Enumerating Tree Decompositions

Nofar Carmeli Batya Kenig Benny Kimelfeld
Technion – Israel Institute of Technology

Motivation

- Q1: Is there a manager with a relative in the company?

Works:

Employee	Project
Alice	A
Anna	A
Bob	B
Barak	B

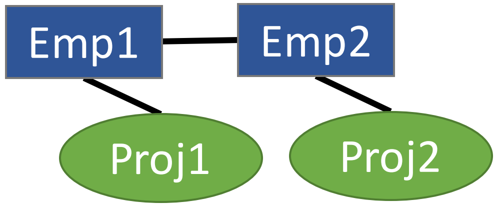
Manages:

Employee	Project
Ester	A
Fady	B
Gil	C
Hava	D

Relative:

Emp1	Emp2
Barak	Hava
Anna	Ester
Carl	Clement
David	Dan

$$\mathrm{works}(\mathrm{Emp1}, \mathrm{Proj1}) \wedge \mathrm{manages}(\mathrm{Emp2}, \mathrm{Proj2}) \wedge \mathrm{relative}(\mathrm{Emp1}, \mathrm{Emp2})$$



Motivation

- Q2: Is there an employee managed by a relative?

Works:

Employee	Project
Alice	A
Anna	A
Bob	B
Barak	B

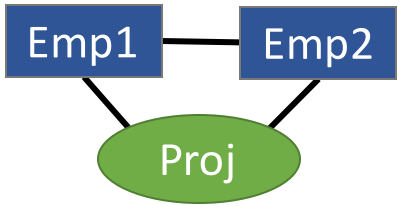
Manages:

Employee	Project
Ester	A
Fady	B
Gil	C
Hava	D

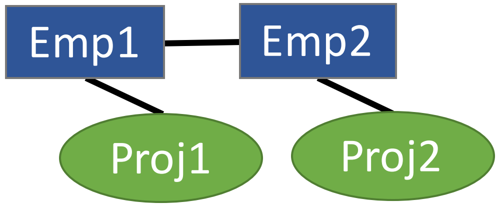
Relative:

Employee	Employee
Barak	Hava
Anna	Ester
Carl	Clement
David	Dan

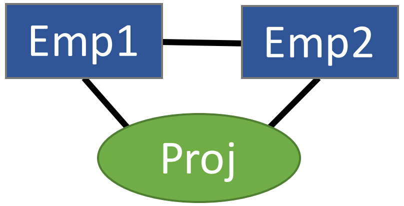
$$\mathrm{works}(\mathrm{Emp1}, \mathrm{Proj}) \wedge \mathrm{manages}(\mathrm{Emp2}, \mathrm{Proj}) \wedge \mathrm{relative}(\mathrm{Emp1}, \mathrm{Emp2})$$



Motivation



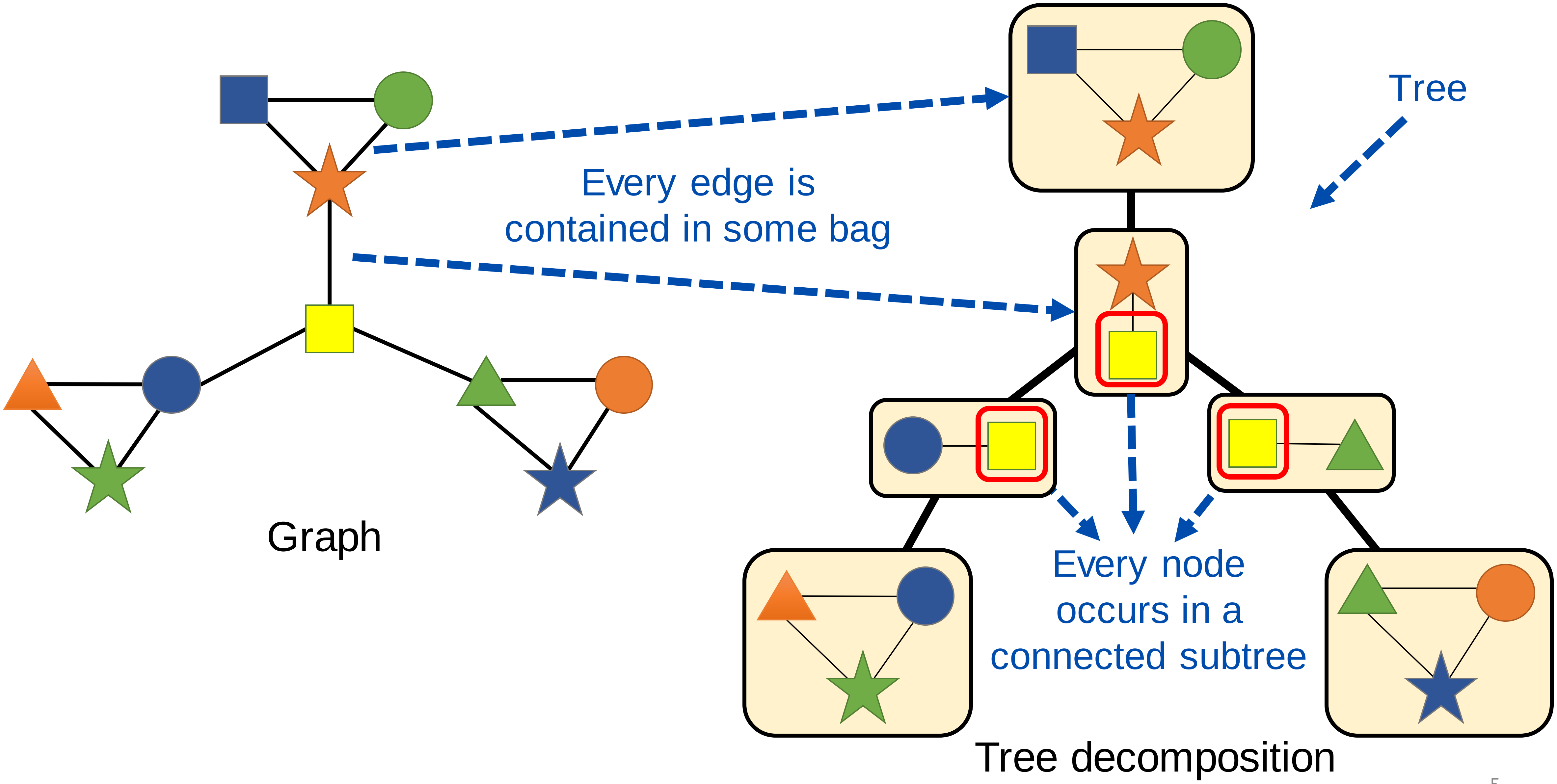
Q1 – acyclic



Q2 – cyclic

- Evaluating a general conjunctive query is NP-complete [Chandra&Merlin77]
- Efficient algorithm for acyclic conjunctive queries [Yannakakis81]
- A **tree decomposition** allows applying Yannakakis's to general conjunctive queries [Chekuri&Rajaraman97]

