Conflict Graphs for Combinatorial Optimization Problems

Ulrich Pferschy

joint work with Andreas Darmann and Joachim Schauer

University of Graz, Austria

Introduction

Combinatorial Optimization Problem CO

Consider any CO with decision variables $x_j \in \{0, 1\}$, $j \in V$, and a feasible domain $x \in S$.

Introduction

Combinatorial Optimization Problem CO

Consider any CO with decision variables $x_j \in \{0,1\}$, $j \in V$, and a feasible domain $x \in S$.

Conflict Structure

Add disjunctive constraints for some pairs of variables:

$$x_i + x_j \leq 1 \quad \text{for } (i,j) \in E \subset V \times V$$

$\Longrightarrow$ at most one of the two variables i, j can be set to 1.

Representation by a **conflict graph** $G = (V, E)$

Edges in G connect conflicting variables of CO.

Introduction

Relation to Independent Set (IS)

Feasible domain for CO problem with a conflict graph:
Intersection of S with an independent / stable set problem in G

Introduction

Relation to Independent Set (IS)

Feasible domain for CO problem with a conflict graph:
Intersection of S with an independent / stable set problem in G

Complexity

IS is already strongly $\mathcal{NP}$-hard, no constant approximation ratio
$\Longrightarrow$ adding an IS condition makes CO (much) more difficult.

Introduction

Relation to Independent Set (IS)

Feasible domain for CO problem with a conflict graph:
Intersection of S with an independent / stable set problem in G

Complexity

IS is already strongly $\mathcal{NP}$-hard, no constant approximation ratio
$\Longrightarrow$ adding an IS condition makes CO (much) more difficult.

One main direction of research:
Identify special graph classes for the conflict graph G such that the considered CO problem

- is polynomially solvable
- permits a (fully) polynomial approximation scheme
- has a constant approximation ratio

Conflict Graphs: Bin Packing

Conflicting items must be in different bins

Conflict Graphs: Bin Packing

Conflicting items must be in different bins

- Jansen, Öhring '97: 5/2- resp. $2+\varepsilon$-approximation for special graph classes; improved by Epstein, Levin '06 (also on-line)
- Epstein et al.'08: extension to two dimensional packing of squares
- Jansen '99: A-FPTAS for special graph classes

e.g.: perfect, bipartite, interval, *d*-inductive graphs.

Conflict Graphs: Bin Packing

Conflicting items must be in different bins

- Jansen, Öhring '97: 5/2- resp. $2+\varepsilon$-approximation for special graph classes; improved by Epstein, Levin '06 (also on-line)
- Epstein et al.'08: extension to two dimensional packing of squares
- Jansen '99: A-FPTAS for special graph classes

e.g.: perfect, bipartite, interval, *d*-inductive graphs.

- Gendreau et al.'04: heuristics and lower bounds
- Malaguti et al.'07: hybrid tabu search
- Malaguti et al.'08: exact algorithm (branch-and-price)

Conflict Graphs: Scheduling

Mutual Exclusion Scheduling

Schedule unit-length jobs on m machines, conflicting jobs not to be executed in the same time interval.

Baker, Coffman '96; Bodlaender, Jansen '93;
polynomially solvable special cases, special graph classes.

Conflict Graphs: Scheduling

Mutual Exclusion Scheduling

Schedule unit-length jobs on m machines, conflicting jobs not to be executed in the same time interval.

Baker, Coffman '96; Bodlaender, Jansen '93;
polynomially solvable special cases, special graph classes.

Scheduling with Incompatible Jobs

Conflicting jobs not to be executed on the same machine.

Bodlaender, Jansen, Woeginger '94:
approximation algorithms for special graph classes.

Knapsack Problem with Conflict Graph (KCG)

Vertices (=items) adjacent in G cannot be packed together in the knapsack!

ILP-Formulation

$$
\begin{array}{lll}
\text{(KCG)} & \max & \sum_{j=1}^{n} p_j x_j \\
 & \text{s.t.} & \sum_{j=1}^{n} w_j x_j \leq c \\
 & & (i,j) \in E \implies x_i + x_j \leq 1 \\
 & & x_j \in \{0,1\}
\end{array}
$$

Knapsack Problem with Conflict Graph (KCG)

Vertices (=items) adjacent in G cannot be packed together in the knapsack!

ILP-Formulation

$$
\begin{aligned}
\text{(KCG)} \quad \max \quad & \textstyle\sum_{j=1}^{n} p_j x_j \\
\text{s.t.} \quad & \textstyle\sum_{j=1}^{n} w_j x_j \leq c \\
& (i,j) \in E \Longrightarrow x_i + x_j \leq 1 \\
& x_j \in \{0,1\}
\end{aligned}
$$

Introduce upper bound P on optimal solution, e.g. $P := \sum_{j=1}^{n} p_j$

Note: Classical Greedy algorithm can perform as bad as possible!

Literature on KCG

Exact Algorithms and Heuristics

- Yamada et al. '02: introduce the problem, present heuristic and exact algorithms (based on Lagrangean relaxation)

Literature on KCG

Exact Algorithms and Heuristics

- Yamada et al. '02: introduce the problem, present heuristic and exact algorithms (based on Lagrangean relaxation)
- Hifi, Michrafy '06: reactive local search algorithm
- Hifi, Michrafy '07: exact algorithms (refined branch-and-bound strategy)

Literature on KCG

Exact Algorithms and Heuristics

- Yamada et al. ’02: introduce the problem, present heuristic and exact algorithms (based on Lagrangean relaxation)
- Hifi, Michrafy ’06: reactive local search algorithm
- Hifi, Michrafy ’07: exact algorithms (refined branch-and-bound strategy)

no special graph classes considered!

