Token Sliding on chordal graphs

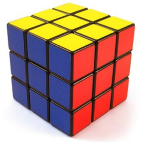
Nicolas Bousquet
joint work with Marthe Bonamy (LaBRI, Bordeaux, France)

Banff - Reconfiguration Workshop

Reconfiguration of Independent Sets

Introduced by Hearn and Demaine in 2005 in a general study of one-player games :

> *A one-player game is a puzzle : one player makes a series of moves, trying to accomplish some goal.*

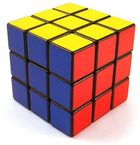


Question :
Giving my current position, can I reach my target position ?

Reconfiguration of Independent Sets

Introduced by Hearn and Demaine in 2005 in a general study of one-player games :

> *A one-player game is a puzzle : one player makes a series of moves, trying to accomplish some goal.*



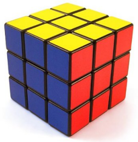
Question :
Giving my current position, can I reach my target position ?

- Equivalence with reconfiguration of satisfiability constraints.
- Generalize the Warehouseman's problem (motion of robots).

Reconfiguration of Independent Sets

Introduced by Hearn and Demaine in 2005 in a general study of one-player games :

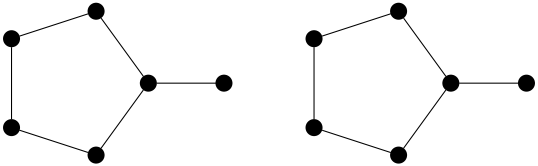
> *A one-player game is a puzzle : one player makes a series of moves, trying to accomplish some goal.*



Question :
Giving my current position, can I reach my target position ?

- Equivalence with reconfiguration of satisfiability constraints.
- Generalize the Warehouseman's problem (motion of robots).
- Introduced for colorings, satisfiability problems, dominating sets, cliques, list colorings, bases of matroids...

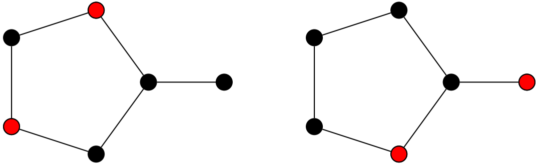
Token Sliding



Definition (TS-sequence)

A TS-sequence $I_1, \ldots, I_\ell$ of independent sets is a sequence such that there exist $v \in I_{j+1}$ and $u \in I_j$ such that $I_{j+1} = I_j \cup \{v\} \setminus \{u\}$ and uv is an edge.

Token Sliding



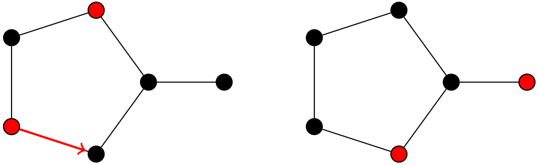
Definition (TS-sequence)

A TS-sequence $I_1, \ldots, I_\ell$ of independent sets is a sequence such that there exist $v \in I_{j+1}$ and $u \in I_j$ such that $I_{j+1} = I_j \cup \{v\} \setminus \{u\}$ and uv is an edge.

Equivalent formulation :

We slide tokens along the edges in such a way the set remains an independent set at any step.

Token Sliding



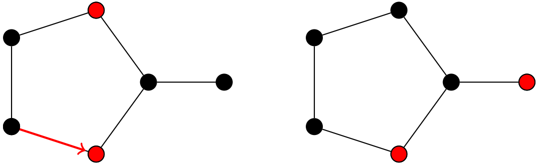
Definition (TS-sequence)

A TS-sequence $I_1, \ldots, I_\ell$ of independent sets is a sequence such that there exist $v \in I_{j+1}$ and $u \in I_j$ such that $I_{j+1} = I_j \cup \{v\} \setminus \{u\}$ and uv is an edge.

Equivalent formulation :

We slide tokens along the edges in such a way the set remains an independent set at any step.

Token Sliding



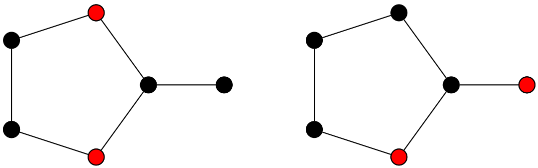
Definition (TS-sequence)

A TS-sequence $I_1, \ldots, I_\ell$ of independent sets is a sequence such that there exist $v \in I_{j+1}$ and $u \in I_j$ such that $I_{j+1} = I_j \cup \{v\} \setminus \{u\}$ and uv is an edge.

Equivalent formulation :

We slide tokens along the edges in such a way the set remains an independent set at any step.

