

Foundations of Artificial Intelligence

28. Constraint Satisfaction Problems: Decomposition Methods

Martin Wehrle

Universität Basel

April 22, 2016

Foundations of Artificial Intelligence

April 22, 2016 — 28. Constraint Satisfaction Problems:
Decomposition Methods

28.1 Decomposition Methods

28.2 Conditioning

28.3 Tree Decomposition

28.4 Summary

Constraint Satisfaction Problems: Overview

Chapter overview: constraint satisfaction problems

- ▶ 22.–23. Introduction
- ▶ 24.–26. Basic Algorithms
- ▶ 27.–28. Problem Structure
 - ▶ 27. Constraint Graphs
 - ▶ 28. Decomposition Methods

28.1 Decomposition Methods

More Complex Graphs

What if the constraint graph is not a tree and does not decompose into several components?

- ▶ idea 1: **conditioning**
- ▶ idea 2: **tree decomposition**

German: Konditionierung, Baumzerlegung

28.2 Conditioning

Conditioning

Conditioning

idea: Apply backtracking with forward checking until the constraint graph **restricted to the remaining unassigned variables** decomposes or is a tree.

remaining problem \rightsquigarrow algorithms for simple constraint graphs

cutset conditioning:

Choose variable order such that early variables form a small **cutset** (i.e., set of variables such that removing these variables results in an acyclic constraint graph).

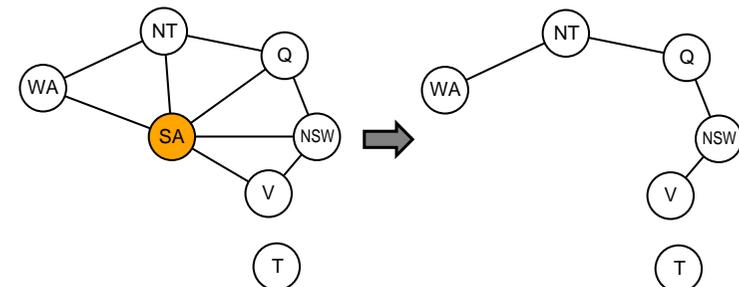
German: Cutset

time complexity: n variables, $m < n$ in cutset,
maximal domain size k : $O(k^m \cdot (n - m)k^2)$

(Finding optimal cutsets is an NP-complete problem.)

Conditioning: Example

Australia example: Cutset of size 1 suffices:



28.3 Tree Decomposition

Tree Decomposition

basic idea of **tree decomposition**:

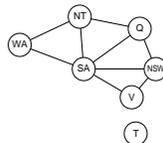
- ▶ Decompose constraint network into smaller **subproblems** (overlapping).
- ▶ Find solutions for the subproblems.
- ▶ Build overall solution based on the subsolutions.

more details:

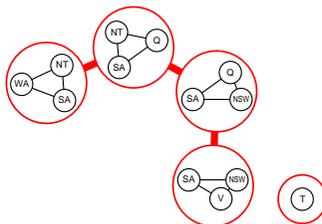
- ▶ “Overall solution building problem” based on subsolutions is a constraint network itself (**meta constraint network**).
- ▶ Choose subproblems in a way that the constraint graph of the meta constraint network is a **tree/forest**.
 ~> build overall solution with efficient tree algorithm

Tree Decomposition: Example

constraint network:



tree decomposition:



Tree Decomposition: Definition

Definition (tree decomposition)

Consider a constraint network \mathcal{C} with variables V .

A **tree decomposition** of \mathcal{C} is a graph \mathcal{T} with the following properties.

requirements on vertices:

- ▶ Every **vertex** of \mathcal{T} corresponds to a subset of the variables V . Such a vertex (and corresponding variable set) is called a **subproblem** of \mathcal{C} .
- ▶ Every **variable** of V appears in **at least one** subproblem of \mathcal{T} .
- ▶ For every **nontrivial constraint** R_{uv} of \mathcal{C} , the variables u and v appear together in **at least one** subproblem in \mathcal{T} .

...

Tree Decomposition: Definition

Definition (tree decomposition)

Consider a constraint network \mathcal{C} with variables V .

A **tree decomposition** of \mathcal{C} is a graph \mathcal{T} with the following properties.