Knapsack Problem with Conflict Graph (KCG)

Our Goal

Identify special graph classes, where KCG can be solved in pseudo-polynomial time and permits an FPTAS.

Knapsack Problem with Conflict Graph (KCG)

Our Goal

Identify special graph classes, where KCG can be solved in pseudo-polynomial time and permits an FPTAS.

Our Results

Pseudo-polynomial time algorithms and FPTAS for KCG on:

- Trees

Knapsack Problem with Conflict Graph (KCG)

Our Goal

Identify special graph classes, where KCG can be solved in pseudo-polynomial time and permits an FPTAS.

Our Results

Pseudo-polynomial time algorithms and FPTAS for KCG on:

- Trees
- Graphs with Bounded Treewidth

Knapsack Problem with Conflict Graph (KCG)

Our Goal

Identify special graph classes, where KCG can be solved in pseudo-polynomial time and permits an FPTAS.

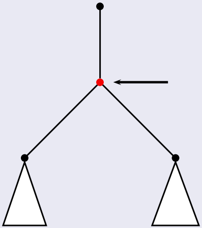
Our Results

Pseudo-polynomial time algorithms and FPTAS for KCG on:

- Trees
- Graphs with Bounded Treewidth
- Chordal Graphs

Idea for KCG on Trees

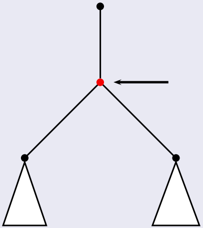
Basic Observation



Apply Dynamic Programming by Profits
$\Longrightarrow$ Scaling yields FPTAS
We use Dynamic Programming by Reaching moving bottom up in the conflict tree.

Idea for KCG on Trees

Basic Observation



Apply Dynamic Programming by Profits

$\Longrightarrow$ Scaling yields FPTAS

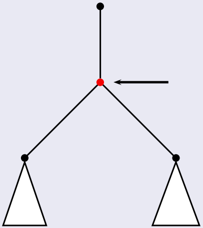
We use Dynamic Programming by Reaching moving bottom up in the conflict tree.

For every vertex i:

Determine the solution of the subproblem defined by the subtree rooted in i.

Idea for KCG on Trees

Basic Observation



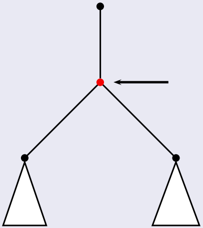
Apply Dynamic Programming by Profits
$\Longrightarrow$ Scaling yields FPTAS
We use Dynamic Programming by Reaching moving bottom up in the conflict tree.
For every vertex i:
Determine the solution of the subproblem defined by the subtree rooted in i.

Notation

$z_i(d)$ solution with profit d and minimal weight found in the subtree $T(i)$ with item i necessarily **included**.

Idea for KCG on Trees

Basic Observation



Apply Dynamic Programming by Profits
$\Longrightarrow$ Scaling yields FPTAS
We use Dynamic Programming by Reaching moving bottom up in the conflict tree.
For every vertex i:
Determine the solution of the subproblem defined by the subtree rooted in i.

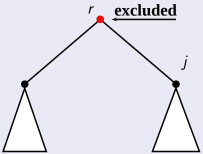
Notation

$z_i(d)$ solution with profit d and minimal weight found in the subtree $T(i)$ with item i necessarily **included**.
$y_i(d)$ solution with profit d and minimal weight found in the subtree $T(i)$ with item i **excluded**.

Algorithmic Details

Tree traversed in Depth-First-Search-Order (DFS)

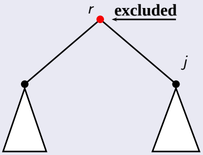
Vertex r excluded



Algorithmic Details

Tree traversed in Depth-First-Search-Order (DFS)

Vertex r excluded



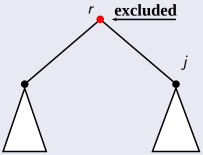
for every child j of r:

$$y_r(d) = \min_k \{y_r(d-k) + \min\{z_j(k), y_j(k)\}\}$$

Algorithmic Details

Tree traversed in Depth-First-Search-Order (DFS)

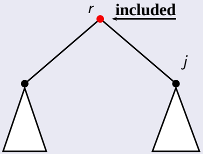
Vertex r excluded



for every child j of r:

$$y_r(d) = \min_k \{y_r(d-k) + \min\{z_j(k), y_j(k)\}\}$$

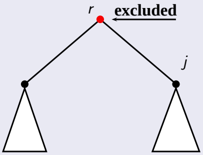
Vertex r included



Algorithmic Details

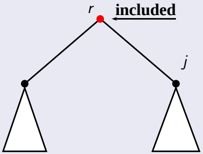
Tree traversed in Depth-First-Search-Order (DFS)

Vertex r excluded



for every child j of r:
$y_r(d) = \min_k \{y_r(d-k) + \min\{z_j(k), y_j(k)\}\}$

Vertex r included

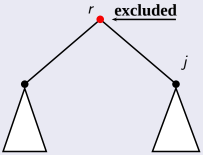


for every child j of r:
$z_r(d) = \min_k \{z_r(d-k) + y_j(k)\}$

Algorithmic Details

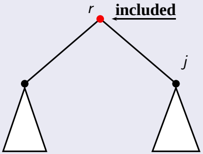
Tree traversed in Depth-First-Search-Order (DFS)

