Structural Tractability of Constraint Optimization

Gianluigi Greco and Francesco Scarcello
University of Calabria, Italy
Classically, CSP: Constraint Satisfaction Problem

However, sometimes a solution is enough to “satisfy” (constraints), but not enough to make (users) “happy”

Hence, several variants of the basic CSP framework:

- E.g., fuzzy, probabilistic, weighted, lexicographic, penalty, valued, semiring-based, …
## Classical CSPs

- **Set of variables** \( \{X_1, \ldots, X_{26}\} \)
- **Set of constraint scopes**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td></td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
</tr>
</tbody>
</table>

- **Set of constraint relations**

<table>
<thead>
<tr>
<th>PARIS</th>
<th>PANDA</th>
<th>LAURA</th>
<th>ANITA</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIMBO</td>
<td>LINGO</td>
<td>PETRA</td>
<td>PAMPA</td>
</tr>
<tr>
<td>PETE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Puzzles for Expert Players...

The puzzle in general admits more than one solution...

E.g., find the solution that **minimizes** the total number of vowels occurring in the words

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
</tr>
</tbody>
</table>
Each mapping \textit{variable-value} has a cost. Then, find an assignment:

- Satisfying all the constraints, and
- Having the minimum total cost.
Typical Constraint Optimization Problems

Each mapping variable-value has a cost.
Then, find an assignment:
  - Satisfying all the constraints, and
  - Having the minimum total cost.

Each tuple has a cost.
Then, find an assignment:
  - Satisfying all the constraints, and
  - Having the minimum total cost.

Tuple-oriented problems (weighted CSPs) may be transformed easily to variable-value problems, without altering structural properties.
From WCSP to CSOP instances
From WCSP to CSOP instances

The mapping:

- Is feasible in linear time
- Preserves the solutions
- Preserves structural properties
  (any fresh variable occurs just once)
Another Example: Combinatorial Auctions
Combinatorial Auctions
Combinatorial Auctions
Winner Determination Problem

Determine the outcome that maximizes the sum of accepted bid prices
Combinatorial Auctions

Total € 180.--
Note that there are two possible encodings:

- Items as variables (as previous pictures suggest)
- Bids as variables, and constraint scopes associated with items (the bids where they occur)

Note that the latter encoding yields a natural example of Constraint Optimization Formula

- Costs are assigned to bids
- This is the only encoding with nice structural tractability properties
**Constraint Formulas: Formal Framework**

**Evaluation Functions**

- $\mathbb{D}$ domain of values, $\succeq$ a total order over it
- *evaluation function* $\mathcal{F}$: a tuple $\langle w, \oplus \rangle$ with $w : \text{Var} \times \mathcal{U} \mapsto \mathbb{D}$
- $\oplus$ commutative, associative, and closed binary operator with an identity element over $\mathbb{D}$
- $\mathcal{F}(\theta) = \bigoplus_{X/u \in \theta} w(X, u)$ (with $\mathcal{F}(\emptyset)$ being the identity w.r.t. $\oplus$)

**Monotone Functions**

$\mathcal{F}(\theta) \succeq \mathcal{F}(\theta') \quad \Rightarrow \quad \mathcal{F}(\theta) \oplus \mathcal{F}(\theta'') \succeq \mathcal{F}(\theta') \oplus \mathcal{F}(\theta''), \quad \forall \theta''$
We often want to express more preferences, e.g.,
- minimize cost, then minimize total time, or
- maximize the profit, then minimize the number of different buyers, or transactions

Formally,

- \( L = [\mathcal{F}_1, \ldots, \mathcal{F}_m] \)
- \( L(\theta) \) denotes \((\mathcal{F}_1(\theta), \ldots, \mathcal{F}_m(\theta)) \in \mathbb{D}_1 \times \cdots \times \mathbb{D}_m\)

Compare vectors by the lexicographical precedence relationship \( \preceq_L \) (Cascade of preferences)
Example

- Domain $\mathcal{U} = \{a, b\}$
- Constraint Formula $\Phi = r_1(X_1, X_2, X_3) \land r_2(X_1, X_4) \land r_3(X_4, X_3)$