Tree Decompositions

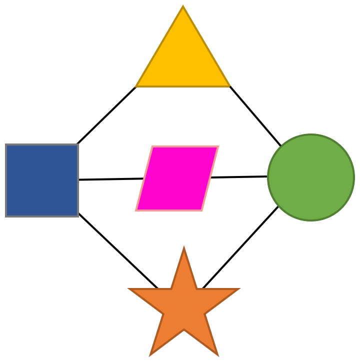


Tree Decompositions

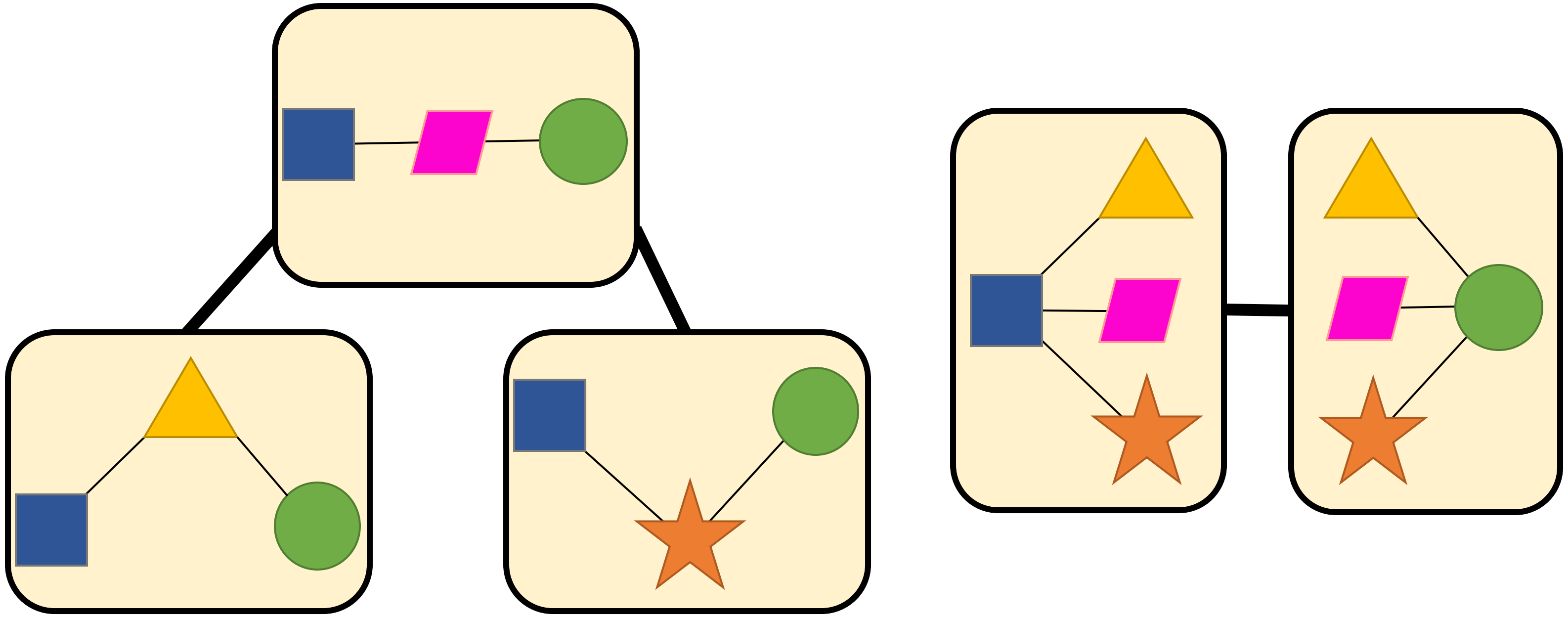
- Many applications beyond join optimization:
  - Games
    - Nash equilibria computation [Gottlob+05]
  - Bioinformatics
    - prediction of RNA secondary structure [Zhao+06]
  - Probabilistic graphical models
    - statistical inference [Lauritzen&Spiegelhalter88]
  - Constraint-satisfaction problems [Kolaitis&Vardi00]
  - Weighted model counting [Li+08]
  - ...

Which TD to use?

- A graph can have many TDs



Graph



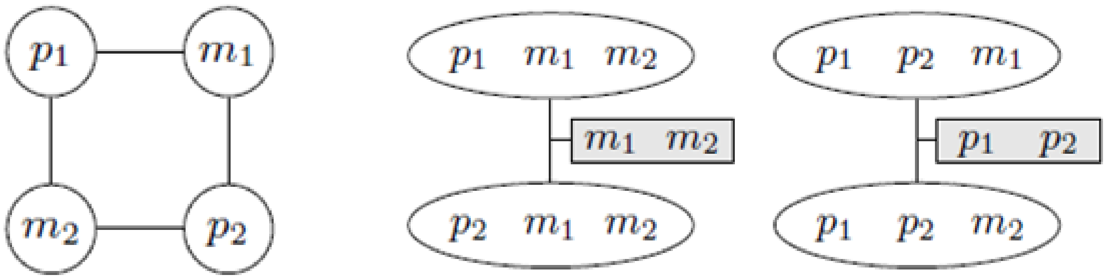
Tree decompositions

- We want the ‘best’ decomposition
- Common – minimize the cardinality of the largest bag (smallest width)

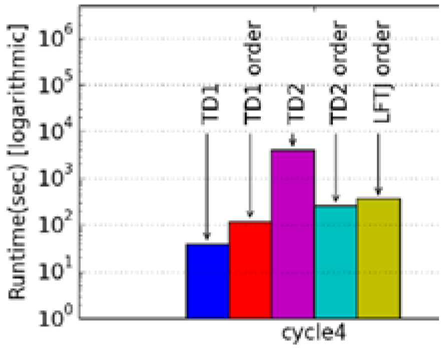
Which TD to use?

