CSPs and fixed-parameter tractability

Dániel Marx[1]

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

International Workshop on Approximation, Parameterized and EXact algorithms
Riga, Latvia
July 7, 2013

Parameterized problems

Main idea

Instead of expressing the running time as a function $T(n)$ of n, we express it as a function $T(n, k)$ of the input size n and some parameter k of the input.

In other words: we do not want to be efficient on all inputs of size n, only for those where k is small.

Parameterized problems

Main idea

Instead of expressing the running time as a function $T(n)$ of n, we express it as a function $T(n, k)$ of the input size n and some parameter k of the input.

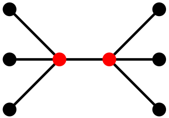
In other words: we do not want to be efficient on all inputs of size n, only for those where k is small.

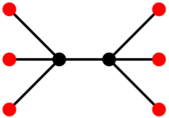
What can be the parameter k?

- The size k of the solution we are looking for.
- The maximum degree of the input graph.
- The dimension of the point set in the input.
- The length of the strings in the input.
- The length of clauses in the input Boolean formula.
- $\ldots$

Parameterized complexity

Problem:	VERTEX COVER	INDEPENDENT SET
Input:	Graph G, integer k	Graph G, integer k
Question:	Is it possible to cover the edges with k vertices?	Is it possible to find k independent vertices?

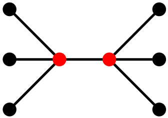


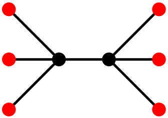


Complexity:	NP-complete	NP-complete

Parameterized complexity

Problem:	VERTEX COVER	INDEPENDENT SET
Input:	Graph G, integer k	Graph G, integer k
Question:	Is it possible to cover the edges with k vertices?	Is it possible to find k independent vertices?

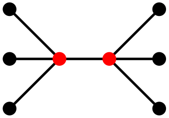


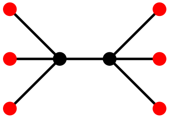


Complexity:	NP-complete	NP-complete
Brute force:	$O(n^k)$ possibilities	$O(n^k)$ possibilities

Parameterized complexity

Problem:	VERTEX COVER	INDEPENDENT SET
Input:	Graph G, integer k	Graph G, integer k
Question:	Is it possible to cover the edges with k vertices?	Is it possible to find k independent vertices?





Complexity:	NP-complete	NP-complete
Brute force:	$O(n^k)$ possibilities	$O(n^k)$ possibilities
	$O(2^k n^2)$ algorithm exists exists	No $n^{o(k)}$ algorithm known

Bounded search tree method

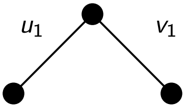
Algorithm for VERTEX COVER:

$e_1 = u_1 v_1$

Bounded search tree method

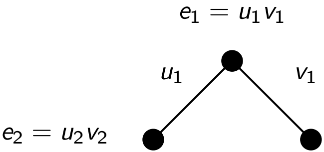
Algorithm for Vertex Cover:

$e_1 = u_1 v_1$



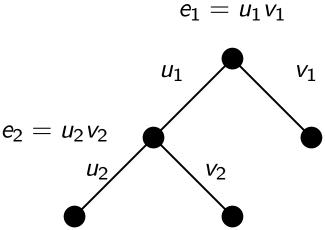
Bounded search tree method

Algorithm for VERTEX COVER:



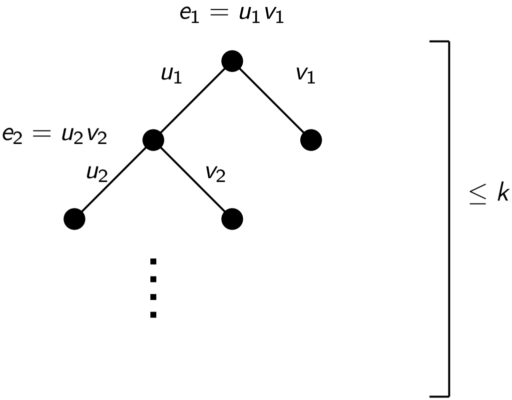
Bounded search tree method

Algorithm for VERTEX COVER:



Bounded search tree method

Algorithm for Vertex Cover:



Height of the search tree $\leq k$ $\Rightarrow$ at most 2^k leaves $\Rightarrow$ $2^k \cdot n^{O(1)}$ time algorithm.