Token Sliding



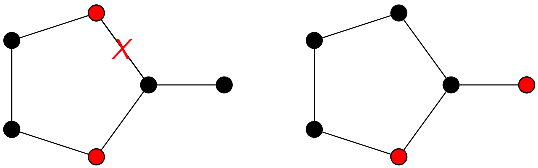
Definition (TS-sequence)

A TS-sequence $I_1, \dots, I_\ell$ of independent sets is a sequence such that there exist $v \in I_{j+1}$ and $u \in I_j$ such that $I_{j+1} = I_j \cup \{v\} \setminus \{u\}$ and uv is an edge.

Equivalent formulation :

We slide tokens along the edges in such a way the set remains an independent set at any step.

Token Sliding



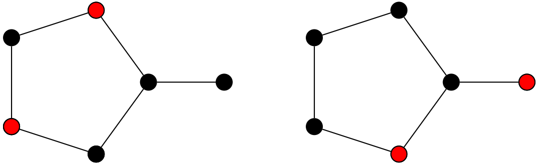
Definition (TS-sequence)

A TS-sequence $I_1, \ldots, I_\ell$ of independent sets is a sequence such that there exist $v \in I_{j+1}$ and $u \in I_j$ such that $I_{j+1} = I_j \cup \{v\} \setminus \{u\}$ and uv is an edge.

Equivalent formulation :

We slide tokens along the edges in such a way the set remains an independent set at any step.

Token Sliding



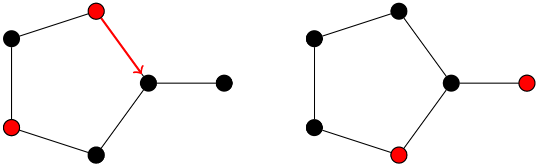
Definition (TS-sequence)

A TS-sequence $I_1, \ldots, I_\ell$ of independent sets is a sequence such that there exist $v \in I_{j+1}$ and $u \in I_j$ such that $I_{j+1} = I_j \cup \{v\} \setminus \{u\}$ and uv is an edge.

Equivalent formulation :

We slide tokens along the edges in such a way the set remains an independent set at any step.

Token Sliding



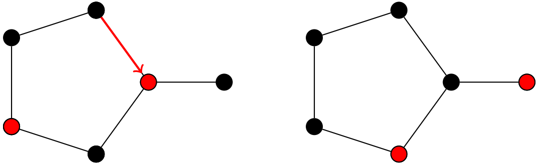
Definition (TS-sequence)

A TS-sequence $I_1, \dots, I_\ell$ of independent sets is a sequence such that there exist $v \in I_{j+1}$ and $u \in I_j$ such that $I_{j+1} = I_j \cup \{v\} \setminus \{u\}$ and uv is an edge.

Equivalent formulation :

We slide tokens along the edges in such a way the set remains an independent set at any step.

Token Sliding



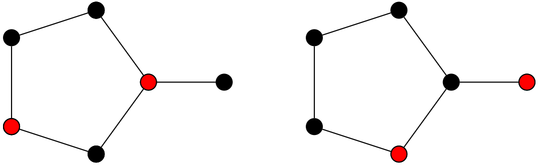
Definition (TS-sequence)

A TS-sequence $I_1, \ldots, I_\ell$ of independent sets is a sequence such that there exist $v \in I_{j+1}$ and $u \in I_j$ such that $I_{j+1} = I_j \cup \{v\} \setminus \{u\}$ and uv is an edge.

Equivalent formulation :

We slide tokens along the edges in such a way the set remains an independent set at any step.

Token Sliding



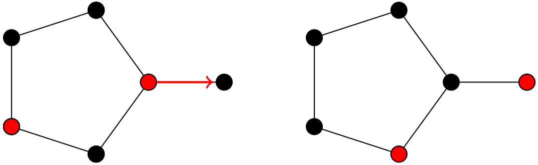
Definition (TS-sequence)

A TS-sequence $I_1, \ldots, I_\ell$ of independent sets is a sequence such that there exist $v \in I_{j+1}$ and $u \in I_j$ such that $I_{j+1} = I_j \cup \{v\} \setminus \{u\}$ and uv is an edge.

Equivalent formulation :

We slide tokens along the edges in such a way the set remains an independent set at any step.

Token Sliding



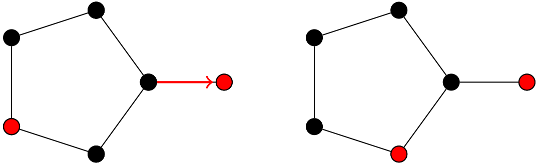
Definition (TS-sequence)

A TS-sequence $I_1, \ldots, I_\ell$ of independent sets is a sequence such that there exist $v \in I_{j+1}$ and $u \in I_j$ such that $I_{j+1} = I_j \cup \{v\} \setminus \{u\}$ and uv is an edge.

Equivalent formulation :

We slide tokens along the edges in such a way the set remains an independent set at any step.