- Smallest width is NP-hard [Arnborg+87]
- Common: Use heuristics
- Width isn't enough

Flexible Caching in Trie Joins [Kalinsky+16]



Query TD1 TD2



TD1 runs 100 times faster!

- Different applications – different requirements

TD enumeration is needed

- Related work:
  - Query plans using generalized hypertree decompositions [Tu&Ré15]
    - Generate all, choose one
    - No complexity guarantees
    - Works for small graphs
  - Improving the efficiency of dynamic programing on tree decompositions using machine learning [Abseher+15]
    - Heuristically generate a pool, choose using machine learning
    - Limited pool, may not contain the best
  - Can we enumerate the TDs with efficiency guarantees?

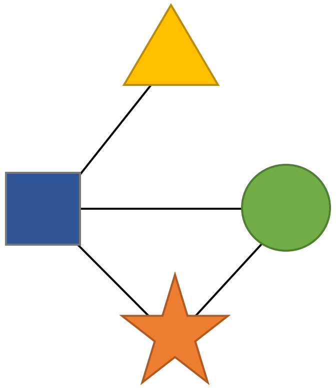
Goal

Problem: Enumerating all TDs of a graph

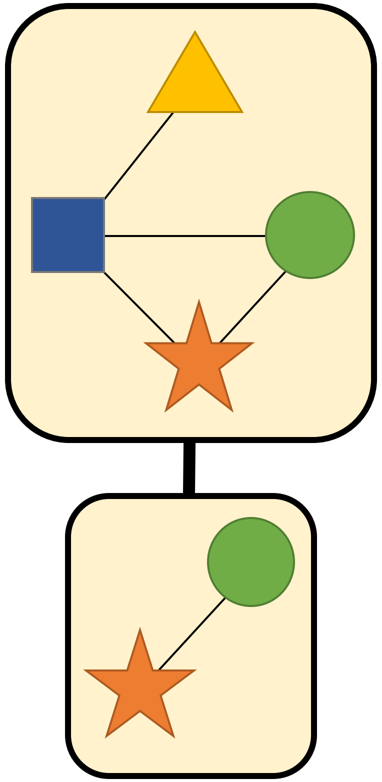
1. Complexity guarantees
2. Effective practical solution

There can be exponentially many TDs!

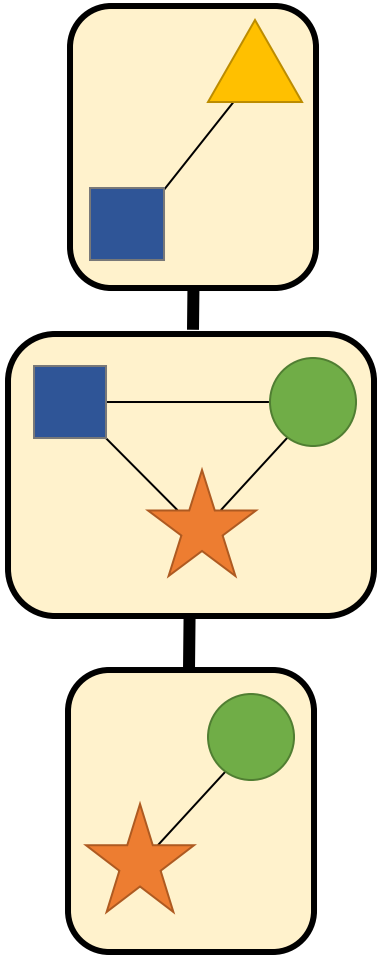
Which TDs to Generate?



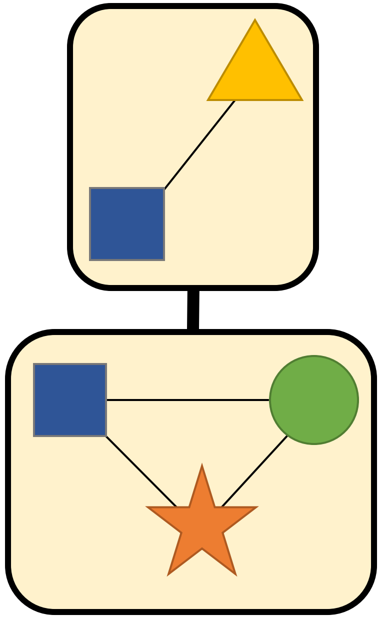
Graph



Tree decomposition



Better tree decomposition



Better tree decomposition

Proper TDs

- We define “proper” TDs
- Intuitively, in a proper TD you cannot:
  - Split bags
  - Remove bags

Goal: Enumerating all proper TDs of a graph

Problem: exponentially many TDs, what is an “efficient” algorithm?

Efficiency of enumeration algorithms

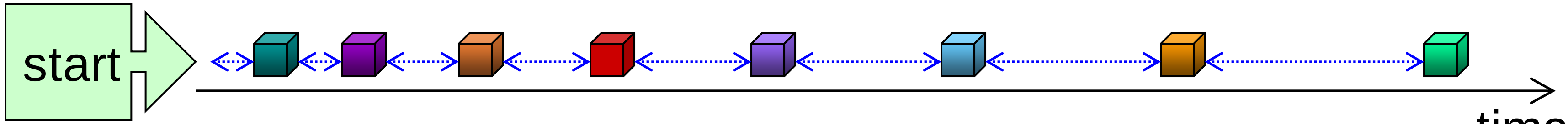
[Johnson, Papadimitriou, Yannakakis 88]