Vertex r excluded



for every child j of r:
$y_r(d) = \min_k \{y_r(d-k) + \min\{z_j(k), y_j(k)\}\}$

Vertex r included

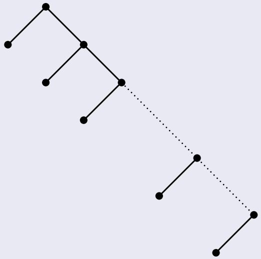


for every child j of r:
$z_r(d) = \min_k \{z_r(d-k) + y_j(k)\}$

Running time: $O(nP^2)$ Space: $O(nP)$ (trivial version)

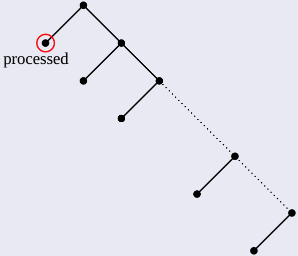
Space Reduction Method

Worst case without reduction: $O(nP)$



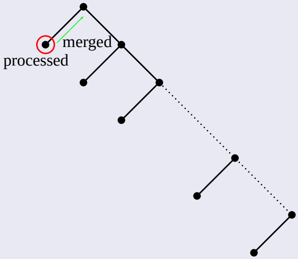
Space Reduction Method

Worst case without reduction: $O(nP)$



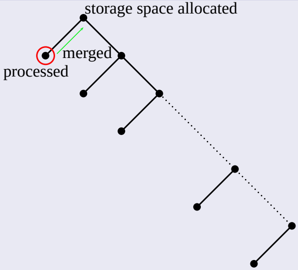
Space Reduction Method

Worst case without reduction: $O(nP)$



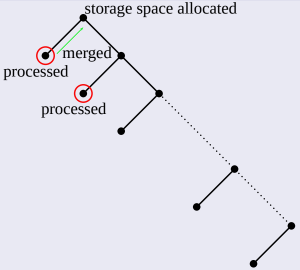
Space Reduction Method

Worst case without reduction: $O(nP)$



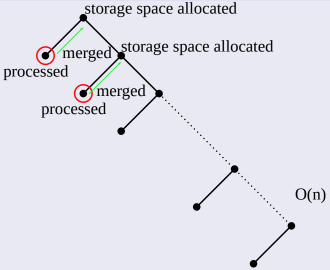
Space Reduction Method

Worst case without reduction: $O(nP)$



Space Reduction Method

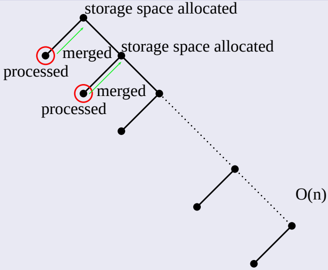
Worst case without reduction: $O(nP)$



Choosing the left child in every step allocates $O(n)$ storage arrays.

Space Reduction Method

Worst case without reduction: $O(nP)$

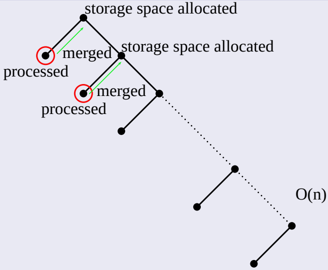


Choosing the left child in every step allocates $O(n)$ storage arrays.

By choosing the right child vertex, only two arrays would be necessary.

Space Reduction Method

Worst case without reduction: $O(nP)$



Choosing the left child in every step allocates $O(n)$ storage arrays.

By choosing the right child vertex, only two arrays would be necessary.

In general: Space can be reduced to $O(\log n)$ arrays.

Worst-case: Complete binary tree

Space Reduction Method (general)

Property 1. Tree T processed in DFS order with following rule: Take child vertex j, whose subtree contains the largest number of vertices.

Space Reduction Method (general)

Property 1. Tree T processed in DFS order with following rule: Take child vertex j, whose subtree contains the largest number of vertices.

Property 2. Each vertex of T requires $O(k)$ space for processing.

Space Reduction Method (general)

Property 1. Tree T processed in DFS order with following rule: Take child vertex j, whose subtree contains the largest number of vertices.

Property 2. Each vertex of T requires $O(k)$ space for processing.

Property 3. Merging child j to parent i requires $2 * O(k)$ space.

Space Reduction Method (general)

Property 1. Tree T processed in DFS order with following rule: Take child vertex j, whose subtree contains the largest number of vertices.

Property 2. Each vertex of T requires $O(k)$ space for processing.

Property 3. Merging child j to parent i requires $2 * O(k)$ space.

Lemma 1. An algorithm A fulfilling Properties 1, 2 and 3 uses at most $(\text{ld}(n) + 1) * O(k)$ space.

Space Reduction Method (general)

Property 1. Tree T processed in DFS order with following rule: Take child vertex j, whose subtree contains the largest number of vertices.

Property 2. Each vertex of T requires $O(k)$ space for processing.

Property 3. Merging child j to parent i requires $2 * O(k)$ space.

Lemma 1. An algorithm A fulfilling Properties 1, 2 and 3 uses at most $(\mathrm{ld}(n) + 1) * O(k)$ space.

