CSPs and fixed-parameter tractability

Dániel Marx\(^1\)

\(^1\)Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI)
Budapest, Hungary

Algorithms Seminar
University of Bergen, Norway
June 7, 2013
Reactions to FPT

Typical graph algorithms researcher:

Hmm... Is my favorite graph problem FPT parameterized by the size of the solution/number of objects/etc.?
Reactions to FPT

**Typical graph algorithms researcher:**

Hmm... Is my favorite graph problem FPT parameterized by the size of the solution/number of objects/etc. ?

**Typical CSP researcher:**

SAT is trivially FPT parameterized by the number of variables. So why should I care?
Parameterizing \texttt{SAT}

**Trivial:** \texttt{3Sat} is FPT parameterized by the number of variables \( (2^k \cdot n^{O(1)} \text{ time algorithm}) \).

**Trivial:** \texttt{3Sat} is FPT parameterized by the number of clauses \( (2^{3k} \cdot n^{O(1)} \text{ time algorithm}) \).

What about \texttt{Sat} parameterized by the number \( k \) of clauses?
Parameterizing $\text{SAT}$

**Trivial:** $\text{3SAT}$ is FPT parameterized by the number of variables $(2^k \cdot n^{O(1)}$ time algorithm).

**Trivial:** $\text{3SAT}$ is FPT parameterized by the number of clauses $(2^{3k} \cdot n^{O(1)}$ time algorithm).

What about $\text{SAT}$ parameterized by the number $k$ of clauses?

Algorithm 1: Problem kernel

- If a clause has more than $k$ literals: can be ignored, removing it does not make the problem any easier.
- If every clause has at most $k$ literals: there are at most $k^2$ variables, use brute force.
Parameterizing \textbf{SAT}

\textbf{Trivial:} $3\text{Sat}$ is FPT parameterized by the number of \textbf{variables} ($2^k \cdot n^{O(1)}$ time algorithm).

\textbf{Trivial:} $3\text{Sat}$ is FPT parameterized by the number of \textbf{clauses} ($2^{3^k} \cdot n^{O(1)}$ time algorithm).

What about $\text{Sat}$ parameterized by the number $k$ of \textbf{clauses}?

\textbf{Algorithm 2: Bounded search tree}

- Pick a variable occurring both positively and negatively, branch on setting it to 0 or 1.
- In both branches, the number of clauses strictly decreases $\Rightarrow$ search tree of size $2^k$. 
Max Sat

- **Max Sat**: Given a formula, satisfy at least $k$ clauses.
- Polynomial for fixed $k$: guess the $k$ clauses, use the previous algorithm to check if they are satisfiable.
- Is the problem FPT?
**Max Sat**

- **Max Sat**: Given a formula, satisfy at least $k$ clauses.
- Polynomial for fixed $k$: guess the $k$ clauses, use the previous algorithm to check if they are satisfiable.
- Is the problem FPT?
  - YES: If there are at least $2k$ clauses, a random assignment satisfies $k$ clauses on average. Otherwise, use the previous algorithm.

This is not very insightful, can we say anything more interesting?
Above average MAX SAT

$m/2$ satisfiable clauses are guaranteed. But can we satisfy $m/2 + k$ clauses?
Above average $\text{MAX SAT}$

$m/2$ satisfiable clauses are guaranteed. But can we satisfy $m/2 + k$ clauses?

- Above average $\text{MAX SAT}$ (satisfy $m/2 + k$ clauses) is FPT [Mahajan and Raman 1999]
- Above average $\text{MAX } r\text{-SAT}$ (satisfy $(1 - 1/2^r)m + k$ clauses) is FPT [Alon et al. 2010]
- Satisfying $\sum_{i=1}^{m}(1 - 1/2^{r_i}) + k$ clauses is NP-hard for $k = 2$ [Crowston et al. 2012]
- Above average $\text{MAX } r\text{-LIN-2}$ (satisfy $m/2 + k$ linear equations) is FPT [Gutin et al. 2010]
- Permutation CSPs such as $\text{MAXIMUM ACYCLIC SUBGRAPH}$ and $\text{BETWEENNESS}$ [Gutin et al. 2010].
- ...
Boolean constraint satisfaction problems

