Computing H-Partitions in ASP and Datalog

Chloé Capon, Nicolas Lecomte and Jef Wijsen

University of Mons, Belgium

ASPOCP 2022
15th Workshop on Answer Set Programming and Other Computing Paradigms

July 31, 2022

Motivations

Initial question

Can a theoretically faster algorithm replace efficiently another theoretically slower one?

Motivations

Initial question

Can a theoretically faster algorithm replace efficiently another theoretically slower one?

- ASP can be used to solve efficiently a lot of problems with a guess-and-check approach;

Motivations

Initial question

Can a theoretically faster algorithm replace efficiently another theoretically slower one?

- ASP can be used to solve efficiently a lot of problems with a guess-and-check approach;
- What if a problem that was in NP, has been proved to be in P? Can the associated new approach be used to get better results than the older one?

Motivations

Initial question

Can a theoretically faster algorithm replace efficiently another theoretically slower one?

- ASP can be used to solve efficiently a lot of problems with a guess-and-check approach;
- What if a problem that was in NP, has been proved to be in P? Can the associated new approach be used to get better results than the older one?

Let's investigate on a problem: finding H-Partitions of a graph.

Table of contents

H-Partition

Given an undirected simple graph G:

> Is there a labeling of G's vertices that respects some constraints encoded in a model graph H ?

H-Partition

Given an undirected simple graph G:

> Is there a labeling of G's vertices that respects some constraints encoded in a model graph H ?

A model graph H is an undirected graph with four vertices, called A, B, C and D. Every edge is of exactly one of two types: full or dotted.

H-Partition

Given an undirected simple graph G:

> Is there a labeling of G's vertices that respects some constraints encoded in a model graph H ?

A model graph H is an undirected graph with four vertices, called A, B, C and D. Every edge is of exactly one of two types: full or dotted.

Constraints

- full edge between A and B in H $\rightsquigarrow$ each vertex labeled by A must be adjacent to each vertex labeled by B in G;
- dotted edge between A and B in H $\rightsquigarrow$ each vertex labeled by A must be nonadjacent to each vertex labeled by B in G.

H-Partition

Given an undirected simple graph G:

> Is there a labeling of G's vertices that respects some constraints encoded in a model graph H ?

A model graph H is an undirected graph with four vertices, called A, B, C and D. Every edge is of exactly one of two types: full or dotted.

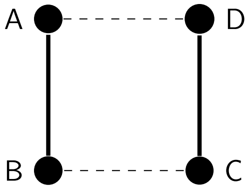
Constraints

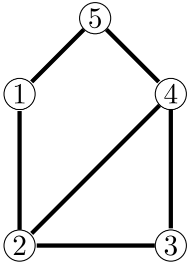
- full edge between A and B in H $\rightsquigarrow$ each vertex labeled by A must be adjacent to each vertex labeled by B in G;
- dotted edge between A and B in H $\rightsquigarrow$ each vertex labeled by A must be nonadjacent to each vertex labeled by B in G.

If there exists such a labeling, (H, G) is called a yes-instance; otherwise (H, G) is a no-instance.

H-Partitions

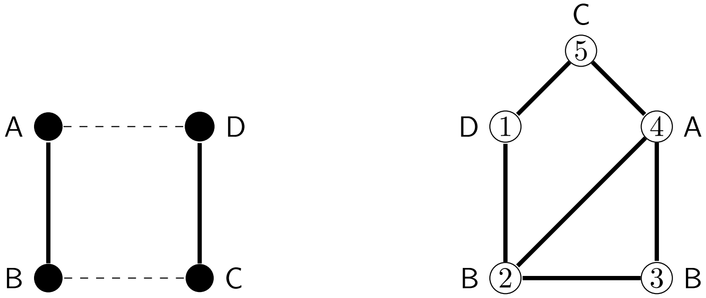
Example





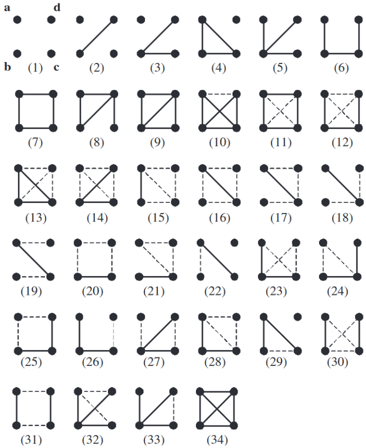
H-Partitions

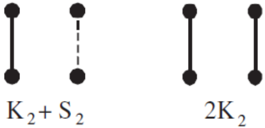
Example



Model graphs

All possible model graphs (up to isomorphism):





Finding H-Partitions

- The H-partitioning problem for $K_2 + S_2$, is in polynomial time [dFKKR00, KR07];
- The H-partitioning problem for $2K_2$ is NP-complete [CDFG05];

