Hybrid Backtracking Bounded by Tree-Decomposition of Constraint Networks

Emilia Oikarinen
Helsinki University of Technology
Laboratory for Theoretical Computer Science

Based on *P. Jégou and C. Terrioux, Hybrid Backtracking Bounded by Tree-Decomposition of Constraint Networks. Artificial Intelligence 146, pp. 43–75, 2003*.

Outline

- Motivation: Decomposition Methods
- Notations
- Tree-Decomposition of CSPs
- Backtracking with Tree-Decomposition (BTD)
- Extensions of BTD
- Conclusions

Motivation: Decomposition methods

- Methods for solving CSPs are usually based on backtracking combined with filtering techniques and heuristics for choosing the next variable/value.
  $\Longrightarrow$ Efficient in practice, exponential theoretical time-complexity.
- *Decomposition methods* isolate the intrinsically exponential parts to induce a second step that run in polynomial time.
  $\Longrightarrow$ Decreases time-complexity, but increases space complexity.
- Decomposition methods exploit topological properties of the *constraint graph* and are based on *tree-decomposition* of graphs
- Jégou and Terrioux combine these two approaches – *hybrid backtracking bounded by tree-decomposition*.

Notations

- *Constraint satisfaction problem (CSP)* $\mathcal{P} = (X, D, C, R)$, where
  - $X = \{x_1, , \ldots, x_n\}$ (variables),
  - $D = \{D_1, \ldots, D_n\}$ (domains),
  - $C = \{C_1, \ldots, C_m\}$ (constraints, $C_i = \{x_{i_1}, \ldots, x_{i_{j_i}}\}$) and
  - $R = \{R_1, \ldots, R_m\}$ (relations, $R_i \subseteq D_{i_1} \times \cdots \times D_{i_{j_i}}$).
- We consider only binary CSPs (constraints have arity of two).
- A *constraint graph* is $\mathcal{G} = (X, C)$.

Tree-Decomposition of CSPs

Let $G = (X, E)$ be a graph. A *tree-decomposition* of G is $(\mathcal{C}, \mathcal{T})$ with $\mathcal{T} = (I, F)$ (a tree) and $\mathcal{C} = \{\mathcal{C}_i \mid i \in I\}$ (a family of subsets of X), such that each $\mathcal{C}_i$ is a node of $\mathcal{T}$ and

- $\bigcup_{i \in I} \mathcal{C}_i = X$
- for all edges $\{x, y\} \in E$ there exists $i \in I$ such that $\{x, y\} \subset \mathcal{C}_i$,
- for all $i, j, k \in I$, if k is in a path from i to j in $\mathcal{T}$, then $\mathcal{C}_i \cap \mathcal{C}_j \subseteq \mathcal{C}_k$.

The *width* of a tree-decomposition $(\mathcal{C}, \mathcal{T})$ is $\max_{i \in I} |\mathcal{C}_i| - 1$.

The *tree-width* of G is minimal width over all tree-decompositions.

Tree-Decomposition of CSPs Cont'd

- Problem of finding a tree-decomposition is $\mathbf{NP}$-hard.
- However, for a *triangulated graph* G, the set of maximal cliques $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ of $G = (X, E)$ corresponds to the family of subsets associated with tree-decomposition.
- We can approximate tree-decomposition by triangulating G.
  - Triangulation of G is $G' = (X, E \cup E')$.
  - E' is a set of edges such that G' has no cycle of length at least 4 *without a chord*.
  - Tree-width of G is equal to the minimal width over all triangulations G'.

Tree-Clustering Method

- Triangulate the constraint graph.
- Find maximal cliques (each clique corresponds to a subproblem).
- Solve every subproblem induced by the maximal cliques.
- Solve the new acyclic n-ary CSP (can be solved in polynomial time w.r.t. the size of the induced n-ary CSP).

Practically inefficient due to space complexity.

Backtracking with Tree-Decomposition

- *Backtracking with tree-decomposition* (BTD) proceeds by an enumerative search guided by a static pre-established partial order induced by tree-decomposition of the constraint graph.
- Exploits the notion of *structural good* and *nogood*.
- Assume CSP $\mathcal{P} = (X, D, C, R)$, where (X, C) is a graph and $\mathcal{A} = (\mathcal{C}, \mathcal{T})$ its tree-decomposition (or an approximation), where $\mathcal{T} = (I, F)$ is a tree.
- Elements of $\mathcal{C} = \{\mathcal{C}_i \mid i \in I\}$ are indexed w.r.t. a *compatible numeration* $N_{\mathcal{C}}$ (a numeration compatible with a prefix numeration of $\mathcal{T} = (I, F)$ with $\mathcal{C}_1$ as the root).