Let $\Gamma$ be a set of Boolean relations. An $\Gamma$-formula is a conjunction of relations in $\Gamma$:

$$R_1(x_1, x_4, x_5) \land R_2(x_2, x_1) \land R_1(x_3, x_3, x_3) \land R_3(x_5, x_1, x_4, x_1)$$

$\text{SAT}(\Gamma)$

- Given: an $\Gamma$-formula $\varphi$
- Find: a variable assignment satisfying $\varphi$
Boolean constraint satisfaction problems

Let $\Gamma$ be a set of Boolean relations. An $\Gamma$-formula is a conjunction of relations in $\Gamma$:

$$R_1(x_1, x_4, x_5) \land R_2(x_2, x_1) \land R_1(x_3, x_3, x_3) \land R_3(x_5, x_1, x_4, x_1)$$

$\text{SAT}(\Gamma)$

- Given: an $\Gamma$-formula $\varphi$
- Find: a variable assignment satisfying $\varphi$

$\Gamma = \{a \neq b\} \Rightarrow \text{SAT}(\Gamma) = 2$-coloring of a graph
$\Gamma = \{a \lor b, a \lor \bar{b}, \bar{a} \lor \bar{b}\} \Rightarrow \text{SAT}(\Gamma) = 2$SAT
$\Gamma = \{a \lor b \lor c, a \lor b \lor \bar{c}, a \lor \bar{b} \lor \bar{c}, \bar{a} \lor \bar{b} \lor \bar{c}\} \Rightarrow \text{SAT}(\Gamma) = 3$SAT

Question: $\text{SAT}(\Gamma)$ is polynomial time solvable for which $\Gamma$?
It is NP-complete for which $\Gamma$?
Theorem [Schaefer 1978]

For every $\Gamma$, the SAT($\Gamma$) problem is polynomial-time solvable if one of the following holds, and NP-complete otherwise:

- Every relation is satisfied by the all 0 assignment
- Every relation is satisfied by the all 1 assignment
- Every relation can be expressed by a 2SAT formula
- Every relation can be expressed by a Horn formula
- Every relation can be expressed by an anti-Horn formula
- Every relation is an affine subspace over GF(2)
Schaefer’s Dichotomy Theorem (1978)

**Theorem [Schaefer 1978]**

For every $\Gamma$, the $\text{SAT}(\Gamma)$ problem is polynomial-time solvable if one of the following holds, and NP-complete otherwise:

- Every relation is satisfied by the all 0 assignment
- Every relation is satisfied by the all 1 assignment
- Every relation can be expressed by a 2SAT formula
- Every relation can be expressed by a Horn formula
- Every relation can be expressed by an anti-Horn formula
- Every relation is an affine subspace over GF(2)

This is surprising for two reasons:

- this family does not contain NP-intermediate problems and
- the boundary of polynomial-time and NP-hard problems can be cleanly characterized.
Other dichotomy results

- Approximability of \textsc{Max-Sat}, \textsc{Min-Unsat} [Khanna et al. 2001]
- Approximability of \textsc{MaxOnes-Sat}, \textsc{MinOnes-Sat} [Khanna et al. 2001]
- Generalization to 3-valued variables [Bulatov 2002]
- Inverse satisfiability [Kavvadias and Sideri, 1999]
- etc.

Celebrated open question: generalize Schaefer's result to relations over variables with non-Boolean, but fixed domain.

\textsc{CSP}(\Gamma): similar to \textsc{SAT}(\Gamma), but with non-Boolean domain.

Conjecture [Feder and Vardi 1998]

Let \( \Gamma \) be a finite set of relations over an arbitrary fixed domain. Then \( \textsc{CSP}(\Gamma) \) is either polynomial-time solvable or \( \text{NP-complete} \).
Other dichotomy results

- Approximability of \textbf{Max-Sat, Min-Unsat} [Khanna et al. 2001]
- Approximability of \textbf{MaxOnes-Sat, MinOnes-Sat} [Khanna et al. 2001]
- Generalization to 3-valued variables [Bulatov 2002]
- Inverse satisfiability [Kavvadias and Sideri, 1999]
- etc.

Celebrated open question: generalize Schaefer’s result to relations over variables with non-Boolean, but fixed domain.

\textbf{CSP}(\Gamma): similar to \textbf{SAT}(\Gamma), but with non-Boolean domain.