polynomial total time

Running time is polynomial in input + output

time

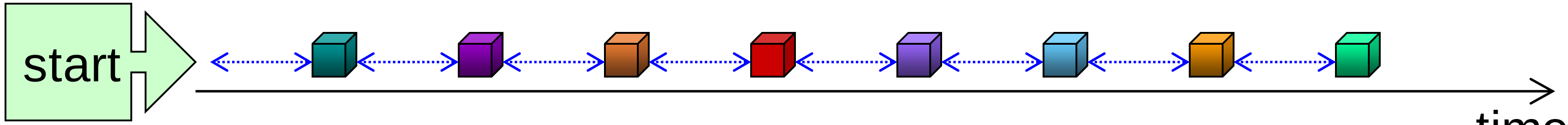
incremental polynomial time



Delay before answer *i* is polynomial in input + *i*

time

polynomial delay



Delay between successive answers is poly(input)

time

The main theoretical result

<u>Main Theorem:</u>

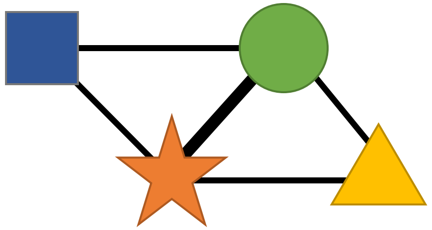
Given a graph, it is possible to enumerate in incremental polynomial time:

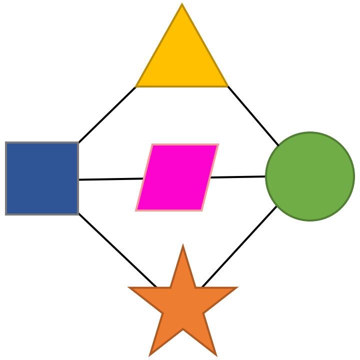
- The proper tree decompositions
- The *minimal triangulations*

Goal: Enumerate Proper TDs

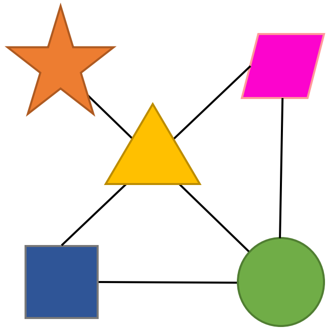
- Chord: An edge between two non-adjacent nodes in a cycle
- Chordal graph: Every cycle of length>3 has a chord

Chord:





Not Chordal

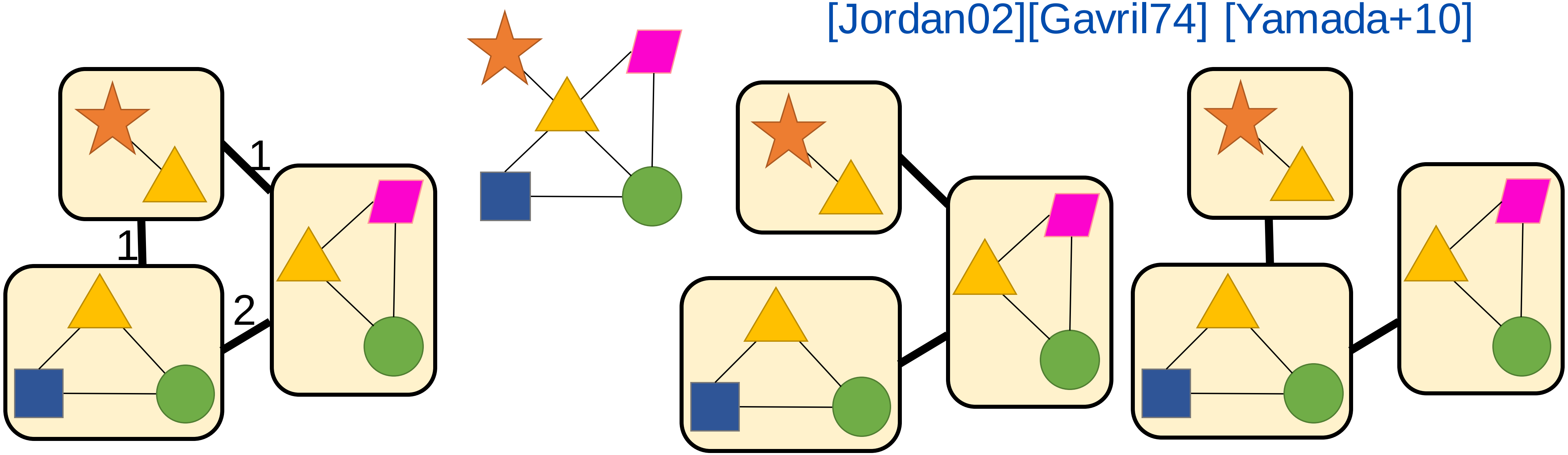


Chordal

Goal: Enumerate Proper TDs

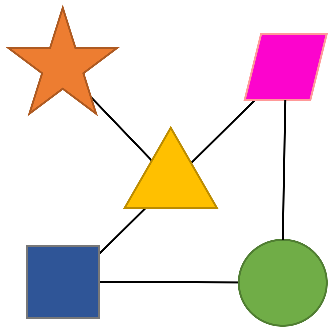
- Chord: An edge between two non-adjacent nodes in a cycle
- Chordal graph: Every cycle of length>3 has a chord
- Finding proper TDs of a chordal graph is easy
  - The bags are the maximal cliques
  - These TDs can be enumerated in polynomial delay

[Jordan02][Gavril74] [Yamada+10]

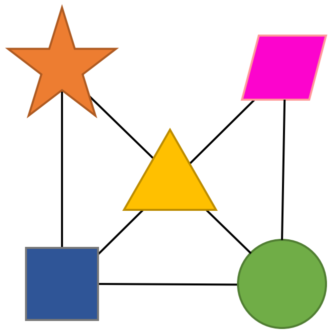


Goal: Enumerate Proper TDs

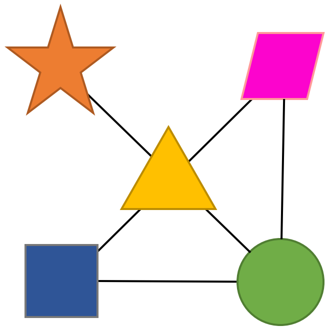
- Triangulation of a graph: Adding edges to make it chordal
- Minimal triangulation:
  Adding a proper subset of the edges does not make it chordal



