex-equal stable matchings

Sex-equal SMI

Input: an SMI instance I and an integer δ.
Task: find a stable matching M for I with sex-equality measure $\leq \delta$.

Sex-equal stable matchings

SEX-EQUAL SMI

Input: an SMI instance I and an integer δ.
Task: find a stable matching M for I with sex-equality measure $\leq \delta$.

- We assume that preference lists are strictly ordered: an SMI instance is an SMTI instance without ties.
- Recall: the sex-equality measure of a matching M is

$$\delta(M) = \left| \sum_{m \in U} c_M(m) - \sum_{w \in W} c_M(w) \right| .$$

Sex-equal stable matchings

Sex-equal SMI

Input: an SMI instance I and an integer δ.
Task: find a stable matching M for I with sex-equality measure $\leq \delta$.

- We assume that preference lists are strictly ordered: an SMI instance is an SMTI instance without ties.
- Recall: the sex-equality measure of a matching M is

$$\delta(M) = \left| \sum_{m \in U} c_M(m) - \sum_{w \in W} c_M(w) \right| .$$

Complexity:

- NP-hard, even if the preference lists are complete. [Kato]
- If ties can occur, then NP-hard to approximate within a factor of εN for some $\varepsilon > 0$. [Halldórsson et al.]

Parameterizing Sex-equal stable matchings

Parameters examined by McDermid and Irving:

- the sex-equality measure δ we aim for;
- a bound on the length of each preference list.

Parameterizing Sex-equal stable matchings

Parameters examined by McDermid and Irving:

- the sex-equality measure δ we aim for;
- a bound on the length of each preference list.

Theorem [McDermid and Irving]

Sex-equal SMI is NP-hard, even if $\delta = 0$ and all preference lists are of length at most 3.

Parameterizing Sex-equal stable matchings

Parameters examined by McDermid and Irving:

- the sex-equality measure δ we aim for;
- a bound on the length of each preference list.

Theorem [McDermid and Irving]

SEX-EQUAL SMI is NP-hard, even if $\delta = 0$ and all preference lists are of length at most 3.

Theorem [McDermid and Irving]

SEX-EQUAL SMI is polynomial-time solvable if the preference lists of women (or men) are of length at most 2.

Parameterizing Sex-equal stable matchings

Parameters examined by McDermid and Irving:

- the sex-equality measure δ we aim for;
- a bound on the length of each preference list.

Theorem [McDermid and Irving]

Sex-equal SMI is NP-hard, even if $\delta = 0$ and all preference lists are of length at most 3.

Theorem [McDermid and Irving]

Sex-equal SMI is polynomial-time solvable if the preference lists of women (or men) are of length at most 2.

$\longrightarrow$ Not parameterized complexity in the strict sense.
$\longrightarrow$ Seek for parameters which, when small, make the problem easy.

The Hospitals/Residents problem

Hospitals/Residents: many-to-one version of Stable Matching.

Problem instance for Hospitals/Residents.

- agents: a set R of residents and a set H of hospitals
- a capacity $f(h)$ for each $h \in H$, giving the number of open jobs
- a strict (but maybe incomplete) preference list for each agent

The Hospitals/Residents problem

Hospitals/Residents: many-to-one version of Stable Matching.

Problem instance for Hospitals/Residents.

- agents: a set R of residents and a set H of hospitals
- a capacity $f(h)$ for each $h \in H$, giving the number of open jobs
- a strict (but maybe incomplete) preference list for each agent

Task: find a stable assignment $M : R \to H$, respecting capacities.

- M is stable $\Leftrightarrow$ no blocking pair exists for M
- $(r, h) \in R \times H$ is a blocking pair for M, if they are both beneficial for each other w.r.t. M, meaning that
  - r is unemployed or prefers h to $M(r)$, and
  - either h has less than $f(h)$ residents in M, or h prefers r to one of its residents in M.

The HOSPITALS/RESIDENTS problem

HOSPITALS/RESIDENTS: many-to-one version of STABLE MATCHING.

Problem instance for Hospitals/Residents.

- agents: a set R of residents and a set H of hospitals
- a capacity $f(h)$ for each $h \in H$, giving the number of open jobs
- a strict (but maybe incomplete) preference list for each agent

Task: find a stable assignment $M : R \to H$, respecting capacities.