- $L = [\mathcal{F}_1, \mathcal{F}_2]$, with $\mathcal{F}_1 = \langle w_1, + \rangle$ and $\mathcal{F}_2 = \langle w_2, + \rangle$ where:
  - $w_1(X_1/a) = 1$, $w_1(X_1/b) = 0$, and
  - $w_1(X_i/u) = 0$, $\forall i \in \{2, 3, 4\}$ and $\forall u \in \mathcal{U}$;
  - $w_2(X_4/a) = 0$, $w_2(X_4/b) = 1$, and
  - $w_2(X_i/u) = 0$, $\forall i \in \{1, 2, 3\}$ and $\forall u \in \mathcal{U}$.
- Constraint Optimization Formula $\Phi_L$
Consider now the three substitutions

- \( \theta_1 = \{X_1/b, X_2/b, X_3/b, X_4/b\} \),
- \( \theta_2 = \{X_1/b, X_2/b, X_3/b, X_4/a\} \),
- \( \theta_3 = \{X_1/b, X_2/a, X_3/a, X_4/b\} \)

- \( L(\theta_1) = (0, 1) \), \( L(\theta_2) = (0, 0) \), and \( L(\theta_3) = (0, 1) \).
- Thus, \( \theta_1 \succeq_L \theta_2 \) and \( \theta_3 \succeq_L \theta_2 \).

Note that \( \succeq_L \) is not antisymmetric:
\( \theta_1 \succeq_L \theta_3 \) and \( \theta_3 \succeq_L \theta_1 \), but \( \theta_1 \neq \theta_3 \)
Linearization (following [Brafman et al’10])

- $\succeq_\mathcal{U}$ an arbitrary total order defined over $\mathcal{U}$
- $\ell = [X_1, \ldots, X_n]$ a list including all the variables in $\text{Var}$

Define the total order $\succeq^\ell_L$

- ties in $\succeq_L$ are resolved according to the lexicographical precedence relationship $\ell$ over variables and the total order $\succeq_\mathcal{U}$ over $\mathcal{U}$
- $\succeq^\ell_L$ is a refinement of $\succeq_L$

In our example

Assume the linearization $\ell = [X_1, X_2, X_3, X_4]$ with $a \succ_\mathcal{U} b$. Then, $\succeq^\ell_L$ is the total order: $\theta_3 \succeq^\ell_L \theta_1 \succeq^\ell_L \theta_2$. 
Input: $\Phi_L, DB, \ell$

**Min**: Compute the best solution (if any)

**Top-K**: Compute the $K$-best solutions,
- $K$ is a natural number additionally given in input

**Next**: Compute the best solution following $\theta$
- $\theta$ is a solution additionally given in input
Objectives

Find efficient algorithms for all these problems

In this paper: exploit structural properties
- Identify large islands of tractability
- Chart the tractability frontier
  (at least for bounded arity instances)
Known Good Results

- **Min** is feasible in polynomial-time (P) and **Top-K** with polynomial delay (WPD) for formulas equipped with **monotonic functions** and whose structures are:
  - Acyclic hypergraphs [Kimelfeld and Sagiv’06]
  - Bounded treewidth [S. de Givry et al’06, Flerova and Dechter’10]
  - Bounded hypertree-width [S. Ndiaye et al’08, Gottlob et al’09]

**Proof hint**: dynamic programming by exploiting the decomposition tree (and monotonicity)
### CSP Structures

- Variables map to nodes
- Scopes map to hyperedges

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>13</td>
<td></td>
<td>14</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>17</td>
<td></td>
<td>18</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
</tr>
</tbody>
</table>

\[ A \]

\[ \mathcal{H}_A \]
Structurally Restricted CSPs

\[ \mathcal{H}_A \]
Structurally Restricted CSPs
Structurally Restricted CSPs

The hypergraph is acyclic

- Acyclicity is efficiently recognizable
- Acyclic CSPs can be efficiently solved
- Generalized arc consistency $\rightarrow$ Global consistency
Tractability of acyclic Instances

Adapt the dynamic programming approach in (Yannakakis’81)
Adapt the dynamic programming approach in (Yannakakis’81)

With a bottom-up computation:
- Filter the tuples that do not match
Adapt the dynamic programming approach in (Yannakakis’81)