Sketch of Proof. r has k childs $i_1 \dots i_k$ so that $|T(i_j)| \leq \frac{n}{2}$ and w.l.o.g $|T(i_1)| \geq |T(i_j)|$ for all $j \in \{1 \dots k\}$:
Then the processing of $T(i_1)$ is done by using at most $(\mathrm{ld}(\frac{n}{2}) + 1) * O(k) = \mathrm{ld}(n) * O(k)$ space. After merging this subtree to r this space can be deallocated, but $O(k)$ space is used at vertex r, which has to be kept until A has finished.

Bounded Treewidth

Tree-Decomposition of graph $G = (V, E)$

Every graph can be represented by a tree whose vertices are subsets of V.
Original graph can be reproduced from the tree-decomposition.
E.g. adjacent vertices of G must be jointly contained in at least one subset.

Treewidth: minimal cardinality over all tree-decompositions of the largest subset-1.

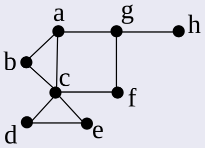
Bounded Treewidth

Tree-Decomposition of graph $G = (V, E)$

Every graph can be represented by a tree whose vertices are subsets of V.
Original graph can be reproduced from the tree-decomposition.
E.g. adjacent vertices of G must be jointly contained in at least one subset.

Treewidth: minimal cardinality over all tree-decompositions of the largest subset-1.

Example



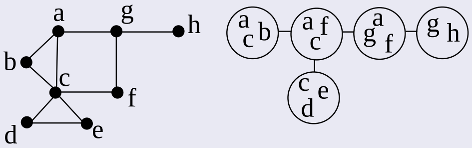
Bounded Treewidth

Tree-Decomposition of graph $G = (V, E)$

Every graph can be represented by a tree whose vertices are subsets of V.
Original graph can be reproduced from the tree-decomposition.
E.g. adjacent vertices of G must be jointly contained in at least one subset.

Treewidth: minimal cardinality over all tree-decompositions of the largest subset-1.

Example



Bounded Treewidth

Treewidth

Treewidth indicates "how far is the graph away from a tree".

- trees have treewidth 1
- series parallel graphs have treewidth 2
- ...

Bounded Treewidth

Treewidth

Treewidth indicates "how far is the graph away from a tree".

- trees have treewidth 1
- series parallel graphs have treewidth 2
- ...

Nice Tree-Decomposition

For algorithmic purposes, the structure of the decomposition is restricted to four simple configurations.

A nice tree-decomposition with the same treewidth can be computed from a tree-decomposition in $O(n)$ time.
[cf. Bodlaender, Koster '08]

Many $\mathcal{NP}$-hard problems can be solved efficiently for graphs with bounded treewidth. [Bodlaender '97]

Solving KCG with Bounded Treewidth

Algorithmic Idea

Take a nice tree-decomposition T of G
Process T in a DFS way.

For every vertex of T (i.e. a subset of V):
Consider all independent sets (IS) of all vertices in T explicitly.
Note: Number of IS is constant!

Solving KCG with Bounded Treewidth

Algorithmic Idea

Take a nice tree-decomposition T of G
Process T in a DFS way.

For every vertex of T (i.e. a subset of V):
Consider all independent sets (IS) of all vertices in T explicitly.
Note: Number of IS is constant!

Perform dynamic programming and consider inclusion or exclusion for every IS.

Solving KCG with Bounded Treewidth

Algorithmic Idea

Take a nice tree-decomposition T of G
Process T in a DFS way.

For every vertex of T (i.e. a subset of V):
Consider all independent sets (IS) of all vertices in T explicitly.
Note: Number of IS is constant!

Perform dynamic programming and consider inclusion or exclusion for every IS.

Time and Space

KCG for conflict graphs of bounded treewidth can be solved in $O(nP^2)$ time and $O(\log n \; P + n)$ space given a tree-decomposition.

The space reduction method of Lemma 1 can be applied again!

Chordal Graphs

Definition

A Chordal Graph (a.k.a. triangulated graph) does not contain induced cycles other than triangles.

$\implies$ every cycle of at least four vertices has a **chord**.

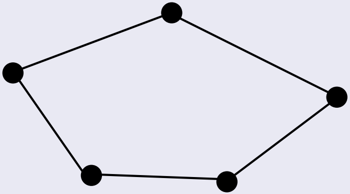
Chordal Graphs

Definition

A Chordal Graph (a.k.a. triangulated graph) does not contain induced cycles other than triangles.

$\Longrightarrow$ every cycle of at least four vertices has a **chord**.

Example



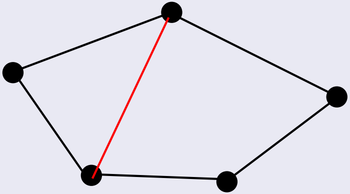
Chordal Graphs

Definition

A Chordal Graph (a.k.a. triangulated graph) does not contain induced cycles other than triangles.

$\Longrightarrow$ every cycle of at least four vertices has a **chord**.

Example



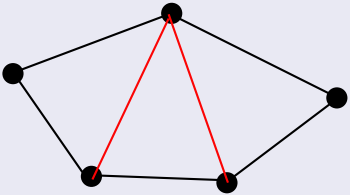
Chordal Graphs

Definition

A Chordal Graph (a.k.a. triangulated graph) does not contain induced cycles other than triangles.

$\Longrightarrow$ every cycle of at least four vertices has a **chord**.