\textbf{Conjecture} [Feder and Vardi 1998]

Let $\Gamma$ be a finite set of relations over an arbitrary fixed domain. Then \textbf{CSP}(\Gamma) is either polynomial-time solvable or NP-complete.
Weighted problems

Parameterizing by the weight (= number of 1s) of the solution.

- **MinOnes-Sat(Γ)**:
  Find a satisfying assignment with weight at most $k$

- **ExactOnes-Sat(Γ)**:
  Find a satisfying assignment with weight exactly $k$

- **MaxOnes-Sat(Γ)**:
  Find a satisfying assignment with weight at least $k$

The first two problems can be always solved in $n^{O(k)}$ time, and the third one as well if $\text{Sat}(Γ)$ is in $\mathsf{P}$.

**Goal:** Characterize which languages $Γ$ make these problems FPT.
**Theorem [Marx 2004]**

$\textbf{ExactOnes-Sat}(\Gamma)$ is FPT if $\Gamma$ is weakly separable and $\text{W}[1]$-hard otherwise.

Examples of weakly separable constraints:
- affine constraints
- “0 or 5 out of 8”

Examples of not weakly separable constraints:
- $(\neg x \lor \neg y)$
- $x \rightarrow y$
- “0 or 4 out of 8”
Larger domains

What is the generalization of \textsc{ExactOnes-Sat}(\Gamma) to larger domains?

1. Find a solution with exactly \(k\) nonzero values (zeros constraint).
2. Find a solution where nonzero value \(i\) appears exactly \(k_i\) times (cardinality constraint).

\textbf{Theorem} [Bulatov and M. 2011]

For every \(\Gamma\) closed under substituting constants, \(\text{CSP}(\Gamma)\) with zeros constraint is FPT or W[1]-hard.
Larger domains

The following two problems are equivalent:

- **CSP(\(\Gamma\))** with cardinality constraint, where \(\Gamma\) contains only the relation \(R = \{00, 10, 02\}\).

- **Biclique**: Find a complete bipartite graph with \(k\) vertices on each side. The fixed-parameter tractability of **Biclique** is a notorious open problem (conjectured to be hard).
Larger domains

The following two problems are equivalent:

- **CSP(Γ)** with cardinality constraint, where Γ contains only the relation $R = \{00, 10, 02\}$.

- **Biclique**: Find a complete bipartite graph with $k$ vertices on each side. The fixed-parameter tractability of **Biclique** is a notorious open problem (conjectured to be hard).

So the best we can get at this point:

**Theorem [Bulatov and M. 2011]**

For every Γ closed under substituting constants, CSP(Γ) with cardinality constraint is FPT or **Biclique**-hard.
**MinOnes-Sat(\(\Gamma\))**

The bounded-search tree algorithm for **Vertex Cover** can be generalized to **MinOnes-Sat**.

<table>
<thead>
<tr>
<th>Observation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MinOnes-Sat((\Gamma))</strong> is FPT for every finite (\Gamma).</td>
</tr>
</tbody>
</table>
**MinOnes-Sat(Γ)**

The bounded-search tree algorithm for **Vertex Cover** can be generalized to **MinOnes-Sat**.

<table>
<thead>
<tr>
<th>Observation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MinOnes-Sat(Γ)</strong> is FPT for every finite Γ.</td>
</tr>
</tbody>
</table>

But can we solve the problem simply by preprocessing?

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>A polynomial kernel is a polynomial-time reduction creating an equivalent instance whose size is polynomial in k.</td>
</tr>
</tbody>
</table>

**Goal**: Characterize the languages Γ for which **MinOnes-Sat(Γ)** has a polynomial kernel.

**Example**: the special case **d-Hitting Set** (where Γ contains only $R = x_1 \lor \cdots \lor x_d$) has a polynomial kernel.
Sunflower lemma

**Definition**
Sets $S_1, S_2, \ldots, S_k$ form a **sunflower** if the sets $S_i \setminus (S_1 \cap S_2 \cap \cdots \cap S_k)$ are disjoint.