Token Sliding



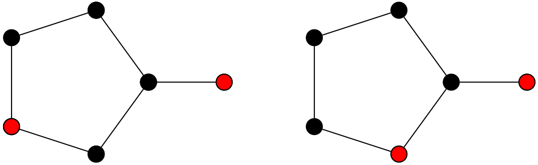
Definition (TS-sequence)

A TS-sequence $I_1, \ldots, I_\ell$ of independent sets is a sequence such that there exist $v \in I_{j+1}$ and $u \in I_j$ such that $I_{j+1} = I_j \cup \{v\} \setminus \{u\}$ and uv is an edge.

Equivalent formulation :

We slide tokens along the edges in such a way the set remains an independent set at any step.

Token Sliding



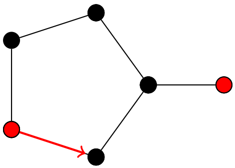
Definition (TS-sequence)

A TS-sequence $I_1, \ldots, I_\ell$ of independent sets is a sequence such that there exist $v \in I_{j+1}$ and $u \in I_j$ such that $I_{j+1} = I_j \cup \{v\} \setminus \{u\}$ and uv is an edge.

Equivalent formulation :

We slide tokens along the edges in such a way the set remains an independent set at any step.

Token Sliding



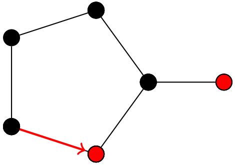
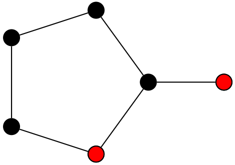
Definition (TS-sequence)

A TS-sequence $I_1, \ldots, I_\ell$ of independent sets is a sequence such that there exist $v \in I_{j+1}$ and $u \in I_j$ such that $I_{j+1} = I_j \cup \{v\} \setminus \{u\}$ and uv is an edge.

Equivalent formulation :

We slide tokens along the edges in such a way the set remains an independent set at any step.

Token Sliding

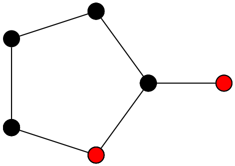
Definition (TS-sequence)

A TS-sequence $I_1, \ldots, I_\ell$ of independent sets is a sequence such that there exist $v \in I_{j+1}$ and $u \in I_j$ such that $I_{j+1} = I_j \cup \{v\} \setminus \{u\}$ and uv is an edge.

Equivalent formulation :

We slide tokens along the edges in such a way the set remains an independent set at any step.

Token Sliding



Definition (TS-sequence)

A TS-sequence $I_1, \ldots, I_\ell$ of independent sets is a sequence such that there exist $v \in I_{j+1}$ and $u \in I_j$ such that $I_{j+1} = I_j \cup \{v\} \setminus \{u\}$ and uv is an edge.

Equivalent formulation :

We slide tokens along the edges in such a way the set remains an independent set at any step.

Main questions

- **Reachability problem.** Given two configurations, is it possible to transform one into the other ?
- **Connectivity problem.** Given any pair of configurations, is it possible to transform one into the other ?
- **Minimization.** Given two configurations, what is the length of a shortest sequence ?

Main questions

- **Reachability problem.** Given two configurations, is it possible to transform one into the other ?
- **Connectivity problem.** Given any pair of configurations, is it possible to transform one into the other ?
- **Minimization.** Given two configurations, what is the length of a shortest sequence ?
- **Algorithmics.** Can we efficiently solve these questions ? (In polynomial time, FPT-time...).

Formal definition of the problems

TS-Reachability

Input : A graph G, $k \in \mathbb{N}$, two independent sets I, J of size k.
Output : YES iff there exists a TS-sequence from I to J.

TS-Connectivity

Input : A graph G, an integer k.
Output : YES iff it is possible to transform any independent set of size k into any other via a TS-sequence.

Theorem (Hearn, Demaine '05)

TS-Reachability is PSPACE-complete on planar graphs.

Polynomial time algorithms for :

- Demaine et al. Trees.
- Kamiński, Medvedev, Milanič. Cographs.
- Bonsma, Kamiński, Wrochna. Claw-free graphs.
- Fox-Epstein et al. Bipartite permutation graphs.

Our results

Question (Demaine et al.)

Can the TS-Reachability problem be decided on polynomial time on interval graphs ? on chordal graphs ?

Our results

Question (Demaine et al.)

Can the TS-Reachability problem be decided on polynomial time on interval graphs ? on chordal graphs ?

Answers

- YES on interval graphs.
  Both TS-Reachability and TS-Connectivity can be decided in polynomial time.

Our results

Question (Demaine et al.)

Can the TS-Reachability problem be decided on polynomial time on interval graphs ? on chordal graphs ?

Answers

- YES on interval graphs.
  Both TS-Reachability and TS-Connectivity can be decided in polynomial time.
- Maybe No on split graphs.
  Deciding TS-Connectivity is co-NP hard and co-W[2]-hard.
  (split graph $= V = V_1 \cup V_2$ where V_1 induces a clique and V_2 a stable set)

Our results

Question (Demaine et al.)