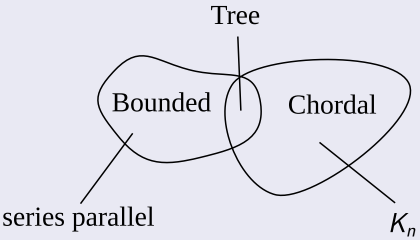
Example



Chordal Graphs

Bounded Treewidth versus Chordal Graph

No subset relation!



Solving KCG on Chordal Graphs

Tree Representation

For every chordal graph G there is a clique tree $T = (\mathcal{K}, \mathcal{E})$:

maximal cliques K of G are vertices of T

for each vertex $v \in G$:

all cliques K containing v induce a subtree in T.

cf. [Blair, Peyton '93]

Solving KCG on Chordal Graphs

Tree Representation

For every chordal graph G there is a clique tree $T = (\mathcal{K}, \mathcal{E})$:

maximal cliques K of G are vertices of T

for each vertex $v \in G$:

all cliques K containing v induce a subtree in T.

cf. [Blair, Peyton '93]

Basic Idea of the Algorithm

The vertices of the clique tree T are complete subgraphs.

$\Longrightarrow$ at most one vertex of each clique can be in the knapsack.

Process T in a DFS way.

Solving KCG on Chordal Graphs

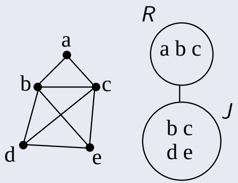
Dynamic Programming Definition

$f_d^v(I)$:
solution with profit d and minimal weight containing item $v \in I$, while considering in the clique tree only the subtree rooted in I.

definition extended to $v = \emptyset$.

Solving KCG on Chordal Graphs

Vertex R with one (or first) child J



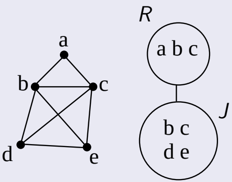
Solving KCG on Chordal Graphs

Vertex R with one (or first) child J

for $v \in R$:

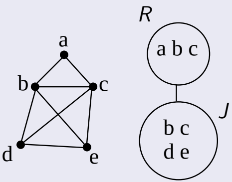
if $v \in R \cap J$: $\{b, c\}$

$$f_d^v(R) = f_d^v(J)$$



Solving KCG on Chordal Graphs

Vertex R with one (or first) child J



for $v \in R$:

if $v \in R \cap J$: $\{b, c\}$

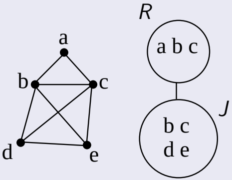
$$f_d^v(R) = f_d^v(J)$$

else: $\{a\}$

$$f_d^v(R) = w(v) + \min_i \left\{ f^i_{d-p(v)}(J) : i \in (J \setminus R) \cup \emptyset \right\}$$

Solving KCG on Chordal Graphs

Vertex R with one (or first) child J



for $v \in R$:

if $v \in R \cap J$: $\{b, c\}$

$$f_d^v(R) = f_d^v(J)$$

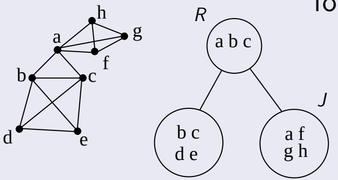
else: $\{a\}$

$$f_d^v(R) = w(v) + \min_i \left\{ f^i_{d-p(v)}(J) \, : \, i \in (J \setminus R) \cup \emptyset \right\}$$

$$f_d^{\emptyset}(R) = \min_i \left\{ f_d^i(J) \, : \, i \in (J \setminus R) \cup \emptyset \right\}$$

Solving KCG on Chordal Graphs

Vertex R with second child J



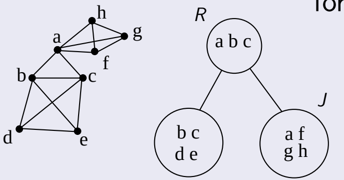
for $v \in R$:

if $v \in R \cap J$: $\{a\}$

$$f_d^v(R) =$$
$$\min_k \left\{ f_k^v(R) + f_{d-k+p(v)}^v(J) \right\}$$
$$f_d^v(R) = f_d^v(R) - w(v)$$

Solving KCG on Chordal Graphs

Vertex R with second child J



for $v \in R$:

if $v \in R \cap J$: $\{a\}$

$f_d^v(R) =$

$\min_k \left\{ f_k^v(R) + f_{d-k+p(v)}^v(J) \right\}$

$f_d^v(R) = f_d^v(R) - w(v)$

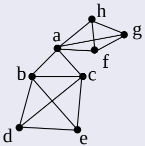
else: $\{b, c\}$

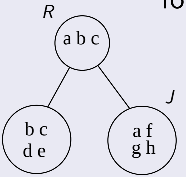
$f_d^v(R) =$

$\min_{i,k} \left\{ f_k^v(R) + f_{d-k}^i(J) : i \in (J \setminus R) \cup \emptyset \right\}$

Solving KCG on Chordal Graphs

Vertex R with second child J





for $v \in R$:

if $v \in R \cap J$: $\{a\}$

$$f_d^v(R) = \min_k \left\{ f_k^v(R) + f_{d-k+p(v)}^v(J) \right\}$$

$$f_d^v(R) = f_d^v(R) - w(v)$$

else: $\{b, c\}$

$$f_d^v(R) = \min_{i,k} \left\{ f_k^v(R) + f_{d-k}^i(J) : i \in (J \setminus R) \cup \emptyset \right\}$$

$$f_d^{\emptyset}(R) = \min_{i,k} \left\{ f_k^{\emptyset}(R) + f_{d-k}^i(J) : i \in (J \setminus R) \cup \emptyset \right\}$$

Solving KCG on Chordal Graphs

Running Time

Straightforward: $O(n^4 P^2)$

Solving KCG on Chordal Graphs

Running Time

Straightforward: $O(n^4 P^2)$

Taking a closer look: $O(n^2 P^2)$

Solving KCG on Chordal Graphs

Running Time

Straightforward: $O(n^4 P^2)$

Taking a closer look: $O(n^2 P^2)$

Space

$O(n \log n \; P)$ with space reduction technique!