**Lemma** [Erdős and Rado, 1960]
If the size of a set system is greater than $(p - 1)^d \cdot d!$ and it contains only sets of size at most $d$, then the system contains a sunflower with $p$ petals.
Sunflowers and \(d\)-Hitting Set

\(d\)-Hitting Set

Given a collection \(S\) of sets of size at most \(d\) and an integer \(k\), find a set \(S\) of \(k\) elements that intersects every set of \(S\).

Reduction Rule

Suppose more than \(k + 1\) sets form a sunflower.

- If the sets are disjoint \(\Rightarrow\) No solution.
- Otherwise, keep only \(k + 1\) of the sets.
Kernelization for general $\text{MINONES-SAT}(\Gamma)$ generalizes the sunflower reduction, and requires that $\Gamma$ is “mergeable.”

**Theorem** [Kratsch and Wahlström 2010]

1. If $\text{MINONES-SAT}(\Gamma)$ is polynomial-time solvable or $\Gamma$ is mergeable, then $\text{MINONES-SAT}(\Gamma)$ has a polynomial kernelization.

2. If $\text{MINONES-SAT}(\Gamma)$ is NP-hard and $\Gamma$ is not mergeable, then $\text{MINONES-SAT}(\Gamma)$ does not have a polynomial kernel, unless the polynomial hierarchy collapses.
Dichotomy for kernelization

Similar results for other problems:

**Theorem [Kratsch, M., Wahlström 2010]**

- If $\Gamma$ has property $X$, then $\text{MaxOnes-Sat}(\Gamma)$ has a polynomial kernel, and otherwise no (unless the polynomial hierarchy collapses).
- If $\Gamma$ has property $Y$, then $\text{ExactOnes-Sat}(\Gamma)$ has a polynomial kernel, and otherwise no (unless the polynomial hierarchy collapses).
Local search

Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.
Local search

Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.

Problem:
Local search can stop at a local optimum (no better solution in the local neighborhood).

More sophisticated variants: simulated annealing, tabu search, etc.
Local search

Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.
Local search

Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.

Problem: local search can stop at a local optimum (no better solution in the local neighborhood).
Local search

Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.
Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.
Local search

Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.
Local search

Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.
Local search

Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.
Local search

Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.
Local search

Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.
Local search

Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.
Local search

Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.

**Problem:** local search can stop at a local optimum (no better solution in the local neighborhood).

More sophisticated variants: simulated annealing, tabu search, etc.
Local neighborhood

The local neighborhood is defined in a problem-specific way:

- For TSP, the neighbors are obtained by swapping 2 cities or replacing 2 edges.
- For a problem with 0-1 variables, the neighbors are obtained by flipping a single variable.
- For subgraph problems, the neighbors are obtained by adding/removing one edge.

More generally: reordering $k$ cities, flipping $k$ variables, etc.
Local neighborhood

The local neighborhood is defined in a problem-specific way:

- For TSP, the neighbors are obtained by swapping 2 cities or replacing 2 edges.
- For a problem with 0-1 variables, the neighbors are obtained by flipping a single variable.
- For subgraph problems, the neighbors are obtained by adding/removing one edge.

More generally: reordering $k$ cities, flipping $k$ variables, etc.

Larger neighborhood (larger $k$):

- algorithm is less likely to get stuck in a local optimum,
- it is more difficult to check if there is a better solution in the neighborhood.
Searching the neighborhood

Question: Is there an efficient way of finding a better solution in the $k$-neighborhood?

We study the complexity of the following problem:

$k$-step Local Search

Input: instance $I$, solution $x$, integer $k$
Find: A solution $x'$ with $\text{dist}(x, x') \leq k$ that is “better” than $x$.  

Remark 1: If the optimization problem is hard, then it is unlikely that this local search problem is polynomial-time solvable: otherwise we would be able to find an optimum solution.

Remark 2: Size of the $k$-neighborhood is usually $n^{O(k)} \Rightarrow$ local search is polynomial-time solvable for every fixed $k$, but this is not practical for larger $k$. 

20
Searching the neighborhood

**Question:** Is there an efficient way of finding a better solution in the $k$-neighborhood?

We study the complexity of the following problem:

<table>
<thead>
<tr>
<th><strong>k-step Local Search</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> instance $I$, solution $x$, integer $k$</td>
</tr>
<tr>
<td><strong>Find:</strong> A solution $x'$ with $\text{dist}(x, x') \leq k$ that is “better” than $x$.</td>
</tr>
</tbody>
</table>

**Remark 1:** If the optimization problem is hard, then it is unlikely that this local search problem is polynomial-time solvable: otherwise we would be able to find an optimum solution.