Fixed-parameter tractability

Main definition

A parameterized problem is **fixed-parameter tractable (FPT)** if there is an $f(k)n^c$ time algorithm for some constant c.

Main goal of parameterized complexity: to find FPT problems.

Fixed-parameter tractability

Main definition

A parameterized problem is **fixed-parameter tractable (FPT)** if there is an $f(k)n^c$ time algorithm for some constant c.

Main goal of parameterized complexity: to find FPT problems.

Examples of NP-hard problems that are FPT:

- Finding a vertex cover of size k.
- Finding a path of length k.
- Finding k disjoint triangles.
- Drawing the graph in the plane with k edge crossings.
- Finding disjoint paths that connect k pairs of points.
- ...

W[1]-hardness

Negative evidence similar to NP-completeness. If a problem is **W[1]-hard,** then the problem is not FPT unless FPT=W[1].

Some W[1]-hard problems:

- Finding a clique/independent set of size k.
- Finding a dominating set of size k.
- Finding k pairwise disjoint sets.
- $\ldots$

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 SAT

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

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

What about SAT parameterized by the number k of clauses?

Parameterizing SAT

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

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

What about 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 SAT

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

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

What about SAT parameterized by the number k of clauses?

Algorithm 2: Bounded search tree

- Pick a variable occuring 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 Max Sat

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

- Above average Max Sat (satisfy $m/2 + k$ clauses) is FPT [Mahajan and Raman 1999]
- Above average Max r-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 Max r-Lin-2 (satisfy $m/2 + k$ linear equations) is FPT [Gutin et al. 2010]
- Permutation CSPs such as Maximum Acyclic Subgraph and Betweenness [Gutin et al. 2010].
- ...

Boolean constraint satisfaction problems

Let Γ be a set of **Boolean** relations. An Γ-formula is a conjunction of relations in Γ:

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

SAT(Γ)

- Given: an Γ-formula φ
- Find: a variable assignment satisfying φ

Boolean constraint satisfaction problems

Let Γ be a set of **Boolean** relations. An Γ-formula is a conjunction of relations in Γ:

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

SAT(Γ)

- Given: an Γ-formula φ
- Find: a variable assignment satisfying φ

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

Question: SAT(Γ) is polynomial time solvable for which Γ?
It is NP-complete for which Γ?

Schaefer's Dichotomy Theorem (1978)

Theorem [Schaefer 1978]

For every Γ, the SAT(Γ) 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 Γ, the $\mathrm{SAT}(\Gamma)$ 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 MAX-SAT, MIN-UNSAT [Khanna et al. 2001]
- Approximability of MAXONES-SAT, MINONES-SAT [Khanna et al. 2001]
- Generalization to 3-valued variables [Bulatov 2002]
- Inverse satisfiability [Kavvadias and Sideri, 1999]
- etc.

Other dichotomy results

- Approximability of MAX-SAT, MIN-UNSAT [Khanna et al. 2001]
- Approximability of 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.

CSP(Γ): similar to SAT(Γ), but with non-Boolean domain.

Conjecture [Feder and Vardi 1998]

Let Γ be a finite set of relations over an arbitrary fixed domain. Then CSP(Γ) 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 Sat(Γ) is in P.

Goal: Characterize which languages Γ make these problems FPT.

ExactOnes-Sat(Γ)

Theorem [Marx 2004]

ExactOnes-Sat(Γ) is FPT if Γ is weakly separable and W[1]-hard otherwise.