Finding H-Partitions

- The H-partitioning problem for $K_2 + S_2$, is in polynomial time [dFKKR00, KR07];
- The H-partitioning problem for $2K_2$ is NP-complete [CDFG05];
- For the other model graphs, Dantas et al. [DdFGK05] provide a polynomial-time algorithm, of low polynomial degree, for the H-partitioning problem;

Finding H-Partitions

- The H-partitioning problem for $K_2 + S_2$, is in polynomial time [dFKKR00, KR07];
- The H-partitioning problem for $2K_2$ is NP-complete [CDFG05];
- For the other model graphs, Dantas et al. [DdFGK05] provide a polynomial-time algorithm, of low polynomial degree, for the H-partitioning problem;

$\rightsquigarrow$ We experimentally compare a Datalog with stratified negation program with a guess-and-check ASP program.

ASP program

Guess and check

```
% Every vertex goes in exactly one partition.
1 { placedIn(X,P) : partition(P) } 1 :- vertex(X).
```

ASP program

Guess and check

```
% Every vertex goes in exactly one partition.
1 { placedIn(X,P) : partition(P) } 1 :- vertex(X).

% No partition is empty.
filled(P) :- placedIn(X,P).
:- partition(P), not filled(P).
```

ASP program

Guess and check

```
% Every vertex goes in exactly one partition.
1 { placedIn(X,P) : partition(P) } 1 :- vertex(X).

% No partition is empty.
filled(P) :- placedIn(X,P).
:- partition(P), not filled(P).

% Constraints of the model graph.
:- placedIn(X,P), placedIn(Y,Q), full(P,Q), not e(X,Y).
:- placedIn(X,P), placedIn(Y,Q), dotted(P,Q), e(X,Y).
```

Datalog approach: Bases

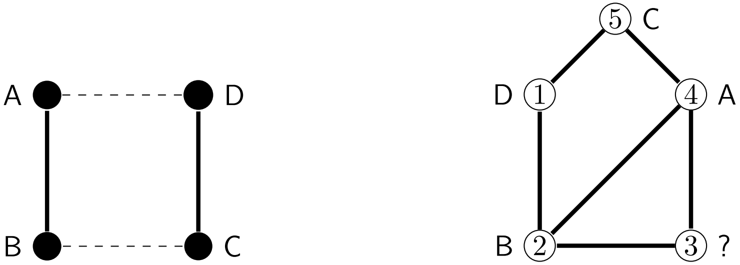
Base

We say that the quadruplet (x_A, x_B, x_C, x_D) is a base for H if the subgraph of G induced by these vertices is a yes-instance of H-PARTITION(H) for a labeling where x_A, x_B, x_C, and x_D are respectively labeled by A, B, C, and D.

Datalog approach: Bases

Base

We say that the quadruplet (x_A, x_B, x_C, x_D) is a base for H if the subgraph of G induced by these vertices is a yes-instance of H-PARTITION(H) for a labeling where x_A, x_B, x_C, and x_D are respectively labeled by A, B, C, and D.



$(4, 2, 5, 1)$ is a base for H.

Datalog program

Algorithm:

For each base until we find a solution:

[1] for model graphs (7), (10) and (11) some additional tests are required.

Datalog program

Algorithm:

For each base until we find a solution:

1. Pick a base;

[1] for model graphs (7), (10) and (11) some additional tests are required.

Datalog program

Algorithm:

For each base until we find a solution:

1. Pick a base;
2. Check if this base can be extended to a complete labeling of all vertices; repeatedly pick an unlabeled vertex, and compute its possible labels:

[1] for model graphs (7), (10) and (11) some additional tests are required.

Datalog program

Algorithm:

For each base until we find a solution:

1. Pick a base;
2. Check if this base can be extended to a complete labeling of all vertices; repeatedly pick an unlabeled vertex, and compute its possible labels:
   - if only one label is possible $\implies$ label that vertex with it;

[1] for model graphs (7), (10) and (11) some additional tests are required.

Datalog program

Algorithm:

For each base until we find a solution:

1. Pick a base;
2. Check if this base can be extended to a complete labeling of all vertices; repeatedly pick an unlabeled vertex, and compute its possible labels:
   - if only one label is possible $\implies$ label that vertex with it;
   - if no label is possible $\implies$ cannot be extended to a complete labeling.

[1] for model graphs (7), (10) and (11) some additional tests are required.

Datalog program

Algorithm:

For each base until we find a solution:

1. Pick a base;
2. Check if this base can be extended to a complete labeling of all vertices; repeatedly pick an unlabeled vertex, and compute its possible labels:
   - if only one label is possible $\implies$ label that vertex with it;
   - if no label is possible $\implies$ cannot be extended to a complete labeling.
3. If for every unlabeled vertex at least two labels remain possible then G is a yes-instance [1].