**Remark 2:** Size of the $k$-neighborhood is usually $n^{O(k)} \Rightarrow$ local search is polynomial-time solvable for every fixed $k$, but this is not practical for larger $k$. 
The question that we want to investigate:

**Question**

Is $k$-step Local Search FPT for a particular problem?

If yes, then local search algorithms can consider larger neighborhoods, improving their efficiency.

**Important:** $k$ is the number of allowed changes and **not** the size of the solution. Relevant even if solution size is large.
$k$-step Local Search

The question that we want to investigate:

**Question**

Is $k$-step Local Search FPT for a particular problem?

If yes, then local search algorithms can consider larger neighborhoods, improving their efficiency.

**Important:** $k$ is the number of allowed changes and not the size of the solution. Relevant even if solution size is large.

**Examples:**

- Local search is easy: it is FPT to find a larger independent set in a planar graph with at most $k$ exchanges [Fellows et al. 2008].
- Local search is hard: it is W[1]-hard to check if it is possible to obtain a shorter TSP tour by replacing at most $k$ arcs [M. 2008].
Local search for SAT

Simple satisfiability:

<table>
<thead>
<tr>
<th>Theorem [Dantsin et al. 2002]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finding a satisfying assignment in the $k$-neighborhood for $q$-SAT is FPT.</td>
</tr>
</tbody>
</table>
Local search for **SAT**

Simple satisfiability:

**Theorem [Dantsin et al. 2002]**
Finding a satisfying assignment in the $k$-neighborhood for $q$-$\text{SAT}$ is FPT.

An optimization problem:

**Theorem [Szeider 2011]**
Finding a better assignment in the $k$-neighborhood for $\text{Max 2-SAT}$ is W[1]-hard.
Local search for \textsc{Sat}

Simple satisfiability:

Theorem [Dantsin et al. 2002]
Finding a satisfying assignment in the $k$-neighborhood for $q$-\textsc{Sat} is FPT.

An optimization problem:

Theorem [Szeider 2011]
Finding a better assignment in the $k$-neighborhood for Max 2-\textsc{Sat} is $W[1]$-hard.

A family of problems:

Theorem [Krokhin and M. 2008]
Dichotomy results for Min\textsc{Ones-Sat}(\Gamma).
Strict vs. permissive

Something strange: for some problems (e.g., Vertex Cover on bipartite graphs), local search is hard, even though the problem is polynomial-time solvable.
Something strange: for some problems (e.g., \textsc{Vertex Cover} on bipartite graphs), local search is hard, even though the problem is polynomial-time solvable.

**Strict \( k \)-step Local Search**

<table>
<thead>
<tr>
<th>Input: instance ( I ), solution ( x ), integer ( k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find: A solution ( x' ) with ( \text{dist}(x, x') \leq k ) that is “better” than ( x ).</td>
</tr>
</tbody>
</table>
Something strange: for some problems (e.g., Vertex Cover on bipartite graphs), local search is hard, even though the problem is polynomial-time solvable.

## Strict $k$-step Local Search

**Input:** instance $I$, solution $x$, integer $k$  
**Find:** A solution $x'$ with $\text{dist}(x, x') \leq k$ that is “better” than $x$.

## Permissive $k$-step Local Search

**Input:** instance $I$, solution $x$, integer $k$  
**Find:** Any solution $x'$ “better” than $x$, if there is such a solution at distance at most $k$. 
Constraint Satisfaction Problems (CSP)

A CSP instance is given by describing the

- variables,
- domain of the variables,
- constraints on the variables.

**Task:** Find an assignment that satisfies every constraint.

\[ I = C_1(x_1, x_2, x_3) \land C_2(x_2, x_4) \land C_3(x_1, x_3, x_4) \]
Constraint Satisfaction Problems (CSP)

A CSP instance is given by describing the

- variables,
- domain of the variables,
- constraints on the variables.