Solving KCG on Chordal Graphs

Running Time

Straightforward: $O(n^4 P^2)$

Taking a closer look: $O(n^2 P^2)$

Space

$O(n \log n\ P)$ with space reduction technique!

Storing Solution Sets (for all three algorithms)

Note: Storing not only solution values but solution sets increases time and space by a factor of n (or $\log n$ for bit-encoding).

Solving KCG on Chordal Graphs

Running Time

Straightforward: $O(n^4 P^2)$

Taking a closer look: $O(n^2 P^2)$

Space

$O(n \log n\ P)$ with space reduction technique!

Storing Solution Sets (for all three algorithms)

Note: Storing not only solution values but solution sets increases time and space by a factor of n (or $\log n$ for bit-encoding).

This can be avoided by applying a general recursive divide and conquer technique (see Pferschy '99).

Deriving an FPTAS from Dynamic Programming

Scaling

Classical approach of an FPTAS for the knapsack problem:

Scale profits:

$$\tilde{p}_j = \left(\frac{n}{\varepsilon p_{\max}}\right) p_j$$

Running time:

$$n \cdot P \quad \xrightarrow{\text{scaling}} n \cdot \tilde{P} \leq n \cdot n \, \tilde{p}_{\max} = n^2 \left(\frac{n}{\varepsilon p_{\max}}\right) p_{\max} = \frac{n^3}{\varepsilon}$$

Induced relative error can be bounded by ε.

Deriving an FPTAS from Dynamic Programming

Scaling

Classical approach of an FPTAS for the knapsack problem:
Scale profits:

$$\tilde{p}_j = \left(\frac{n}{\varepsilon p_{\max}}\right) p_j$$

Running time:

$$n \cdot P \quad \xrightarrow{\text{scaling}} n \cdot \tilde{P} \leq n \cdot n\, \tilde{p}_{\max} = n^2 \left(\frac{n}{\varepsilon p_{\max}}\right) p_{\max} = \frac{n^3}{\varepsilon}$$

Induced relative error can be bounded by ε.

FPTAS for KCG

Same scaling approach can be applied to all three KCG algorithms.
Technical details rather straightforward.

Minimum Spanning Tree with Conflict Graph (MSTCG)

Problem Description

given:

a weighted graph $H = (V, E, w)$,

a conflict graph $G = (E, \bar{E})$ whose m vertices correspond uniquely to edges in E.

edge $\bar{e} = (i, j) \in \bar{E} \implies$ conflict between the two edges $i, j \in E$

Minimum Spanning Tree with Conflict Graph (MSTCG)

Problem Description

given:

a weighted graph $H = (V, E, w)$,
a conflict graph $G = (E, \bar{E})$ whose m vertices correspond uniquely to edges in E.

edge $\bar{e} = (i, j) \in \bar{E} \Longrightarrow$ conflict between the two edges $i, j \in E$

Problem (MSTCG):

find a minimum spanning tree $T \subseteq E$ of H
without conflicts w.r.t. G

Minimum Spanning Tree with Conflict Graph (MSTCG)

Question:
For which type of conflict graph does MSTCG become $\mathcal{NP}$-hard?

Minimum Spanning Tree with Conflict Graph (MSTCG)

Question:

For which type of conflict graph does MSTCG become $\mathcal{NP}$-hard?

Definition

2-**ladder**: graph whose components are paths of length one.

3-**ladder**: graph whose components are paths of length two.

Minimum Spanning Tree with Conflict Graph (MSTCG)

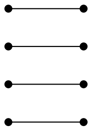
Question:
For which type of conflict graph does MSTCG become $\mathcal{NP}$-hard?

Definition

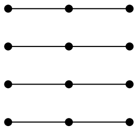
2-**ladder**: graph whose components are paths of length one.

3-**ladder**: graph whose components are paths of length two.

2-ladder



3-ladder



(imagine a rope ladder)

Minimum Spanning Tree with Conflict Graph (MSTCG)

Question:
For which type of conflict graph does MSTCG become $\mathcal{NP}$-hard?

Definition

2-**ladder**: graph whose components are paths of length one.

3-**ladder**: graph whose components are paths of length two.

Our Complexity Results

- MSTCG is polynomially solvable if the conflict graph G is a 2-ladder.

Minimum Spanning Tree with Conflict Graph (MSTCG)

Question:
For which type of conflict graph does MSTCG become $\mathcal{NP}$-hard?

Definition

2-**ladder**: graph whose components are paths of length one.

3-**ladder**: graph whose components are paths of length two.

Our Complexity Results

- MSTCG is polynomially solvable if the conflict graph G is a 2-ladder.
- MSTCG is strongly $\mathcal{NP}$-hard if the conflict graph G is a 3-ladder.