Can the TS-Reachability problem be decided on polynomial time on interval graphs ? on chordal graphs ?

Answers

- YES on interval graphs.
  Both TS-Reachability and TS-Connectivity can be decided in polynomial time.
- Maybe No on split graphs.
  Deciding TS-Connectivity is co-NP hard and co-W[2]-hard.
  (split graph $= V = V_1 \cup V_2$ where V_1 induces a clique and V_2 a stable set)

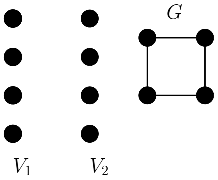
Remark :

With a similar construction $\Rightarrow$ TS-connectivity is co-NP hard and co-W[2]-hard on bipartite graphs.

Hardness result on split graphs

Let G be a graph. Create a graph H :

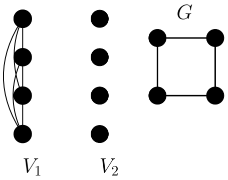
- Create two copies V_1, V_2 of $V(G)$.



Hardness result on split graphs

Let G be a graph. Create a graph H :

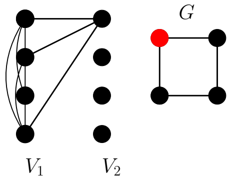
- Create two copies V_1, V_2 of $V(G)$.
- V_1 induces a clique and V_2 a stable set.



Hardness result on split graphs

Let G be a graph. Create a graph H :

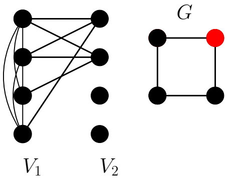
- Create two copies V_1, V_2 of $V(G)$.
- V_1 induces a clique and V_2 a stable set.
- We create an edge x_1y_2 iff $y \in N[x]$.



Hardness result on split graphs

Let G be a graph. Create a graph H :

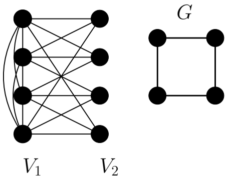
- Create two copies V_1, V_2 of $V(G)$.
- V_1 induces a clique and V_2 a stable set.
- We create an edge $x_1 y_2$ iff $y \in N[x]$.



Hardness result on split graphs

Let G be a graph. Create a graph H :

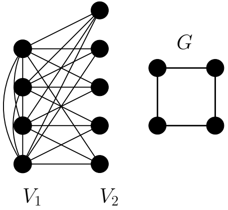
- Create two copies V_1, V_2 of $V(G)$.
- V_1 induces a clique and V_2 a stable set.
- We create an edge $x_1 y_2$ iff $y \in N[x]$.



Hardness result on split graphs

Let G be a graph. Create a graph H :

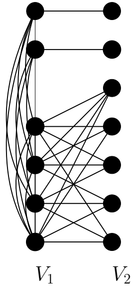
- Create two copies V_1, V_2 of $V(G)$.
- V_1 induces a clique and V_2 a stable set.
- We create an edge x_1y_2 iff $y \in N[x]$.
- We add a vertex in V_2 universal to V_1.

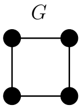


Hardness result on split graphs

Let G be a graph. Create a graph H :

- Create two copies V_1, V_2 of $V(G)$.
- V_1 induces a clique and V_2 a stable set.
- We create an edge $x_1 y_2$ iff $y \in N[x]$.
- We add a vertex in V_2 universal to V_1.
- Add a matching of size k.

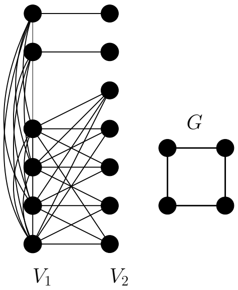




Hardness result on split graphs

Let G be a graph. Create a graph H :

- Create two copies V_1, V_2 of $V(G)$.
- V_1 induces a clique and V_2 a stable set.
- We create an edge $x_1 y_2$ iff $y \in N[x]$.
- We add a vertex in V_2 universal to V_1.
- Add a matching of size k.



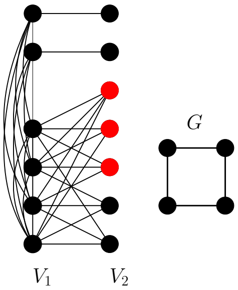
Lemma

We can transform any independent set of H of size $k+1$ into any other iff there is no dominating set of size k in G.

Hardness result on split graphs

Let G be a graph. Create a graph H :

- Create two copies V_1, V_2 of $V(G)$.
- V_1 induces a clique and V_2 a stable set.
- We create an edge $x_1 y_2$ iff $y \in N[x]$.
- We add a vertex in V_2 universal to V_1.
- Add a matching of size k.