With a bottom-up computation:
- Filter the tuples that do not match
- Compute the cost of the best partial solution, by looking at the children

\[
\text{cost}(A/A1) + \text{cost}(B/B1) + \text{cost}(H/H1) + \text{cost}(C/C1) + \text{cost}(D/D1) + \text{cost}(E/E1) + \text{cost}(F/F1)
\]
Adapt the dynamic programming approach in (Yannakakis’81)

With a bottom-up computation:
- Filter the tuples that do not match
- Compute the cost of the best partial solution, by looking at the children
- Propagate the best partial solution (resolving ties arbitrarily)
Tractability of acyclic Instances (Monotone)

- Adapt the dynamic programming approach in *(Yannakakis’81)*
- Over “nearly-acyclic” instances…
Adapt the dynamic programming approach in (Yannakakis’81). Over “nearly-acyclic” instances…

Apply “acyclicization” via decomposition methods.

Bounded Hypertree Width Instances are Tractable.
Decomposition Methods
Transform the hypergraph into an acyclic one:

- Organize its edges (or nodes) in clusters
- Arrange the clusters as a tree, by satisfying the connectedness condition
Generalized Hypertree Decompositions

Transform the hypergraph into an acyclic one:
- Organize its edges (or nodes) in clusters
- Arrange the clusters as a tree, by satisfying the connectedness condition
Generalized Hypertree Decompositions

Transform the hypergraph into an acyclic one:
- Organize its edges (or nodes) in clusters
- Arrange the clusters as a tree, by satisfying the connectedness condition

Each cluster can be seen as a subproblem

{1,2,3,4,5,20,21,22,23,24,25,26} \{1H,20H\}

{1,7,11,16,20,22} \{1V,20H\}

{5,8,14,18,24,26} \{5V,20H\}

{11,12,13,17,22} \{11H,13V\}

{8,9,10,6,15,19,26} \{8H,6V\}
Transform the hypergraph into an acyclic one:
- Organize its edges (or nodes) in clusters
- Arrange the clusters as a tree, by satisfying the connectedness condition

Each cluster can be seen as a subproblem.
Transform the hypergraph into an acyclic one:

- Organize its edges (or nodes) in clusters
- Arrange the clusters as a tree, by satisfying the connectedness condition

Each cluster can be seen as a subproblem

\( H_A \)
The Winner Determination Problem is in \textbf{P} for classes having bounded hypertree-width hypergraphs (bid-vertices encoding)

The Winner Determination Problem is \textbf{NP-hard} even for acyclic instances (item-vertices encoding)
Learning from Negative Results

Simple structures are not enough:

- **Next** is NP-hard even for acyclic formulas with monotone evaluation functions [Brafman et al’10]
  - Proof exploits large values

Moreover, we show that

- **Min, Top-K, and Next** are NP-hard even for acyclic formulas, even if the domain contains three elements at most and function images contain two small values at most (-1,1)
  - Proof exploits non-monotonicity, and lists of evaluation functions (whose size is not fixed---depends on the input)
Multi-objective Optimization, through lists of preference functions

Not only acyclicity, but any (purely) structural decomposition method

Not only monotone functions, but also non-monotone ones
Smooth Functions

- $\mathcal{F}$ is smooth (w.r.t. $\Phi$ and DB) if, $\forall \theta$, the value $\mathcal{F}(\theta)$ is polynomially-bounded by the size of $\Phi$, DB, and $\mathcal{F}$
- A list $L$ of evaluation functions is smooth if it consists of a constant number of smooth evaluation functions

- Manipulate small (polynomially bounded) values
- Occur in many applications (for instance, in counting-based optimizations)
- May be non-monotonic

\begin{center}
\begin{tikzpicture}
    \node[ellipse,draw, inner sep=0.1cm] (mono) at (0,0) {monotone};
    \node[ellipse,draw, inner sep=0.1cm] (smooth) at (5,0) {smooth};
    \node[ellipse,draw, inner sep=0.1cm] (nonmono) at (3,-2) {non-monotone};
    \end{tikzpicture}
\end{center}
Example