[1] for model graphs (7), (10) and (11) some additional tests are required.

Generating instances

To compare the efficiency of our programs, we need to generate yes-instances and no-instances of arbitrary size.

Generating instances

To compare the efficiency of our programs, we need to generate yes-instances and no-instances of arbitrary size.

⇝ We need to distinguishing between yes-instances and no-instances.

Motivation:

- on a yes-instance, an algorithm can stop as soon as a labeling is found;
- on no-instances no such early stopping is possible.

Generation of no-instances

model graph	# yes	# no	model graph	# yes	# no
(1)	1000	0	(18)	999	1
(2)	1000	0	(19)	185	815
(3)	1000	0	(20)	15	985
(4)	998	2	(21)	35	965
(5)	15	985	(22)	1000	0
(6)	175	825	(23)	15	985
(7)	13	987	(24)	15	985
(8)	15	985	(25)	14	986
(9)	13	987	(26)	56	944
(10)	9	991	(27)	15	985
(11)	0	1000	(28)	5	995
(12)	0	1000	(29)	1000	0
(13)	0	1000	(30)	0	1000
(14)	2	998	(31)	0	1000
(15)	4	996	(32)	6	994
(16)	4	996	(33)	15	985
(17)	183	817	(34)	12	988

For most cases: repeatedly generating random graphs will quickly lead to a no-instance.

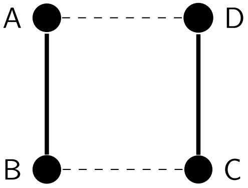
Generation of yes-instances

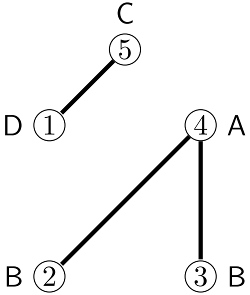
Let $a, b, c, d \in \mathbb{N}_0$ such that $a + b + c + d = n$. One can wonder if there exists a yes-instance with m edges where the numbers of vertices labeled by A, B, C, and D are, respectively, a, b, c, and d.

Generation of yes-instances

Let $a, b, c, d \in \mathbb{N}_0$ such that $a + b + c + d = n$. One can wonder if there exists a yes-instance with m edges where the numbers of vertices labeled by A, B, C, and D are, respectively, a, b, c, and d.

- G_{min} is the graph where the only edges are the ones following the full constraints from H;



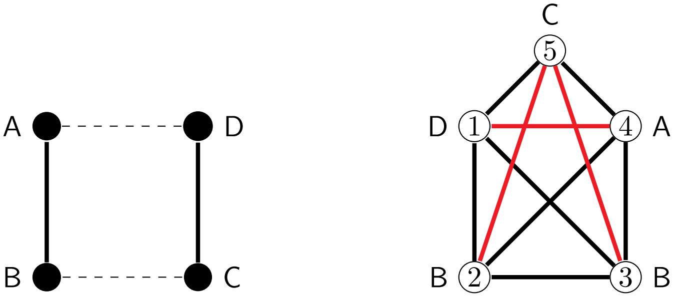


Here, $n = 5$ with $a = c = d = 1$ and $b = 2$.

Generation of yes-instances

Let $a, b, c, d \in \mathbb{N}_0$ such that $a + b + c + d = n$. One can wonder if there exists a yes-instance with m edges where the numbers of vertices labeled by A, B, C, and D are, respectively, a, b, c, and d.

- G_{min} is the graph where the only edges are the ones following the full constraints from H;
- G_{max} is the complete graph without the edges following the dotted constraints from H.

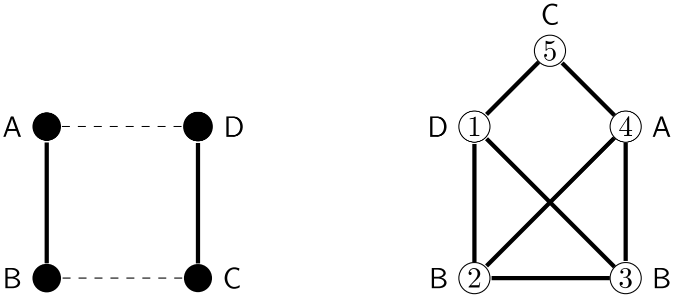


Here, $n = 5$ with $a = c = d = 1$ and $b = 2$.

Generation of yes-instances

Let $a, b, c, d \in \mathbb{N}_0$ such that $a + b + c + d = n$. One can wonder if there exists a yes-instance with m edges where the numbers of vertices labeled by A, B, C, and D are, respectively, a, b, c, and d.

- G_{min} is the graph where the only edges are the ones following the full constraints from H;
- G_{max} is the complete graph without the edges following the dotted constraints from H.



Here, $n = 5$ with $a = c = d = 1$ and $b = 2$.