Lemma

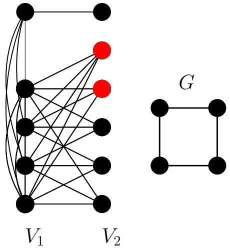
We can transform any independent set of H of size $k+1$ into any other iff there is no dominating set of size k in G.

$\Rightarrow$ A dominating set plus the universal vertex is a frozen independent set.

Hardness result on split graphs

Let G be a graph. Create a graph H :

- Create two copies V_1, V_2 of $V(G)$.
- V_1 induces a clique and V_2 a stable set.
- We create an edge $x_1 y_2$ iff $y \in N[x]$.
- We add a vertex in V_2 universal to V_1.
- Add a matching of size k.



Lemma

We can transform any independent set of H of size $k+1$ into any other iff there is no dominating set of size k in G.

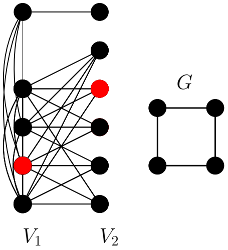
$\Rightarrow$ A dominating set plus the universal vertex is a frozen independent set.

$\Leftarrow$ Move one by one vertices to the top.

Hardness result on split graphs

Let G be a graph. Create a graph H :

- Create two copies V_1, V_2 of $V(G)$.
- V_1 induces a clique and V_2 a stable set.
- We create an edge $x_1 y_2$ iff $y \in N[x]$.
- We add a vertex in V_2 universal to V_1.
- Add a matching of size k.



Lemma

We can transform any independent set of H of size $k+1$ into any other iff there is no dominating set of size k in G.

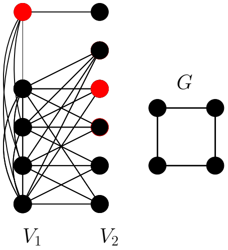
$\Rightarrow$ A dominating set plus the universal vertex is a frozen independent set.

$\Leftarrow$ Move one by one vertices to the top.

Hardness result on split graphs

Let G be a graph. Create a graph H :

- Create two copies V_1, V_2 of $V(G)$.
- V_1 induces a clique and V_2 a stable set.
- We create an edge x_1y_2 iff $y \in N[x]$.
- We add a vertex in V_2 universal to V_1.
- Add a matching of size k.



Lemma

We can transform any independent set of H of size $k+1$ into any other iff there is no dominating set of size k in G.

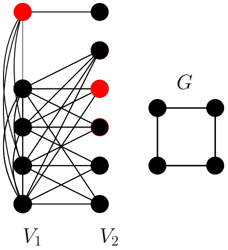
$\Rightarrow$ A dominating set plus the universal vertex is a frozen independent set.

$\Leftarrow$ Move one by one vertices to the top.

Hardness result on split graphs

Let G be a graph. Create a graph H :

- Create two copies V_1, V_2 of $V(G)$.
- V_1 induces a clique and V_2 a stable set.
- We create an edge $x_1 y_2$ iff $y \in N[x]$.
- We add a vertex in V_2 universal to V_1.
- Add a matching of size k.



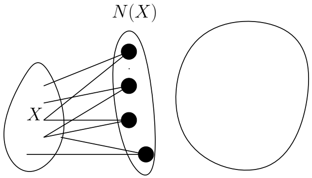
~~Lemma~~

~~We can transform any independent set of H of size $k+1$ into any other iff there is no dominating set of size k in G.~~

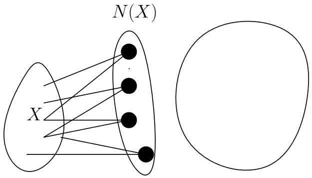
$\Rightarrow$ A dominating set plus the universal vertex is a frozen independent set.

$\Leftarrow$ Move one by one vertices to the top. **Not Always Possible !**

Fix the problem

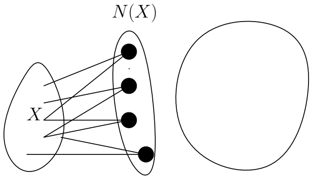


Fix the problem



- y is a private neighbor of x in X if $N(y) \cap X = \{x\}$.
- The set X is j-blocking if $|X| = j$ and no vertex of X has a private neighbor.

Fix the problem

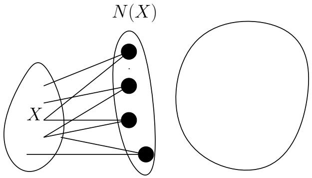


- y is a private neighbor of x in X if $N(y) \cap X = \{x\}$.
- The set X is j-blocking if $|X| = j$ and no vertex of X has a private neighbor.

For every G, we can construct in polynomial time a graph G' :

- with no blocking set of size $j \leq k + 1$, and
- with a dominating set of size at most k iff G has.