1. Finding solutions minimizing the number of variables mapped to certain domain values
   - It is smooth and monotone

2. Finding solutions with an odd number of variables mapped to certain values (e.g. switch variables)
   - It is smooth and non-monotone

- \([2,1]\) (or vice versa) is a smooth list of evaluation functions
Results for Bounded Hypertree-width classes

<table>
<thead>
<tr>
<th>Problem</th>
<th>No Restriction</th>
<th>Monotone</th>
<th>Smooth</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN</td>
<td>NP-hard</td>
<td>in P</td>
<td>in P</td>
</tr>
<tr>
<td>NEXT</td>
<td>NP-hard</td>
<td>NP-hard</td>
<td>in P</td>
</tr>
<tr>
<td>TOP-K</td>
<td>NP-hard</td>
<td>WPD</td>
<td>WPD &amp; PS</td>
</tr>
</tbody>
</table>

- Tractability results are proved in the more general tree-projections framework (encompassing all known purely structural decomposition methods)
- For bounded-arity recursively-enumerable classes tractability results are tight (unless FPT=W[1])
Hints for Min, acyclic hyp., monotone lists

- Extend the dynamic programming approach
- Because of linearization we have a total order
- The algorithm exploits an extended list of evaluation functions (still monotone) \( [\mathcal{F}_1, \ldots, \mathcal{F}_m, \mathcal{F}_\ell] \), \( \mathcal{F}_\ell = \langle w_\ell, + \rangle \)
- where \( w_\ell(X_i, u) = |\mathcal{U}|^{n-i} \times r_u(u) \)

For each tuple, manage an additional vector with the best values for the \( m+1 \) functions.

Note that we have no ties, because of the additional function.
Use the Min algorithm as a procedure, and Lawler’s approach [Lawler’72] based on a log-time data structure HL keeping suitable pairs \((S,I')\) where \(S\) is the best solution of some modified instance \(I'\).

Initialize HL by computing and inserting in it \((\text{Min}(I), I)\), where \(I\) is the original input instance.

Repeat until we get the desired K solutions (or there are no more solutions):
- Extract from HL the pair \((S,I')\) with the best solution, and output this solution;
- Compute \(n\) modified instances of \(I'\) such that \(S\) is no longer a solution, but no further relevant solution is missed;
- Compute the Min solutions of these modified instances and put the resulting new pairs in HL.

Note that, after many iterations, HL may occupy exponential space. Yet, each iteration takes polynomial-time and each iteration outputs a new solution.
The classical dynamic programming approach does not work

- Good (partial) solutions in the subtree may lead to bad final solutions

However, smooth lists of evaluation functions have only polynomially-many possible values
For each possible vector $V$ of values, starting from the minimum one,

Check whether there exists a solution with cost $V$;
  - In $\mathbf{P}$ (use a LogSpace Alternating Turing Machine)

Once we found the optimal vector $V_o$, compute the solution with that cost
  - Use a self-reduction argument
  - Start with the least (cost) assignment for the more important variable (according to the given linearization)
Algorithm SolutionExistence (smooth)

Input: A set $Var$ of variables, a CSP instance $(\Phi, DB)$, a smooth list $L = [F_1, ..., F_k]$ of evaluation functions, where $F_i = \langle w_i, \oplus_i \rangle$ is a function over $D_i$, for each $i \in \{1, ..., k\}$, a vector $(v_1, ..., v_k) \in D_1 \times \cdots \times D_k$, and a join tree $T = (N, E)$ of the hypergraph $\mathcal{H}(\Phi)$;

Output: TRUE if, and only if, there is a solution $\theta \in \Phi^{DB}$ such that $L(\theta) = (v_1, ..., v_k)$;

begin
  let $c_1, ..., c_r$ be the children of $v$ in $T$;
  for each $i \in \{1, ..., r\}$ do
    guess a vector $(a^c_i, ..., a^c_i) \in D_1 \times \cdots \times D_k$;
    guess a substitution $\theta_{c_i} \in rel_{c_i}$;
  end for
  let $\theta'_v := \theta_v \setminus \theta_p$;
  check that all the following conditions hold
    $C1: (a_1, ..., a_k) = (F_1(\theta'_v) \oplus_1 a^c_1 \oplus_1 \cdots \oplus_1 a^c_r, ..., F_k(\theta'_v) \oplus_k a^c_1 \oplus_k \cdots \oplus_k a^c_r)$;
    $C2: \theta_v \approx \theta_{c_1}, ..., \theta_v \approx \theta_{c_r}$;
  if this check fails then return FALSE;
  for each $i \in \{1, ..., r\}$ do
    if not $findSolution(c_i, \theta_v, \theta_{c_i}, (a^c_i, ..., a^c_i))$ then return FALSE;
  end for
  return TRUE;
end