MSTCG with a 2-ladder

Conflict-free Matroid

Subsets of edges NOT containing any conflicting pair define the **conflict-free matroid**:

$$\mathcal{I} := \left\{ E' \subseteq E \mid \nexists (e,f) \in \bar{E} : \{e,f\} \subseteq E' \right\}$$

Note that in a 2-ladder conflicting pairs are independent from each other.

MSTCG with a 2-ladder

Conflict-free Matroid

Subsets of edges NOT containing any conflicting pair define the **conflict-free matroid**:

$$\mathcal{I} := \left\{ E' \subseteq E \mid \nexists (e, f) \in \bar{E} : \{e, f\} \subseteq E' \right\}$$

Note that in a 2-ladder conflicting pairs are independent from each other.

$\Longrightarrow$ MSTCG is the intersection of the *graphic matroid* (MST) with the *conflict-free matroid*

Matroid Intersection

MSTCG with a 2-ladder can be solved by Edmonds' weighted matroid intersection algorithm in polynomial time.

MSTCG with a 3-ladder

Complexity Result

MSTCG with a 3-ladder is strongly $\mathcal{NP}$-hard.

MSTCG with a 3-ladder

Complexity Result

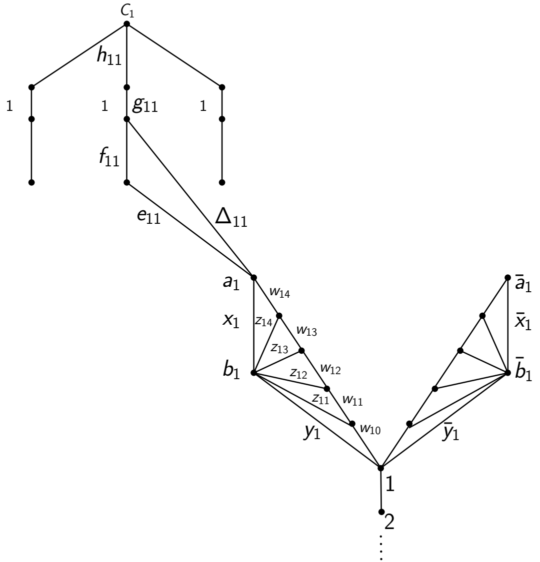
MSTCG with a 3-ladder is strongly $\mathcal{NP}$-hard.

Construction

Reduction from the special case of 3-SAT, where each variable occurs in at most 5 clauses (still $\mathcal{NP}$-complete).

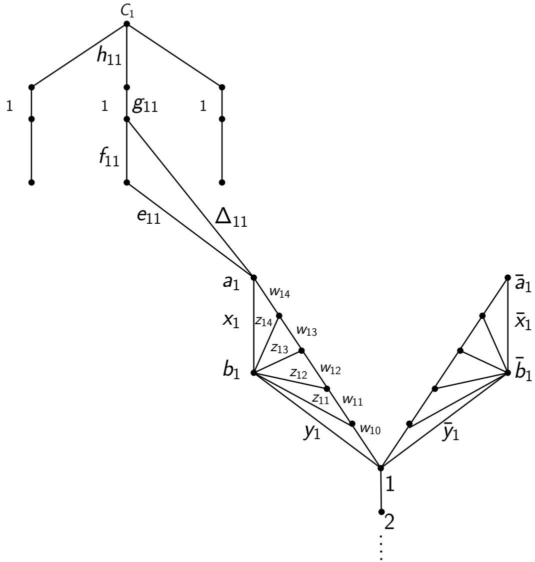
Construct an instance of MST and a 3-ladder conflict graph as follows:

MSTCG with a 3-ladder



For every clause, e.g. C_1, a *fork* represents the three literals.

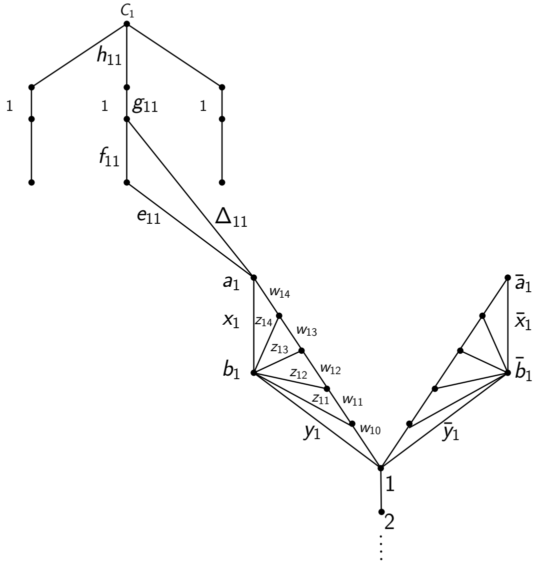
MSTCG with a 3-ladder



For every clause, e.g. C_1, a *fork* represents the three literals.

For every variable, two *gadgets* represent x_1 and $\bar{x}_1$.
Edges w_{1j}, z_{1j} for the ≤ 5 clauses variable 1 appears in.