Graph



Triangulation

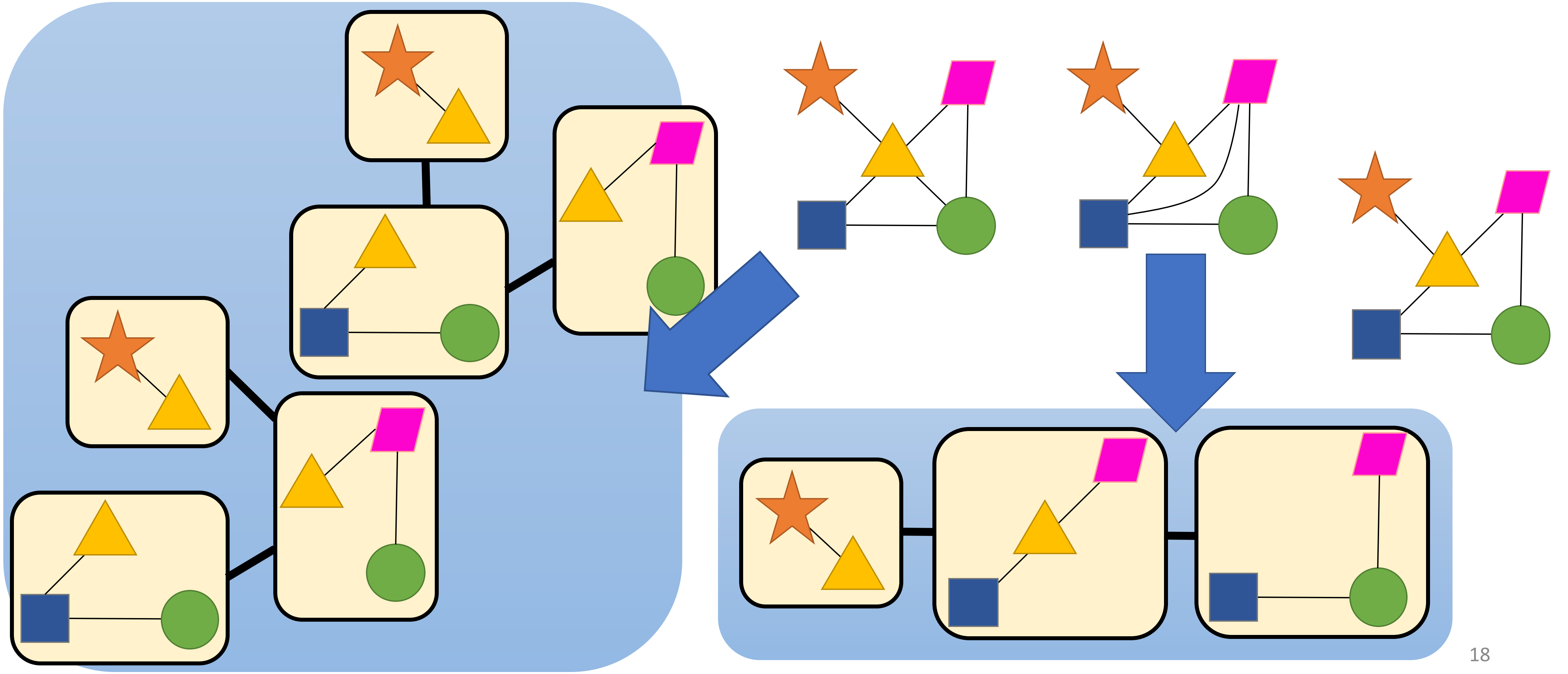


Minimal triangulation

Goal: Enumerate Proper TDs

- A bijection:

classes of bag equivalent proper TDs ↔ min triangulations



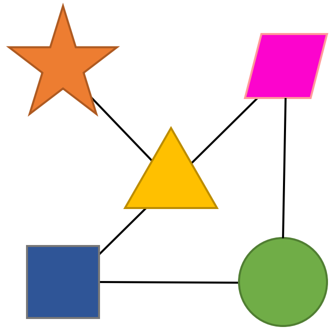
Goal: Enumerate Proper TDs

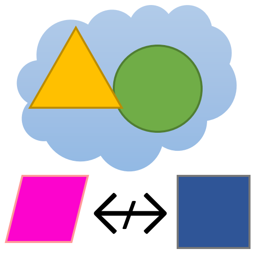
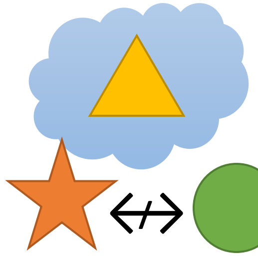
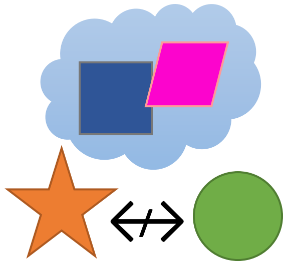
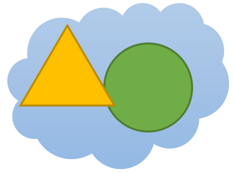
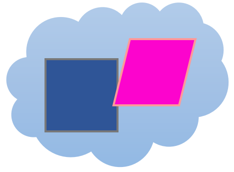
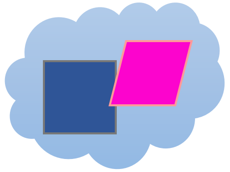
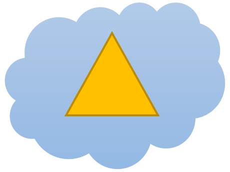
Goal: Enumerating all proper TDs of a graph