BTD Cont'd

- $\mathrm{Desc}(\mathcal{C}_j) = \{x_i \in \bigcup \mathcal{C}_k \mid \mathcal{C}_k$ descendant of $\mathcal{C}_j, \mathcal{C}_j$ included$\}$
- A *compatible enumeration order* of variables of X is an order $\preceq_X$ such that $\forall x \in \mathcal{C}_i, \forall y \in \mathcal{C}_j$ with $i < j$, $x \preceq_X y$.
- **Theorem.** Let $\mathcal{C}_j$ be a son of $\mathcal{C}_i$ (thus $i < j$). There is no edge $\{x, y\}$ in the graph (X, C) such that

$$x \in (\bigcup_{k=1}^{j-1} \mathcal{C}_k) \setminus (\mathcal{C}_i \cap \mathcal{C}_j), \text{ and}$$

$$y \in \mathrm{Desc}(\mathcal{C}_j) \setminus (\mathcal{C}_i \cap \mathcal{C}_j).$$

- $\mathcal{C}_i \cap \mathcal{C}_j$ is a *separator* of the graph (X, C).

E. Oikarinen Hybrid Backtracking Bounded by Tree-Decomposition of Constraint Networks

Structural Goods and Nogoods

- Given $\mathcal{C}_i$ and one of its sons $\mathcal{C}_j$, a *good* (respectively *nogood*) of $\mathcal{C}_i$ w.r.t. $\mathcal{C}_j$, denoted $\mathrm{g}(\mathcal{C}_i/\mathcal{C}_j)$ ($\mathrm{ng}(\mathcal{C}_i/\mathcal{C}_j)$) is a consistent assignment $\mathcal{A}$ of $\mathcal{C}_i \cap \mathcal{C}_j$ such that there exists (does not exist) a consistent extension of $\mathcal{A}$ on $\mathrm{Desc}(\mathcal{C}_j)$.

E. Oikarinen Hybrid Backtracking Bounded by Tree-Decomposition of Constraint Networks

Exploiting Goods and Nogoods

- Given $\mathcal{C}_i$, one of its sons $\mathcal{C}_j$ and $Y \subset X$ such that $\mathrm{Desc}(\mathcal{C}_j) \cap Y = \mathcal{C}_i \cap \mathcal{C}_j$, the following hold
  - *Jump by the goods.* For all $\mathrm{g}(\mathcal{C}_i/\mathcal{C}_j)$ every consistent instantiation $\mathcal{A}$ of Y such that $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j] = \mathrm{g}(\mathcal{C}_i/\mathcal{C}_j)$ has a consistent extension on $\mathrm{Desc}(\mathcal{C}_j)$.
  - *Cut by the nogoods.* For all $\mathrm{ng}(\mathcal{C}_i/\mathcal{C}_j)$ there is no assignment $\mathcal{A}$ of Y such that $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j] = \mathrm{ng}(\mathcal{C}_i/\mathcal{C}_j)$ and $\mathcal{A}$ has a consistent extension on $\mathrm{Desc}(\mathcal{C}_j)$.

E. Oikarinen Hybrid Backtracking Bounded by Tree-Decomposition of Constraint Networks

The BTD Algorithm (1/2)

- Find a tree-decomposition (or an approximation) of the constraint graph (X, C).
- Explore the search space using the compatible order $\preceq_X$ beginning with the variables in $\mathcal{C}_1$.
- Inside $\mathcal{C}_i$ the enumeration works is classical way.
  - When all the variables in $\mathcal{C}_i$ have been assigned a value and all the involved constraints are satisfied, we get a consistent instantiation $\mathcal{A}$ of variables of $\mathcal{C}_1 \cup \cdots \cup \mathcal{C}_i$.
  - Search continues with the variables of a son $\mathcal{C}_j$ of $\mathcal{C}_i$.

The BTD Algorithm (2/2)

- Consider $\mathcal{C}_j$, a son of $\mathcal{C}_i$. Check if $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j]$ is a good or nogood.
  - If nogood, change the current instantiation on $\mathcal{C}_i$.
  - If good, forward-jump to continue the enumeration with the first variable after those in $\mathrm{Desc}(\mathcal{C}_j)$.
  - Otherwise, try to extend $\mathcal{A}$ in a consistent way on the variables of $\mathrm{Desc}(\mathcal{C}_j)$. If $\mathcal{A}$ can be extended, $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j]$ is recorded as a good; otherwise $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j]$ is recorded as a nogood.
- **Theorem.** BTD is sound, complete and terminates.

Extensions of BTD

- The BTD algorithm presented is based on chronological backtracking which is not efficient in practice.
- Extensions that exploit lookahead techniques like *arc-consistency* or *forward-checking* must be integrated to the BTD approach.

  FC-BTD – uses filtering used in forward-checking
  MAC-BTD – uses arc-consistency filtering
- Another way to improve backtracking is to use non-chronological backtracking like *back-jumping*.

  BTD-BJ – BTD using back-jumping
  FC-BTD-BJ, *MAC-BTD-BJ*

Conclusions

- BTD is based both on backtracking techniques and the notion of tree-decomposition of the constraint graph.
- BTD inherits the advantages of the two approaches:
  - practical efficiency of backtracking, and
  - warranty of limited space/time complexity.