- M is stable $\Leftrightarrow$ no blocking pair exists for M
- $(r, h) \in R \times H$ is a blocking pair for M, if they are both beneficial for each other w.r.t. M, meaning that
  - r is unemployed or prefers h to $M(r)$, and
  - either h has less than $f(h)$ residents in M, or h prefers r to one of its residents in M.

Complexity: solvable by an extension of the Gale-Shapley algorithm.

Hospitals/Residents with Couples (HRC)

Problem instance for HRC

- a set H of hospitals with a capacity function f
- a set C of couples, each $c \in C$ is a pair (c_1, c_2) of residents
- a set S of single residents
- strict preference lists (denoted by L)
  - hospitals rank acceptable residents
  - singles rank acceptable hospitals
  - couples rank acceptable pairs of hospitals
    example: $L(c) : (h_1, h_1), (h_2, h_3)$

Motivation:

- NRMP program in the US: assigning residents to hospitals
- US Navy detailing process

Hospitals/Residents with Couples (HRC)

Stability under HRC:

- assignment M is stable $\Leftrightarrow$ no blocking pair exists for M
- blocking "pair" for M:
  - **(a)** (s, h) where s is a single and h a hospital that are beneficial for each other w.r.t. M, or
  - **(b)** a couple $c = (c_1, c_2)$ and a pair of hospitals (h_1, h_2) such that
    - c prefers (h_1, h_2) to $M(c)$ or c is unmatched,
    - c_1 is beneficial for h_1 w.r.t. M, and
    - c_2 is beneficial for h_2 w.r.t. M.

  If $h_1 = h_2$, then we need to be more precise...

Hospitals/Residents with Couples (HRC)

Stability under HRC:

- assignment M is stable $\Leftrightarrow$ no blocking pair exists for M
- blocking "pair" for M:
  - **(a)** (s, h) where s is a single and h a hospital that are beneficial for each other w.r.t. M, or
  - **(b)** a couple $c = (c_1, c_2)$ and a pair of hospitals (h_1, h_2) such that
    - c prefers (h_1, h_2) to $M(c)$ or c is unmatched,
    - c_1 is beneficial for h_1 w.r.t. M, and
    - c_2 is beneficial for h_2 w.r.t. M.

    If $h_1 = h_2$, then we need to be more precise...

Complexity of HRC:

- It is NP-hard to decide whether a stable assignment exists. [Ronn]
- Stable assignments of various sizes may exist.
  The size of an assignment: the number of residents having a job.

Parameterized complexity of HRC

Parameter: the number $|C|$ of couples

- natural parameter: the number of couples is small in practical applications, compared to the number of singles
- the case $|C| = 0$ is linear-time solvable

Parameterized complexity of HRC

Parameter: the number $|C|$ of couples

- natural parameter: the number of couples is small in practical applications, compared to the number of singles
- the case $|C| = 0$ is linear-time solvable
- HRC is solvable in $|I|^{O(|C|)}$ time:
  1. We try each possible way to fix the assignment on each couple: $\approx |H|^{2|C|}$ possibilities.
  2. Assign as many singles as possible to the remaining jobs: easy!

$\Longrightarrow$ Polynomial-time solvable for each fixed $|C|$.
Is it FPT with parameter $|C|$?

Parameterized complexity of HRC

Parameter: the number $|C|$ of couples

- natural parameter: the number of couples is small in practical applications, compared to the number of singles
- the case $|C| = 0$ is linear-time solvable
- HRC is solvable in $|I|^{O(|C|)}$ time:
  1. We try each possible way to fix the assignment on each couple: $\approx |H|^{2|C|}$ possibilities.
  2. Assign as many singles as possible to the remaining jobs: easy!

$\Longrightarrow$ Polynomial-time solvable for each fixed $|C|$.
Is it FPT with parameter $|C|$?

Theorem

The existence version of Hospitals/Residents with Couples problem is W[1]-hard with parameter $|C|$.

Local Improvement for HRC

Suppose we already have a stable assignment, possibly not maximal.
Question: can we improve it efficiently?