Goal: Enumerating all min triangulations of a graph

Goal: Enumerate Minimal Triangulations

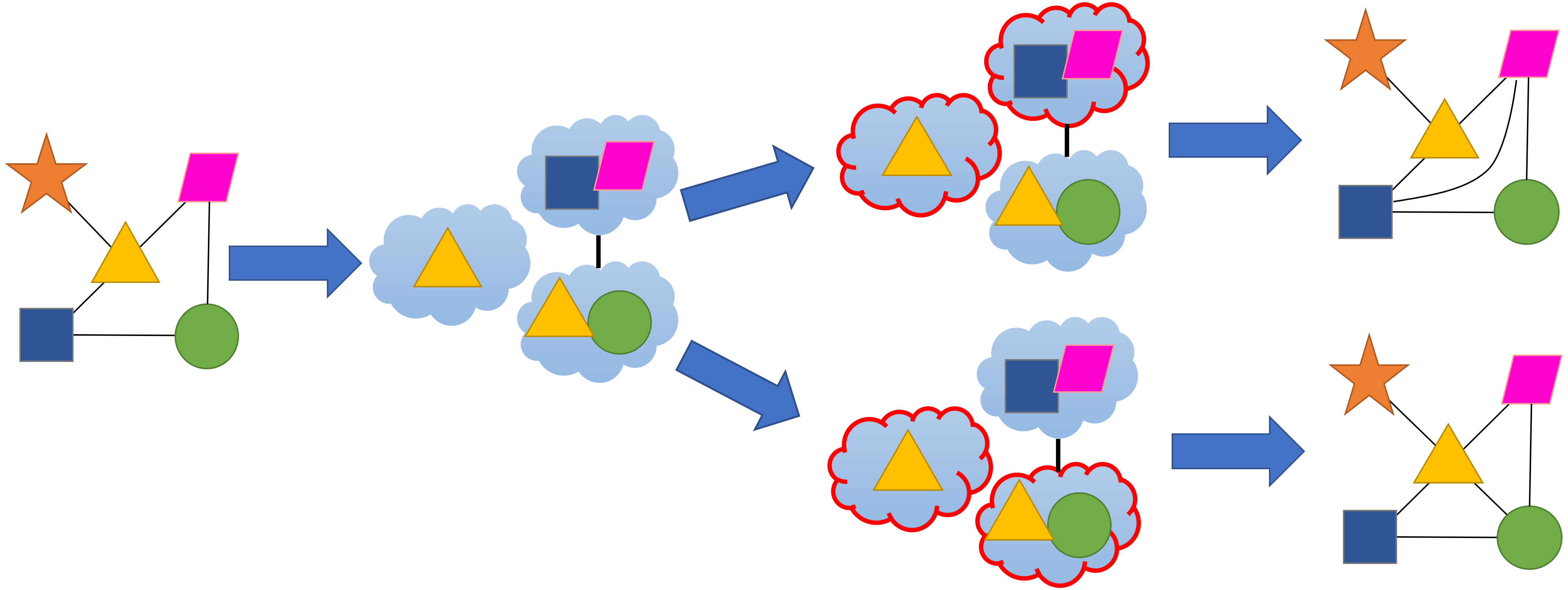
- Minimal Separator:
  Removing these nodes separates some u and v
  No proper subset separates u and v
- Crossing separators:
  One of them separates nodes of the other



- Minimal separators:   
- Crossing separators:  and 
- Parallel separators:  and 

Goal: Enumerate Minimal Triangulations

- A bijection [Parra&Scheffler97]:
  minimal triangulations ↔ maximal sets of non crossing
  minimal separators



Goal: Enumerate Proper TDs

Goal: Enumerating all min triangulations of a graph

⇓

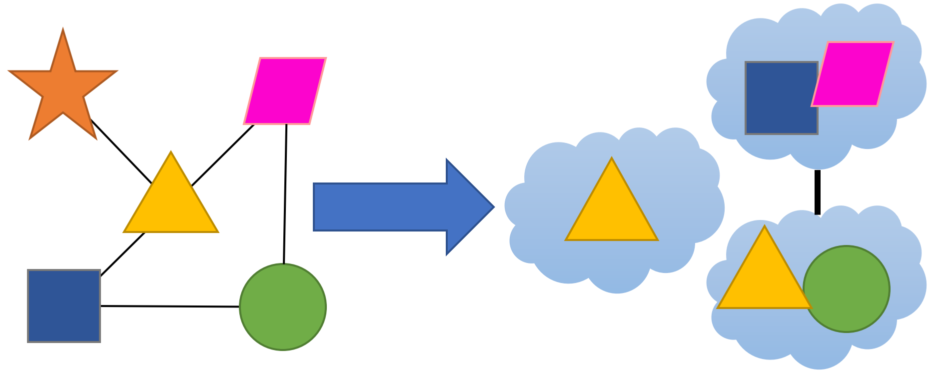
Goal: Enumerating all max independent sets of a graph

Goal: Enumerate Maximal Independent Sets

Enumerating max independent sets can be done in polynomial delay [Johnson+88]

Problem:
The graph may be of exponential size!

Challenge:
Solve without generating the graph



The Algorithm (Enumerating max independent sets)