MSTCG with a 3-ladder



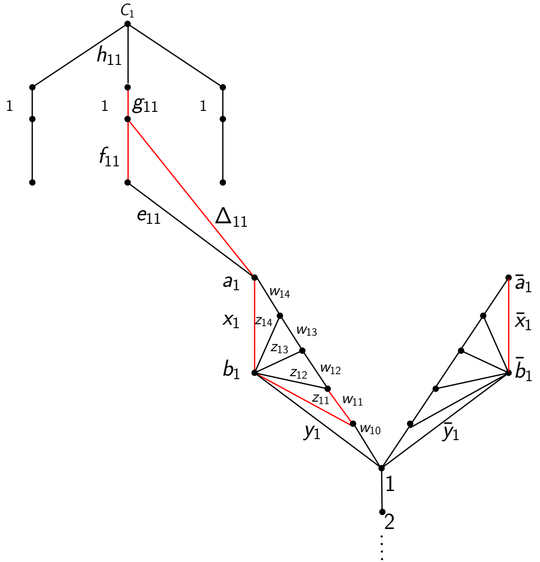
For every clause, e.g. C_1, a *fork* represents the three literals.

For every variable, two *gadgets* represent x_1 and $\bar{x}_1$.
Edges w_{1j}, z_{1j} for the ≤ 5 clauses variable 1 appears in.

Spanning tree:
Connection from C_1 to 1 either via x_1 or via $w_{10}, \ldots, w_{14}$.

edge x_1 in the tree $\Leftrightarrow$ $x_1 = \text{TRUE}$

MSTCG with a 3-ladder



Conflict Graph G:

$(x_1, \bar{x}_1)$

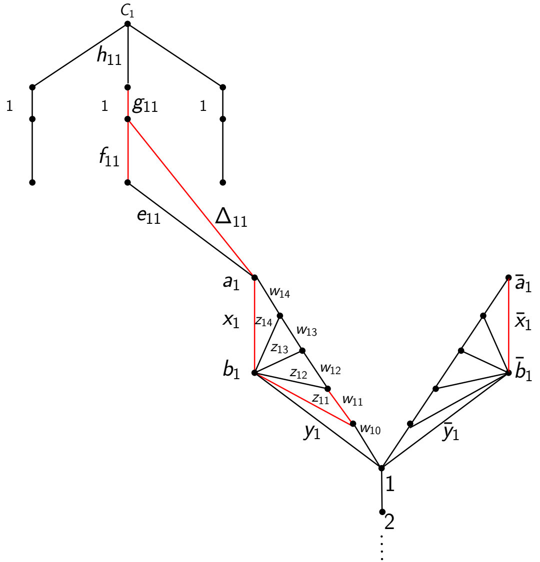
(Δ_{11}, g_{11})

(z_{11}, w_{11}, f_{11})

(z_{1j}, w_{1j}, f_{1j}), $j = 2, 3, 4$

(w_{10}, f_{10})

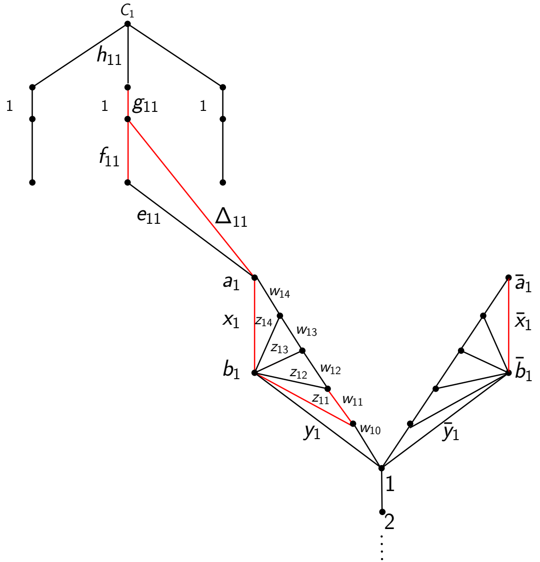
MSTCG with a 3-ladder



Conflict Graph G:
$(x_1, \bar{x}_1)$
(Δ_{11}, g_{11})
(z_{11}, w_{11}, f_{11})
(z_{1j}, w_{1j}, f_{1j}), $j = 2, 3, 4$
(w_{10}, f_{10})

Connect C_1 to 1 in MST:
choose $g_{11} \longrightarrow f_{11} \Rightarrow$
w_{11} forbidden
other w_{1j}: either forbidden or chosen $\Rightarrow$
z_{1j} forbidden $\Longrightarrow$
only connection via x_1

MSTCG with a 3-ladder



Conflict Graph G:

$(x_1, \bar{x}_1)$

(Δ_{11}, g_{11})

(z_{11}, w_{11}, f_{11})

(z_{1j}, w_{1j}, f_{1j}), $j = 2, 3, 4$

(w_{10}, f_{10})

Connect C_1 to 1 in MST: choose $g_{11} \rightarrow f_{11} \Rightarrow$ w_{11} forbidden other w_{1j}: either forbidden or chosen $\Rightarrow$ z_{1j} forbidden $\Longrightarrow$ only connection via x_1

MST: exactly one edge g_{ij} for every clause j

MSTCG with a 3-ladder

Construction

It can be shown:
There is a truth assignment for a 3-Sat instance with k clauses
$\Longleftrightarrow$
there is a conflict-free spanning tree with weight $\leq k$.

MSTCG with a 3-ladder

Construction

It can be shown:
There is a truth assignment for a 3-Sat instance with k clauses
$\Longleftrightarrow$
there is a conflict-free spanning tree with weight $\leq k$.

Conclusion

Conflicts makes optimization (and life in general) much more difficult.

$\Longrightarrow$ try to avoid conflicts whenever possible!

Thank you for your attention!