- Such an algorithm would be extremely useful in practice!
- General form: as hard as the original problem.
- What if we only look for small modifications? → local search

Local Improvement for HRC

Suppose we already have a stable assignment, possibly not maximal.
Question: can we improve it efficiently?

- Such an algorithm would be extremely useful in practice!
- General form: as hard as the original problem.
- What if we only look for small modifications? $\rightarrow$ local search

Local Improvement for HRC

Input: an instance I of HRC, a stable assignment M for I, and $\Delta \in \mathbb{N}$.
Task: find a stable assignment M' for I such that

1. M' is larger than M, and
2. M' differs from M only for at most Δ residents.

Local Improvement for HRC

Suppose we already have a stable assignment, possibly not maximal.
Question: can we improve it efficiently?

- Such an algorithm would be extremely useful in practice!
- General form: as hard as the original problem.
- What if we only look for small modifications? → local search

Local Improvement for HRC

Input: an instance I of HRC, a stable assignment M for I, and $\Delta \in \mathbb{N}$.
Task: find a stable assignment M' for I such that

1. M' is larger than M, and
2. M' differs from M only for at most Δ residents.

Theorem [D. Marx and I. S.]

Local Improvement for HRC is FPT with parameters $(|C|, \Delta)$.

Matching with Couples

Simplification of HRC:

- We forget about the preferences.
- We aim for an acceptable assignment:
  each agent (single, couple, or hospital) must be assigned to an acceptable partner.
- Application in scheduling.

Matching with Couples

Simplification of HRC:

- We forget about the preferences.
- We aim for an acceptable assignment:
  each agent (single, couple, or hospital) must be assigned to an acceptable partner.
- Application in scheduling.

Maximum Matching with Couples

Input: An instance $I = (H, S, C, A, f)$ of MMC.

- H, S, C, f: as before (hospitals, singles, couples, capacities)
- A: list of acceptable partners for each agent.

Task: Find an assignment for I having maximum size.

Complexity of Maximum Matching with Couples

Classical complexity:

Theorem [Glass, Kellerer],[Biró,McDermid]

Maximum Matching with Couples is NP-hard, even if each capacity is 2.

Complexity of Maximum Matching with Couples

Classical complexity:

Theorem [Glass, Kellerer],[Biró,McDermid]

Maximum Matching with Couples is NP-hard, even if each capacity is 2.

Parameterized complexity with parameter $|C|$:

Theorem [D. Marx and I. S.]

Maximum Matching with Couples can be solved in randomized FPT time, if the parameter is the number $|C|$ of couples.

Special HRC with master list

Special HRC: simplifications based on real-world applications.

- Each resident has a score, yielding a master list for them $\Longrightarrow$ hospitals rank the residents according to their scores.
- Hospital pairs can be compatible or not.
- Preference list of a couple $c = (a, b)$:
  - a and b have individual preference lists.
  - $(h_1, h_2) \in L(c) \iff$ (i) $h_1 \in L(a)$ and $h_2 \in L(b)$, and (ii) h_1 and h_2 are compatible.
  - Responsive preferences: if $h_1 \succ_a h_3$ and $h_2 \succ_b h_4$, then $(h_1, h_2) \succ_c (h_3, h_4)$.
- Assumption: strict preferences.

Special HRC with master list

Special HRC: simplifications based on real-world applications.

- Each resident has a score, yielding a master list for them $\Longrightarrow$ hospitals rank the residents according to their scores.
- Hospital pairs can be compatible or not.
- Preference list of a couple $c = (a, b)$:
  - a and b have individual preference lists.
  - $(h_1, h_2) \in L(c) \iff$ (i) $h_1 \in L(a)$ and $h_2 \in L(b)$, and (ii) h_1 and h_2 are compatible.
  - Responsive preferences: if $h_1 \succ_a h_3$ and $h_2 \succ_b h_4$, then $(h_1, h_2) \succ_c (h_3, h_4)$.
- Assumption: strict preferences.

Special HRC

Input: a Special HRC instance I as described above.
Task: find a stable assignment for I.

Complexity of Special HRC

Classical complexity:

Theorem [P. Biró, R. W. Irving, and I. S.]

Special HRC is NP-hard, even if each hospital has capacity 1.