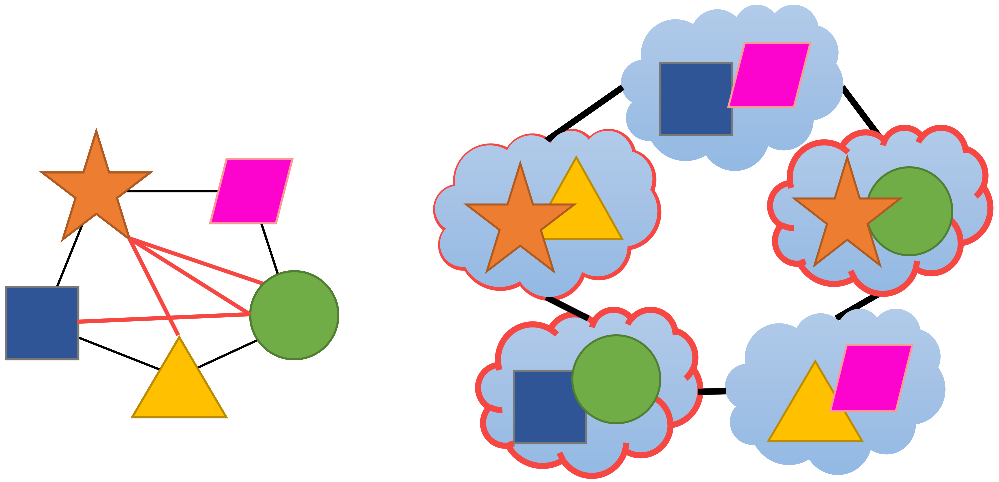
$J := \mathsf{Extend}(x, \emptyset)$
print J
$\mathcal{Q} := \{J\}$
$\mathcal{P} := \emptyset$
$\mathcal{V} := \emptyset$
$\text{iterator} := A_{\mathsf{V}}(x)$
while $\mathcal{Q} \neq \emptyset$ **do**
  $J := \mathcal{Q}.\mathsf{pop}()$
  $\mathcal{P}.\mathsf{push}(J)$
  for all $v \in \mathcal{V}$ **do**
    $J_v := \{v\} \cup \{u \in J \mid \neg A_{\mathsf{E}}(x, v, u)\}$
    $K := \mathsf{Extend}(x, J_v)$
    if $K \notin \mathcal{Q} \cup \mathcal{P}$ **then**
      print K
      $\mathcal{Q} := \mathcal{Q} \cup \{K\}$
  while $\mathcal{Q} = \emptyset$ **and** $\text{iterator}.\mathsf{hasNext}()$ **do**
    $v := \text{iterator}.\mathsf{next}()$
    $\mathcal{V} := \mathcal{V} \cup \{v\}$
    for all $J' \in \mathcal{P}$ **do**
      $J'_v := \{v\} \cup \{u \in J' \mid \neg A_{\mathsf{E}}(x, v, u)\}$
      $K := \mathsf{Extend}(x, J'_v)$
      if $K \notin \mathcal{Q} \cup \mathcal{P}$ **then**
        print K
        $\mathcal{Q} := \mathcal{Q} \cup \{K\}$

- Redesign of an algorithm for hereditary graph properties [Cohen+08]
- Assuming:
  - Efficiently enumerating nodes
  - Efficiently checking edges
  - Efficiently extending an independent set
  - Polynomial size of max independent sets
- Extends all nodes in the direction of all independent sets.
- Runs in incremental poly time

The Algorithm (Enumerating max independent sets)

```
J := Extend(x, ∅)
print J
Q := {J}
P := ∅
V := ∅
iterator := A_V(x)
while Q ≠ ∅ do
    J := Q.pop()
    P.push(J)
    for all v ∈ V do
        J_v := {v} ∪ {u ∈ J | ¬A_E(x, v, u)}
        K := Extend(x, J_v)
        if K ∉ Q ∪ P then
            print K
            Q := Q ∪ {K}
    while Q = ∅ and iterator.hasNext() do
        v := iterator.next()
        V := V ∪ {v}
        for all J' ∈ P do
            J'_v := {v} ∪ {u ∈ J' | ¬A_E(x, v, u)}
            K := Extend(x, J'_v)
            if K ∉ Q ∪ P then
                print K
                Q := Q ∪ {K}
```

- In our case, extending = triangulating
- We can use **any** triangulation or tree decomposition algorithm
- First result = algorithm's result



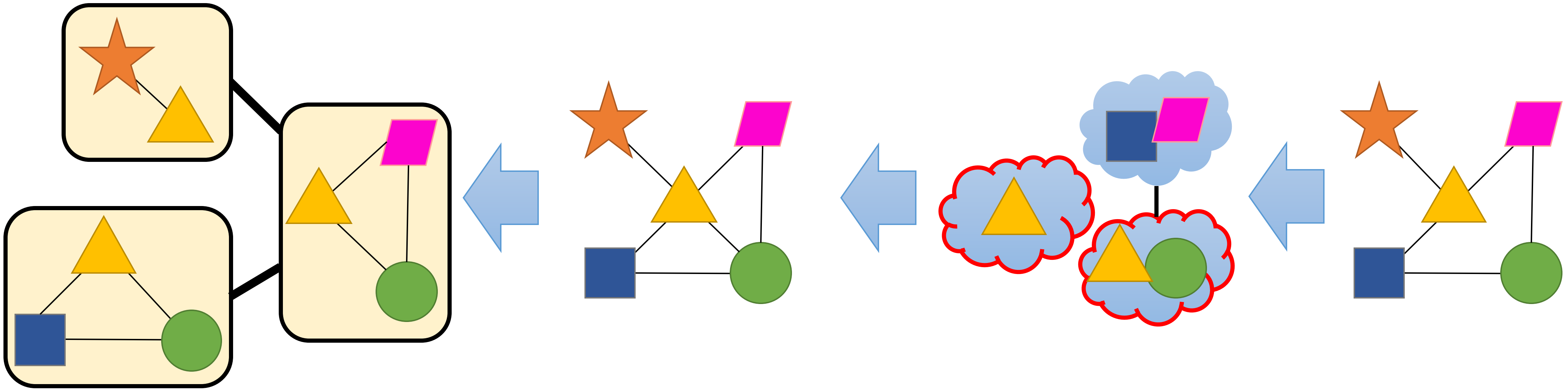
Goal: Enumerating max independent sets

Goal: Enumerating all max independent sets of a graph

Find a single minimal triangulation

Solution Summary

Enumerate proper TDs → Enumerate min triangulations → Enumerate max independent sets → Single min triangulation