Fix the problem



- y is a private neighbor of x in X if $N(y) \cap X = \{x\}$.
- The set X is j-blocking if $|X| = j$ and no vertex of X has a private neighbor.

For every G, we can construct in polynomial time a graph G' :

- with no blocking set of size $j \leq k+1$, and
- with a dominating set of size at most k iff G has.

Lemma

We can transform any independent set of H' of size $k+1$ into any other iff there is no dominating set of size k in G'.

Conclusion

k-Dominating Set is NP-hard and W[2]-hard.

Conclusion

k-Dominating Set is NP-hard and W[2]-hard.

$$\Downarrow$$

k-Dominating Set with no blocking set of size $\leq k+1$
is NP-hard and W[2]-hard.

Conclusion

k-Dominating Set is NP-hard and W[2]-hard.

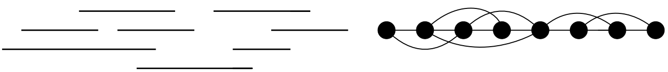
$$\Downarrow$$

k-Dominating Set with no blocking set of size $\leq k+1$
is NP-hard and W[2]-hard.

$$\Downarrow$$

k-TS-Connectivity is co-NP-hard and co-W[2]-hard.

Interval graphs

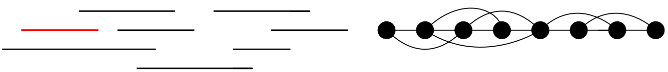


An interval graph is an intersection graph of intervals on the line.

Remark :

A geometric representation can be obtained in polynomial time.

Interval graphs



An interval graph is an intersection graph of intervals on the line.

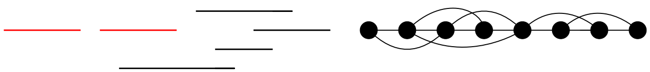
Remark :

A geometric representation can be obtained in polynomial time.

The Leftmost Independent Set (LIS) satisfies :

- The LIS contains the leftmost vertex, i.e. the vertex x with minimum right-end.

Interval graphs



An interval graph is an intersection graph of intervals on the line.

Remark :

A geometric representation can be obtained in polynomial time.

The Leftmost Independent Set (LIS) satisfies :

- The LIS contains the leftmost vertex, i.e. the vertex x with minimum right-end.
- $LIS(G) = x \cup LIS(G[V \setminus N[x]])$.

Interval graphs

An interval graph is an intersection graph of intervals on the line.

Remark :

A geometric representation can be obtained in polynomial time.

The Leftmost Independent Set (LIS) satisfies :

- The LIS contains the leftmost vertex, i.e. the vertex x with minimum right-end.
- $LIS(G) = x \cup LIS(G[V \setminus N[x]])$.

Interval graphs

An interval graph is an intersection graph of intervals on the line.

Remark :

A geometric representation can be obtained in polynomial time.

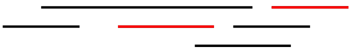
The Leftmost Independent Set (LIS) satisfies :

- The LIS contains the leftmost vertex, i.e. the vertex x with minimum right-end.
- $LIS(G) = x \cup LIS(G[V \setminus N[x]])$.

Informal goal

Decide if an Independent Set of size k can be transformed into the LIS.

First try : Naive Method

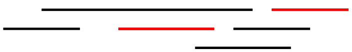


Lemma

I can be transformed into the LIS iff

- The leftmost vertex of x of I can be pushed to the leftmost vertex y of $LIS(G)$.
- $I \setminus x$ can be transformed into $LIS(G) \setminus y$ in $G[V \setminus N[y]]$.

First try : Naive Method



Lemma

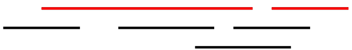
I can be transformed into the LIS iff

- The leftmost vertex of x of I can be pushed to the leftmost vertex y of $LIS(G)$.
- $I \setminus x$ can be transformed into $LIS(G) \setminus y$ in $G[V \setminus N[y]]$.

Naive algorithm :

- Push the leftmost vertex of the independent set to the left and check that it can be transformed into the leftmost vertex.

First try : Naive Method



Lemma

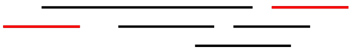
I can be transformed into the LIS iff

- The leftmost vertex of x of I can be pushed to the leftmost vertex y of $LIS(G)$.
- $I \setminus x$ can be transformed into $LIS(G) \setminus y$ in $G[V \setminus N[y]]$.

Naive algorithm :

- Push the leftmost vertex of the independent set to the left and check that it can be transformed into the leftmost vertex.

First try : Naive Method



Lemma

I can be transformed into the LIS iff

- The leftmost vertex of x of I can be pushed to the leftmost vertex y of $LIS(G)$.
- $I \setminus x$ can be transformed into $LIS(G) \setminus y$ in $G[V \setminus N[y]]$.