Examples of weakly separable constraints:

- affine constraints
- "0 or 5 out of 8"

Examples of not weakly separable constraints:

- $(\neg x \lor \neg y)$
- $x \to y$
- "0 or 4 out of 8"

Larger domains

What is the generalization of ExactOnes-Sat(Γ) 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).

Theorem [Bulatov and M. 2011]

For every Γ closed under substituting constants, CSP(Γ) with zeros constraint is FPT or W[1]-hard.

(E.g., if $R(x_1, x_2, x_3, x_4) \in \Gamma$, then $R(x_1, 3, x_3, 0) \in \Gamma$.)

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).

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(Γ)

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

Observation

MinOnes-Sat(Γ) is FPT for every finite Γ.

MinOnes-Sat(Γ)

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

Observation

MinOnes-Sat(Γ) is FPT for every finite Γ.

But can we solve the problem simply by preprocessing?

Definition

A polynomial kernel is a polynomial-time reduction creating an equivalent instance whose size is polynomial in k.

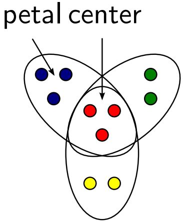
Goal: Characterize the languages Γ for which MinOnes-Sat(Γ) has a polynomial kernel.

Example: the special case d-Hitting Set (where Γ contains only $R = x_1 \vee \cdots \vee x_d$) 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.



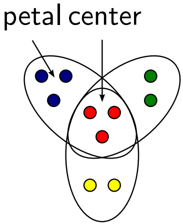
Lemma [Erdős and Rado, 1960]

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 $\mathcal{S}$ of sets of size at most d and an integer k, find a set S of k elements that intersects every set of $\mathcal{S}$.



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.

Dichotomy for kernelization

Kernelization for general MinOnes-Sat(Γ) generalizes the sunflower reduction, and requires that Γ is "mergeable."

Theorem [Kratsch and Wahlström 2010]

(1) If MinOnes-Sat(Γ) is polynomial-time solvable or Γ is mergeable, then MinOnes-Sat(Γ) has a polynomial kernelization.

(2) If MinOnes-Sat(Γ) is NP-hard and Γ is not mergebable, then MinOnes-Sat(Γ) 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 Γ has property X, then MaxOnes-Sat(Γ) has a polynomial kernel, and otherwise no (unless the polynomial hierarchy collapses).
- If Γ has property Y, then ExactOnes-Sat(Γ) has a polynomial kernel, and otherwise no (unless the polynomial hierarchy collapses).

Local search

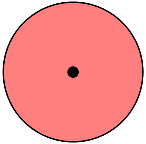
Local search

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

Local search

Local search

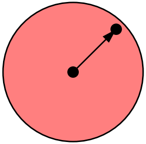
Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.



Local search

Local search

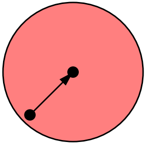
Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.



Local search

Local search

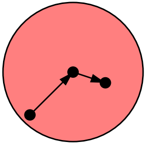
Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.



Local search

Local search

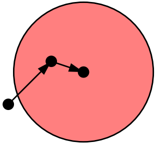
Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.



Local search

Local search

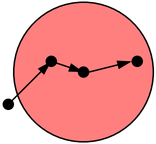
Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.



Local search

Local search

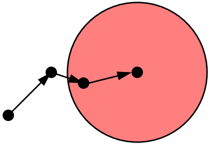
Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.



Local search

Local search

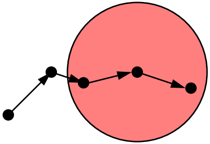
Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.



Local search

Local search

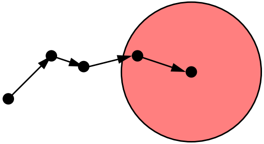
Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.



Local search

Local search

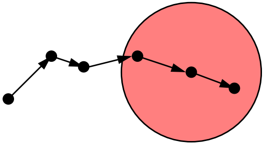
Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.