Generation of yes-instances

Let $a, b, c, d \in \mathbb{N}_0$ such that $a + b + c + d = n$. One can wonder if there exists a yes-instance with m edges where the numbers of vertices labeled by A, B, C, and D are, respectively, a, b, c, and d.

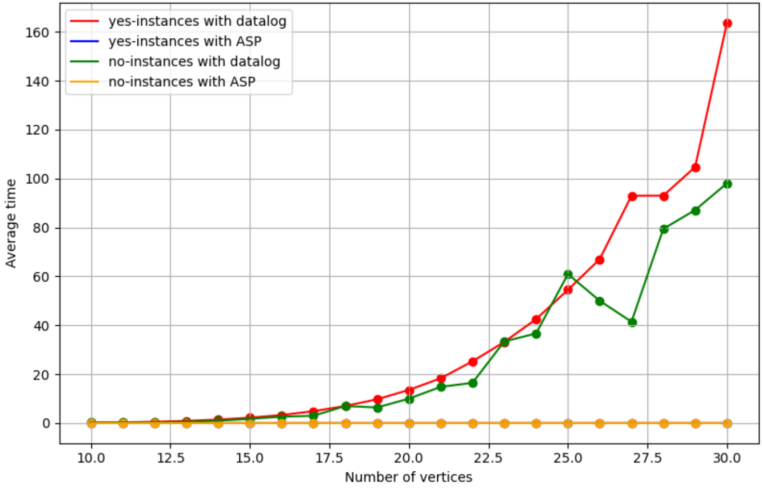
- G_{min} is the graph where the only edges are the ones following the full constraints from H;
- G_{max} is the complete graph without the edges following the dotted constraints from H.

Theorem

Let H be a model graph. Let $a, b, c, d \in \mathbb{N}_0$ and $n = a + b + c + d$. Every graph G such that $G_{min} \subseteq G \subseteq G_{max}$ has a solution with a vertices labeled by A, b labeled by B, etc.

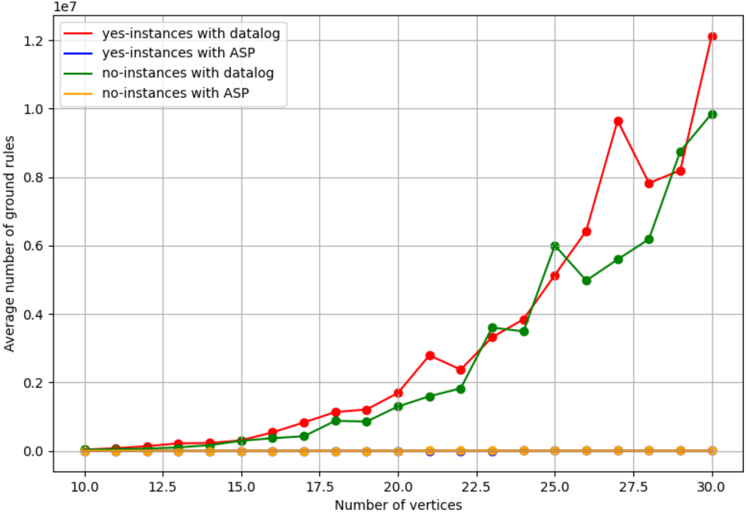
Experimentations

Time of resolution with Clingo



Experimentations

Number of ground rules with Clingo



Conclusion

- We have given two programs:
  1. a program in Datalog with stratified negation;
  2. a guess-and-check ASP program.
- We have generated yes and no-instances to compare the programs.

$\rightsquigarrow$ The Datalog approach is slower than the guess-and-check because it leads to too many ground rules.

Future work

- Automatic generation of no-instances;
- Test on a datalog engine.

Thank you for your attention!

Bibliography I

C. N. Campos, Simone Dantas, Luérbio Faria, and Sylvain Gravier.
$2K_2$-partition problem.
Electron. Notes Discret. Math., 22:217–221, 2005.

Simone Dantas, Celina M. H. de Figueiredo, Sylvain Gravier, and Sulamita Klein.
Finding H-partitions efficiently.
RAIRO Theor. Informatics Appl., 39(1):133–144, 2005.

Celina M. H. de Figueiredo, Sulamita Klein, Yoshiharu Kohayakawa, and Bruce A. Reed.
Finding skew partitions efficiently.
J. Algorithms, 37(2):505–521, 2000.

Bibliography II

William S. Kennedy and Bruce A. Reed.
Fast skew partition recognition.
In Hiro Ito, Mikio Kano, Naoki Katoh, and Yushi Uno, editors, *Computational Geometry and Graph Theory - International Conference, KyotoCGGT 2007, Kyoto, Japan, June 11-15, 2007. Revised Selected Papers*, volume 4535 of *Lecture Notes in Computer Science*, pages 101–107. Springer, 2007.