Experiments

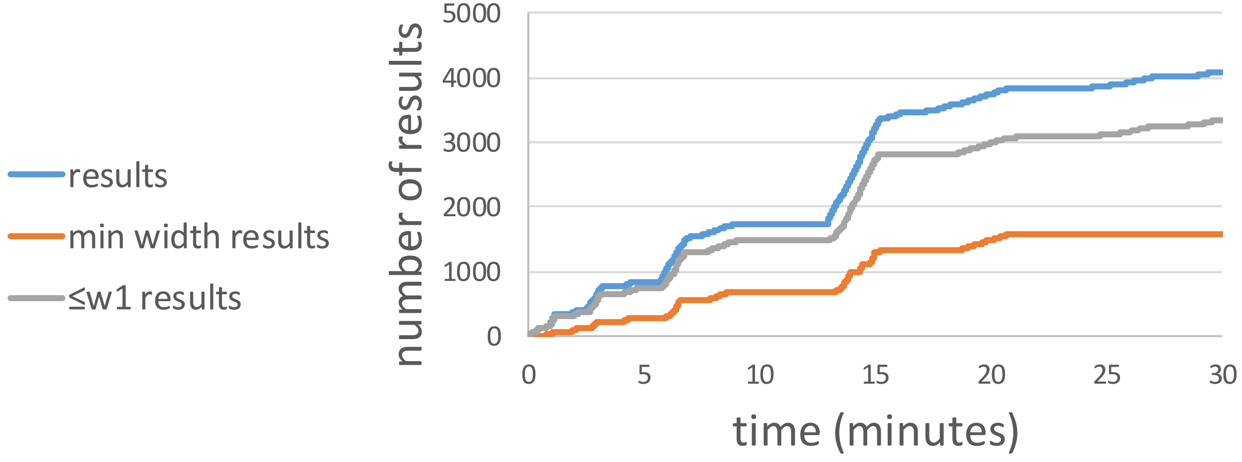
- Goals: check efficiency and quality
- C++ implementation
- Triangulation algorithms:
  - MCS-M [Berry+02]
  - LB-Triang [Berry+06] with min fill heuristics
- Benchmarks:
  - DunceCap [Tu&Ré15]
  - Heuristics (First result)

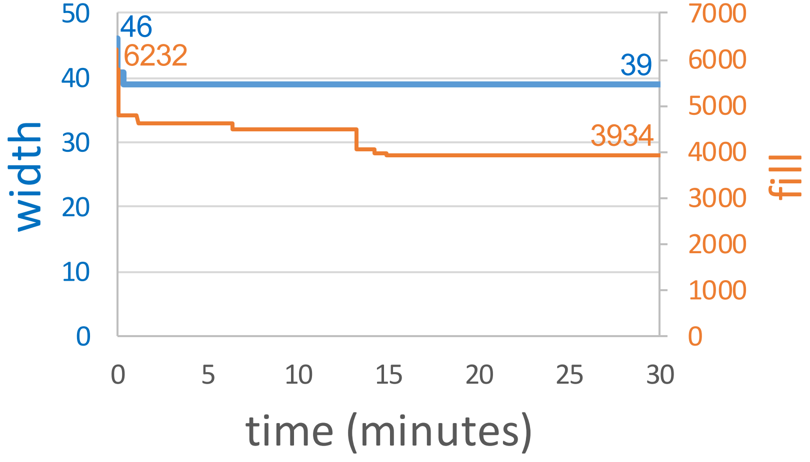
Experiments

- Datasets:
  - Database queries
    - TPC-H (LogicBlox translation)
    - 2-19 nodes, 1-46 edges
  - Probabilistic graphical models
    - UAI inference challenge
    - 60-1039 nodes, 135-1696 edges
  - Random
    - 30-200 nodes, 131-13955 edges

Experiments

- A single run (UAI, 414 nodes, 801 edges, MCS-M, 30 minutes)

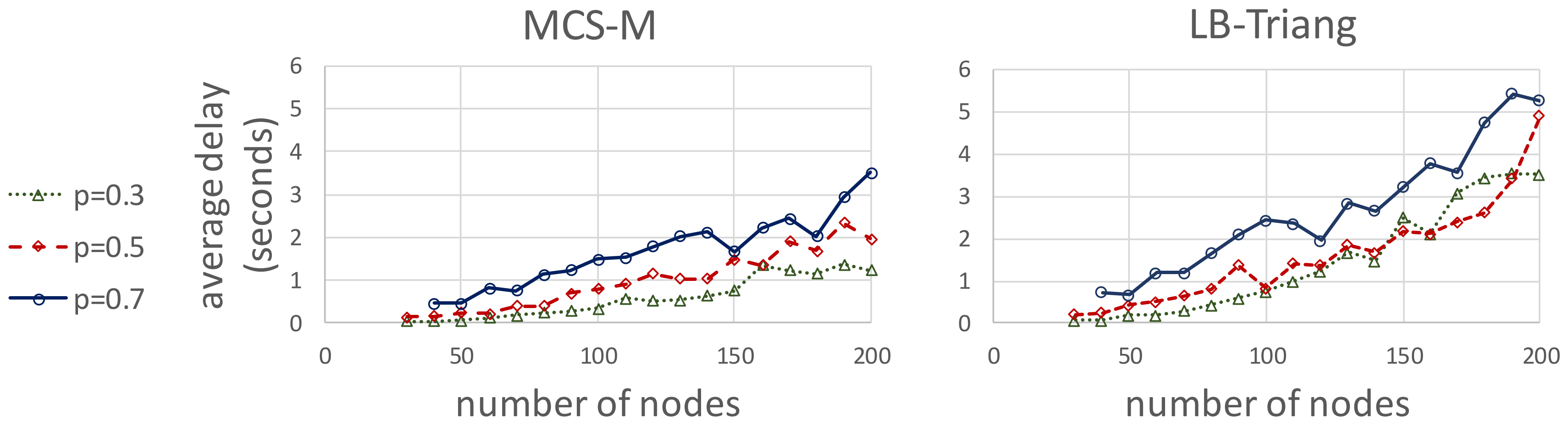




- Queries, completed within 5 seconds
  - 11 graphs: triangulated
  - 9 graphs: 2-5 triangulations
  - 1 graph: 588 triangulations
  - 1 graph: 700 triangulations

Experiments

- Random (30 minutes)



- Probabilistic graphical models (30 minutes)

alg.	measure	avg #results	avg #≤first	avg min	avg %improv	max %improv
MCS-M	width	33635.0	12733.4	20.2	2.6%	26.3%
MCS-M	fill	33635.0	12724.9	2043.8	14.4%	55.8%
LB-T(fill)	width	11998.3	4744.1	18.5	3.4%	20.7%
LB-T(fill)	fill	11998.3	1013.6	965.8	2.2%	27.6%

Future Work

- Practical
  - Parallelized implementation
  - Heuristics for ranked enumeration
- Theoretical
  - Polynomial delay
  - Restricted versions

Questions?