**Task:** Find an assignment that satisfies every constraint.

\[ I = C_1(x_1, x_2, x_3) \land C_2(x_2, x_4) \land C_3(x_1, x_3, x_4) \]

**Examples:**

- **3Sat:** 2-element domain, every constraint is ternary
- **Vertex Coloring:** domain is the set of colors, binary constraints
- **k-Clique** (in graph \( G \)): \( k \) variables, domain is the vertices of \( G \), \( \binom{k}{2} \) binary constraints
Graphs and hypergraphs related to CSP

**Gaifman/primal graph:** vertices are the variables, two variables are adjacent if they appear in a common constraint.

**Incidence graph:** bipartite graph, vertices are the variables and constraints.

**Hypergraph:** vertices are the variables, constraints are the hyperedges.

\[ I = C_1(x_2, x_1, x_3) \land C_2(x_4, x_3) \land C_3(x_1, x_4, x_2) \]
Theorem [Freuder 1990]

For every fixed \( k \), CSP can be solved in polynomial time if the primal graph of the instance has treewidth at most \( k \).

**Note:** The running time is \(|D|^{O(k)}\), which is not FPT parameterized by treewidth.
Treewidth and CSP

Theorem [Freuder 1990]

For every fixed $k$, CSP can be solved in polynomial time if the primal graph of the instance has treewidth at most $k$.

Note: The running time is $|D|^{O(k)}$, which is not FPT parameterized by treewidth.

We know that binary CSP$(G)$ is polynomial-time solvable for every class $G$ of graphs with bounded treewidth. Are there other polynomial cases?
Question: Which graph properties lead to polynomial-time solvable CSP instances?

Systematic study:
- Binary CSP: Every constraint is of arity 2.
- CSP($\mathcal{G}$): problem restricted to binary CSP instances with primal graph in $\mathcal{G}$.
- Which classes $\mathcal{G}$ make CSP($\mathcal{G}$) FPT?
- E.g., if $\mathcal{G}$ is the set of trees, then it is easy, if $\mathcal{G}$ is the set of 3-regular graphs, then it is W[1]-hard.
Dichotomy for binary CSP

Complete answer for every class $G$:

**Theorem [Grohe-Schwentick-Segoufin 2001]**

Let $G$ be a computable class of graphs.

1. If $G$ has bounded treewidth, then $\text{CSP}(G)$ is FPT parameterized by number of variables (in fact, polynomial-time solvable).

2. If $G$ has unbounded treewidth, then $\text{CSP}(G)$ is $W[1]$-hard parameterized by number of variables.

*Note:* In (2), $\text{CSP}(G)$ is not necessarily NP-hard.
Dichotomy for binary CSP

Complete answer for every class $G$:

**Theorem [Grohe-Schwentick-Segoufin 2001]**

Let $G$ be a recursively enumerable class of graphs. Assuming $\text{FPT} \neq \text{W}[1]$, the following are equivalent:

- Binary $\text{CSP}(G)$ is polynomial-time solvable.
- Binary $\text{CSP}(G)$ is FPT.
- $G$ has bounded treewidth.

**Note:** Fixed-parameter tractability does not give us more power here than polynomial-time solvability!
Combination of parameters

CSP can be parameterized by many (combination of) parameters.

Examples:

- CSP is \( W[1] \)-hard parameterized by the treewidth of the primal graph.
- CSP is FPT parameterized by the treewidth of the primal graph and the domain size.
Combination of parameters

CSP can be parameterized by many (combination of) parameters.

**Examples:**

- CSP is W[1]-hard parameterized by the treewidth of the primal graph.
- CSP is FPT parameterized by the treewidth of the primal graph and the domain size.

[Samer and Szeider 2010] considered 11 parameters and determined the complexity of CSP by any subset of these parameters.

- \( \text{tw} \): treewidth of primal graph
- \( \text{tw}^d \): tw of dual graph
- \( \text{tw}^* \): tw of incidence graph
- \( \text{vars} \): number of variables
- \( \text{dom} \): domain size
- \( \text{cons} \): number of constraints
- \( \text{arity} \): maximum arity
- \( \text{dep} \): largest relation size
- \( \text{deg} \): largest variable occurrence
- \( \text{ovl} \): largest overlap between scopes
- \( \text{diff} \): largest difference between scopes
Summary

- Fixed-parameter tractability results for SAT and CSPs do exist.
- Choice of parameter is not obvious.
- Above average parameterization.
- Local search.
- Parameters related to the graph of the constraints.