Complexity of Special HRC

Classical complexity:

Theorem [P. Biró, R. W. Irving, and I. S.]

Special HRC is NP-hard, even if each hospital has capacity 1.

Parameterized complexity with parameter $|C|$:

Theorem [P. Biró, R. W. Irving, and I. S.]

Special HRC can be solved in FPT time, if the parameter is the number $|C|$ of couples.

FPT-algorithm for SPECIAL HRC

1. Preprocessing phase: finds an initial assignment M_0.
   - Order the single residents decreasingly by their score, and assign each one to its most preferred hospital still available.
   - For each resident r, delete all hospitals h from $L(r)$ for which M_0 assigns $c(h)$ applicants better than r.
     $\longrightarrow$ if r is a member of a couple c, update the list $L(c)$ as well.

FPT-algorithm for SPECIAL HRC

1. Preprocessing phase: finds an initial assignment M_0.
   - Order the single residents decreasingly by their score, and assign each one to its most preferred hospital still available.
   - For each resident r, delete all hospitals h from $L(r)$ for which M_0 assigns $c(h)$ applicants better than r.
     $\longrightarrow$ if r is a member of a couple c, update the list $L(c)$ as well.
2. While there is an unassigned couple:
   - (a, b): the couple where a is the best member of any couple.
   - Prune $L(a, b)$ to contain at most $2|C|$ entries.
     Crucial step: must be done safely!
     $\Longrightarrow$ We only remove irrelevant entries.
   - Try each possibility for assigning (a, b), and update the capacities.

FPT-algorithm for SPECIAL HRC

1. Preprocessing phase: finds an initial assignment M_0.
   - Order the single residents decreasingly by their score, and assign each one to its most preferred hospital still available.
   - For each resident r, delete all hospitals h from $L(r)$ for which M_0 assigns $c(h)$ applicants better than r.
     $\longrightarrow$ if r is a member of a couple c, update the list $L(c)$ as well.
2. While there is an unassigned couple:
   - (a, b): the couple where a is the best member of any couple.
   - Prune $L(a, b)$ to contain at most $2|C|$ entries.
     Crucial step: must be done safely!
     $\Longrightarrow$ We only remove irrelevant entries.
   - Try each possibility for assigning (a, b), and update the capacities.
3. Assign all singles, taking them decreasingly by their scores.
   If the obtained assignment is stable, output it.

FPT-algorithm for Special HRC

1. Preprocessing phase: finds an initial assignment M_0.
   - Order the single residents decreasingly by their score, and assign each one to its most preferred hospital still available.
   - For each resident r, delete all hospitals h from $L(r)$ for which M_0 assigns $c(h)$ applicants better than r.
     $\longrightarrow$ if r is a member of a couple c, update the list $L(c)$ as well.
2. While there is an unassigned couple:
   - (a, b): the couple where a is the best member of any couple.
   - Prune $L(a, b)$ to contain at most $2|C|$ entries.
     Crucial step: must be done safely!
     $\Longrightarrow$ We only remove irrelevant entries.
   - Try each possibility for assigning (a, b), and update the capacities.
3. Assign all singles, taking them decreasingly by their scores.
   If the obtained assignment is stable, output it.

Running time: $(2|C|)^{|C|} \cdot |I|^{O(1)}$ $\Longrightarrow$ FPT with parameter $|C|$. ✓

Conclusion

Take home message.

Parameterized complexity is a powerful and rich framework to deal with computationally hard problems.

Further research:

- Plenty of work to do!
  Parameterized results related to stable matchings: <10 papers.
  Parameterized results in computational social choice: much more.
- We need more FPT results
  $\Longrightarrow$ find good parameters and tractable models!
- Multiple parameters $\Longrightarrow$ more detailed insight.
- Advanced techniques, e.g. kernelization.

Conclusion

Take home message.

Parameterized complexity is a powerful and rich framework to deal with computationally hard problems.

Further research:

- Plenty of work to do!
  Parameterized results related to stable matchings: <10 papers.
  Parameterized results in computational social choice: much more.
- We need more FPT results
  $\Longrightarrow$ find good parameters and tractable models!
- Multiple parameters $\Longrightarrow$ more detailed insight.
- Advanced techniques, e.g. kernelization.

Thank you!