Local search

Local search

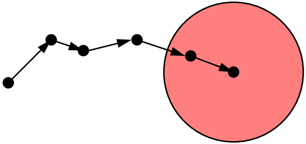
Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.



Local search

Local search

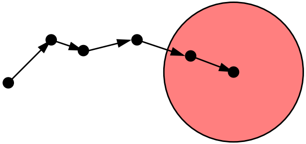
Walk in the solution space by iteratively replacing the current solution with a better solution in the local neighborhood.



Local search

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.

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.

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.

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:

Theorem [Dantsin et al. 2002]

Finding a satisfying assignment in the k-neighborhood for q-SAT is FPT.

Local search for SAT

Simple satisfiability:

Theorem [Dantsin et al. 2002]

Finding a satisfying assignment in the k-neighborhood for q-SAT is FPT.

An optimization problem:

Theorem [Szeider 2011]

Finding a better assignment in the k-neighborhood for MAX 2-SAT is W[1]-hard.

Local search for SAT

Simple satisfiability:

Theorem [Dantsin et al. 2002]

Finding a satisfying assignment in the k-neighborhood for q-SAT is FPT.

An optimization problem:

Theorem [Szeider 2011]

Finding a better assignment in the k-neighborhood for MAX 2-SAT is W[1]-hard.

A family of problems:

Theorem [Krokhin and M. 2008]

Dichotomy results for MINONES-SAT(Γ).

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.

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.

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.

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.

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.

$$I = C_1(x_1, x_2, x_3) \wedge C_2(x_2, x_4) \wedge C_3(x_1, x_3, x_4)$$

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.

$$I = C_1(x_1, x_2, x_3) \wedge C_2(x_2, x_4) \wedge C_3(x_1, x_3, x_4)$$

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

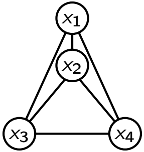
Graphs and hypergraphs related to CSP

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

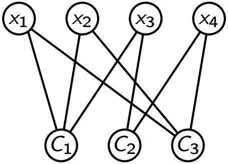
Incidence graph: bipartite graph, vertices are the variables and constraints.

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

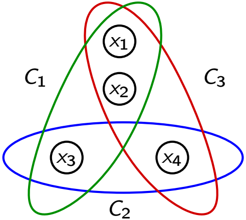
$$I = C_1(x_2, x_1, x_3) \wedge C_2(x_4, x_3) \wedge C_3(x_1, x_4, x_2)$$



Primal graph



Incidence graph



Hypergraph

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.

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 $\mathrm{CSP}(\mathcal{G})$ is polynomial-time solvable for every class $\mathcal{G}$ of graphs with bounded treewidth. Are there other polynomial cases?

Tractable structures

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($\mathcal{G}$) polynomial-time solvable?
- 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 parameterized by the number of variables (hence unlikely to be polynomial-time solvable).

Dichotomy for binary CSP

Complete answer for **every** class $\mathcal{G}$:

Theorem [Grohe-Schwentick-Segoufin 2001]

Let $\mathcal{G}$ be a computable class of graphs.

(1) If $\mathcal{G}$ has bounded treewidth, then CSP($\mathcal{G}$) is polynomial-time solvable.

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

Note: In (2), CSP($\mathcal{G}$) is not necessarily NP-hard.

Dichotomy for binary CSP

Complete answer for **every** class $\mathcal{G}$:

Theorem [Grohe-Schwentick-Segoufin 2001]

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

- Binary CSP($\mathcal{G}$) is polynomial-time solvable.
- Binary CSP($\mathcal{G}$) is FPT parameterized by the number of variables.
- $\mathcal{G}$ 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.

tw:	treewidth of primal graph
tw^d:	tw of dual graph
tw^*:	tw of incidence graph
vars:	number of variables
dom:	domain size
cons:	number of constraints
arity:	maximum arity
dep:	largest relation size
deg:	largest variable occurrence
ovl:	largest overlap between scopes
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.