...

requirements on edges:

- ▶ For each variable $v \in V$, let \mathcal{T}_v be the set of vertices corresponding to the subproblems that contain v .
- ▶ For each variable v , the set \mathcal{T}_v is **connected**, i.e., each vertex in \mathcal{T}_v is reachable from every other vertex in \mathcal{T}_v without visiting vertices not contained in \mathcal{T}_v .
- ▶ \mathcal{T} is **acyclic** (a tree/forest)

Meta Constraint Network

meta constraint network $\mathcal{C}^{\mathcal{T}} = \langle V^{\mathcal{T}}, \text{dom}^{\mathcal{T}}, (R_{uv}^{\mathcal{T}}) \rangle$

based on tree decomposition \mathcal{T}

- ▶ $V^{\mathcal{T}} :=$ vertices of \mathcal{T} (i.e., subproblems of \mathcal{C} occurring in \mathcal{T})
- ▶ $\text{dom}^{\mathcal{T}}(v) :=$ set of solutions of subproblem v
- ▶ $R_{uv}^{\mathcal{T}} := \{ \langle s, t \rangle \mid s, t \text{ compatible solutions of subproblems } u, v \}$

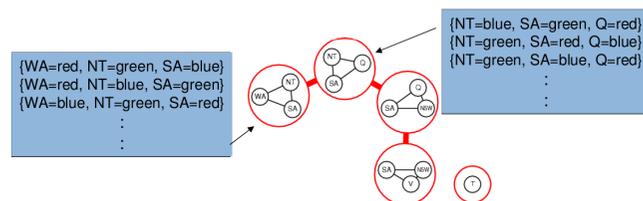
German: Meta-Constraintnetz

Solutions of two subproblems are called **compatible** if all overlapping variables are assigned identically.

Solving with Tree Decompositions: Algorithm

algorithm:

- ▶ Find **all solutions** for **all subproblems** in the decomposition and build a tree-like **meta constraint network**.
- ▶ Constraints in meta constraint network: subsolutions must be **compatible**.
- ▶ Solve meta constraint network with an algorithm for tree-like networks.



Good Tree Decompositions

goal: each subproblem has as few variables as possible

- ▶ crucial: subproblem V' in \mathcal{T} with highest number of variables
- ▶ number of variables in V' minus 1 is called **width** of the decomposition
- ▶ best width over all decompositions: **tree width** of the constraint graph (computation is NP-complete)

time complexity of solving algorithm based on tree decompositions:

$O(nk^{w+1})$, where w is width of decomposition

(requires specialized version of revise; otherwise $O(nk^{2w+2})$.)

28.4 Summary

Summary: This Chapter

- ▶ Reduce **complex** constraint graphs to **simple** constraint graphs.
- ▶ **cutset conditioning**:
 - ▶ Choose **as few** variables as possible (cutset) such that an assignment to these variables yields a **remaining problem** which is structurally simple.
 - ▶ **search** over assignments of variables in cutset
- ▶ **tree decomposition**: build **tree-like** meta constraint network
 - ▶ meta variables: **groups** of original variables that jointly cover all variables and constraints
 - ▶ **values** correspond to consistent assignments to the groups
 - ▶ constraints between **overlapping** groups to ensure **compatibility**
 - ▶ overall algorithm exponential in **width** of decomposition (size of largest group)

Summary: CSPs

Constraint Satisfaction Problems (CSP)

General formalism for problems where

- ▶ values have to be assigned to variables
- ▶ such that the given constraints are satisfied.
- ▶ algorithms: **backtracking search + inference** (e.g., forward checking, arc consistency, path consistency)
- ▶ variable and value orders important
- ▶ more efficient: exploit **structure of constraint graph** (connected components; trees)

More Advanced Topics

more advanced topics (not considered in this course):

- ▶ **backjumping**: backtracking over several layers
- ▶ **no-good learning**: infer additional constraints based on information collected during backtracking
- ▶ **local search methods** in the space of total, but not necessarily consistent assignments
- ▶ **tractable constraint classes**: identification of constraint types that allow for polynomial algorithms
- ▶ solutions of different quality: **constraint optimization problems (COP)**

↔ more than enough content for a one-semester course