Naive algorithm :

- Push the leftmost vertex of the independent set to the left and check that it can be transformed into the leftmost vertex.

First try : Naive Method

Lemma

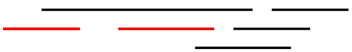
I can be transformed into the LIS iff

- The leftmost vertex of x of I can be pushed to the leftmost vertex y of $LIS(G)$.
- $I \setminus x$ can be transformed into $LIS(G) \setminus y$ in $G[V \setminus N[y]]$.

Naive algorithm :

- Push the leftmost vertex of the independent set to the left and check that it can be transformed into the leftmost vertex.
- Repeat in $V \setminus N[y]$ for the remaining vertices.

First try : Naive Method



Lemma

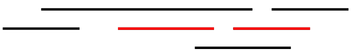
I can be transformed into the LIS iff

- The leftmost vertex of x of I can be pushed to the leftmost vertex y of $LIS(G)$.
- $I \setminus x$ can be transformed into $LIS(G) \setminus y$ in $G[V \setminus N[y]]$.

Naive algorithm :

- Push the leftmost vertex of the independent set to the left and check that it can be transformed into the leftmost vertex.
- Repeat in $V \setminus N[y]$ for the remaining vertices.

First try : Naive Method



Lemma

I can be transformed into the LIS iff

- The leftmost vertex of x of I can be pushed to the leftmost vertex y of $LIS(G)$.
- $I \setminus x$ can be transformed into $LIS(G) \setminus y$ in $G[V \setminus N[y]]$.

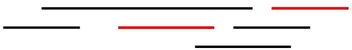
Naive algorithm :

- Push the leftmost vertex of the independent set to the left and check that it can be transformed into the leftmost vertex.
- Repeat in $V \setminus N[y]$ for the remaining vertices.

Problem :

We might need to move vertices to the right to push the leftmost vertex to the left.

First try : Naive Method



Lemma

I can be transformed into the LIS iff

- The leftmost vertex of x of I can be pushed to the leftmost vertex y of $LIS(G)$.
- $I \setminus x$ can be transformed into $LIS(G) \setminus y$ in $G[V \setminus N[y]]$.

Naive algorithm :

- Push the leftmost vertex of the independent set to the left and check that it can be transformed into the leftmost vertex.
- Repeat in $V \setminus N[y]$ for the remaining vertices.

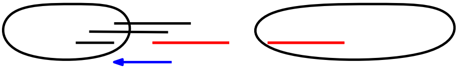
Problem :

We might need to move vertices to the right to push the leftmost vertex to the left.

Informal algorithm

Repeat the following procedure

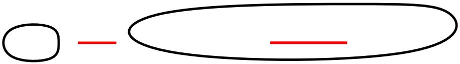
Informal algorithm



Repeat the following procedure

- Push the first vertex to the left.

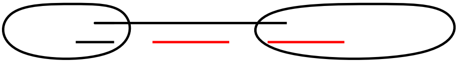
Informal algorithm



Repeat the following procedure

- Push the first vertex to the left.

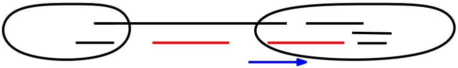
Informal algorithm



Repeat the following procedure

- Push the first vertex to the left.
- Push the independent set minus its first vertex to the right.

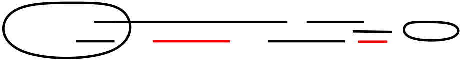
Informal algorithm



Repeat the following procedure

- Push the first vertex to the left.
- Push the independent set minus its first vertex to the right.

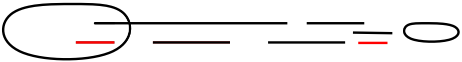
Informal algorithm



Repeat the following procedure

- Push the first vertex to the left.
- Push the independent set minus its first vertex to the right.

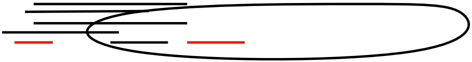
Informal algorithm



Repeat the following procedure

- Push the first vertex to the left.
- Push the independent set minus its first vertex to the right.

Informal algorithm



Repeat the following procedure

- Push the first vertex to the left.
- Push the independent set minus its first vertex to the right.
- If the leftmost vertex is the first vertex of the LIS, apply induction (with $k \leftarrow k - 1$).

Informal algorithm

Repeat the following procedure

- Push the first vertex to the left.
- Push the independent set minus its first vertex to the right.

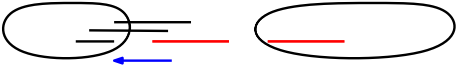
- If the leftmost vertex is the first vertex of the LIS, apply induction (with $k \leftarrow k - 1$).

Informal algorithm

Repeat the following procedure

- Push the first vertex to the left.
- Push the independent set minus its first vertex to the right.
- If the leftmost vertex is the first vertex of the LIS, apply induction (with $k \leftarrow k-1$).
- If no vertices of the independent sets have moved, make a decision.