begin (* MAIN *)
  let $r$ be the root of $T$;
  guess a substitution $\theta_r \in rel_r$;
  return $findSolution(r, \emptyset, \theta_r, (v_1, ..., v_k))$;
**Next** and **Top-K**: smooth lists, acyclic instances

- **Next**: same technique as for **Min**, starting from the given input solution

- **Top-K**: just use the polynomial-time algorithm for **Next** as a procedure
Assume $\text{FPT} \neq \text{W}[1]$.

Let $\mathcal{C}$ be any recursively enumerable class of instances of bounded arity.

Then, the following are equivalent:

1. $\mathcal{C}$ has bounded (hyper)tree-width;
2. $\text{MIN}[s](\mathcal{C})$ can be solved in polynomial time and polynomial space;
3. $\text{NEXT}[s](\mathcal{C})$ can be solved in polynomial time and polynomial space;
4. $\text{MIN}[m](\mathcal{C})$ can be solved in polynomial time and output-polynomial space;
5. $\text{TOP}-K[s](\mathcal{C})$ can be solved with polynomial delay and output-polynomial space;
6. $\text{TOP}-K[m](\mathcal{C})$ can be solved with polynomial delay.
Bounded-arity tractability frontier: proof hints

To prove that unbounded treewidth classes are intractable, show that if **Next** is FPT for any of these classes, then so is the W[1]-hard p-clique problem.

Use

- The Excluded-grid Theorem
- Grohe’s reduction from p-clique for satisfaction problems
- Modify the construction by penalizing variables mapped to undesired values
- Define the constraint relations so that there is a special solution \( o \) having value 0, as well as any “good” solution (and no one else)

Computing any solution following \( o \) allows one to solve the p-clique problem.
The paper provides a picture of the computational complexity of Constraint Optimization Formulas under structural restrictions.

Positive results for both monotone and some non-monotone lists of evaluation functions.

Tight results for bounded-arity instances.

Future Work

- Look at further possible interesting classes of non-monotone functions.
- Find further positive results for unbounded-arity instances, e.g., exploiting non purely-structural notions as the submodular-width [Marx'10].
Thank you!
Tree Projections (by Example)

Δ : \( r_1(A, B, C) \quad r_2(A, F) \quad r_3(C, D) \quad r_4(D, E, F) \)
\( r_5(E, F, G) \quad r_6(G, H, I) \quad r_7(I, J) \quad r_8(J, K) \)

\( \mathcal{H}_\Delta \)

Structure of the CSP
Tree Projections (by Example)

\[ \Delta : r_1(A, B, C) \quad r_2(A, F) \quad r_3(C, D) \quad r_4(D, E, F) \\
\quad r_5(E, F, G) \quad r_6(G, H, I) \quad r_7(I, J) \quad r_8(J, K) \]

\( \mathcal{H}_\Delta \)

Structure of the CSP

\( \mathcal{H}_{\Delta v} \)

Available Views
Tree Projections (by Example)

\[ \Delta : r_1(A, B, C) \quad r_2(A, F) \quad r_3(C, D) \quad r_4(D, E, F) \quad r_5(E, F, G) \quad r_6(G, H, I) \quad r_7(I, J) \quad r_8(J, K) \]

\[ \mathcal{H}_\Delta \quad \mathcal{H}_a \quad \mathcal{H}_a^\perp \]

Structure of the CSP    Tree Projection    Available Views
Tree Projections (by Example)

\[ A \downarrow: r_1(A, B, C), r_2(A, F), r_3(C, D), r_4(D, E, F), r_5(E, F, G), r_6(G, H, I), r_7(I, J), r_8(J, K) \]

Structure of the CSP

Tree Projection

Available Views