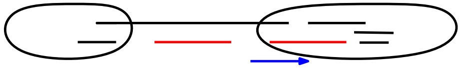
Algorithm for TS-Reachability



We repeat the following procedure on each independent set I and J :

- Push the first vertex to the left.

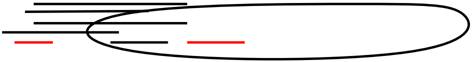
Algorithm for TS-Reachability



We repeat the following procedure on each independent set I and J :

- Push the first vertex to the left.
- Push the independent set minus its first vertex to the right.

Algorithm for TS-Reachability



We repeat the following procedure on each independent set I and J :

- Push the first vertex to the left.
- Push the independent set minus its first vertex to the right.
- If the leftmost vertex is the first vertex of the LIS, apply induction (with $k \leftarrow k - 1$).

Algorithm for TS-Reachability

We repeat the following procedure on each independent set I and J :

- Push the first vertex to the left.
- Push the independent set minus its first vertex to the right.
- If the leftmost vertex is the first vertex of the LIS, apply induction (with $k \leftarrow k - 1$).

Algorithm for TS-Reachability

We repeat the following procedure on each independent set I and J :

- Push the first vertex to the left.
- Push the independent set minus its first vertex to the right.
- If the leftmost vertex is the first vertex of the LIS, apply induction (with $k \leftarrow k-1$).
- If no vertices of the independent sets have moved, compare the first vertices of I' and J' :
  - If they are different : answer NO.
  - If they are the same : delete their first vertices and their neighborhoods and repeat.

Running time

This sequence might not be polynomial...

Running time

This sequence might not be polynomial...

Assume that the first vertex of I is the ith vertex. We might use $\mathcal{O}(i)$ times induction to move the first vertex on the leftmost vertex.

Running time

This sequence might not be polynomial...

Assume that the first vertex of I is the ith vertex. We might use $\mathcal{O}(i)$ times induction to move the first vertex on the leftmost vertex.

$$C(n,k) \approx \max_{i \leq n} \Big(i \cdot C(n-i, k-1) \Big) \approx n^k$$

Running time

This sequence might not be polynomial...

Assume that the first vertex of I is the ith vertex. We might use $\mathcal{O}(i)$ times induction to move the first vertex on the leftmost vertex.

$$C(n,k) \approx \max_{i \leq n} \Big(i \cdot C(n-i, k-1) \Big) \approx n^k$$

$\Rightarrow$ Exponential running time (*a priori*).

Running time

This sequence might not be polynomial...

Assume that the first vertex of I is the ith vertex. We might use $\mathcal{O}(i)$ times induction to move the first vertex on the leftmost vertex.

$$C(n,k) \approx \max_{i \leq n} \Big(i \cdot C(n-i, k-1) \Big) \approx n^k$$

$\Rightarrow$ Exponential running time (*a priori*).

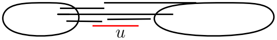
Questions

- Given two independent sets, does there exist a polynomial P such that a minimum transformation between I and J, if it exists, has length at most $P(n)$?
- If yes, is the sequence of this algorithm polynomial ?

Dynamic programming

G_u is the graph at the right of u, i.e. :

- without vertices strictly before u,
- without vertices that intersect u.



Dynamic programming

G_u is the graph at the right of u, i.e. :

- without vertices strictly before u,
- without vertices that intersect u.

Dynamic programming

G_u is the graph at the right of u, i.e. :

- without vertices strictly before u,
- without vertices that intersect u.

Let $I = \{u_1, \ldots, u_k\}$ be an independent set.

Definition

$R(v, i)$: rightmost possible first vertex of an IS we can reach from $\{u_i, \ldots, u_k\}$ in G_v.

Dynamic programming

G_u is the graph at the right of u, i.e. :

- without vertices strictly before u,
- without vertices that intersect u.

Let $I = \{u_1, \ldots, u_k\}$ be an independent set.

Definition

$R(v, i)$: rightmost possible first vertex of an IS we can reach from $\{u_i, \ldots, u_k\}$ in G_v.

Lemma : $R(v, i)$ can be computed in polynomial time.

Dynamic programming

G_u is the graph at the right of u, i.e. :

- without vertices strictly before u,
- without vertices that intersect u.

Let $I = \{u_1, \ldots, u_k\}$ be an independent set.

Definition

$R(v, i)$: rightmost possible first vertex of an IS we can reach from $\{u_i, \ldots, u_k\}$ in G_v.

Lemma : $R(v, i)$ can be computed in polynomial time.

- $R(v, k)$ can be computed in polynomial time (rightmost vertex in the component of u_k in G_v).

Dynamic programming

G_u is the graph at the right of u, i.e. :

- without vertices strictly before u,
- without vertices that intersect u.

Let $I = \{u_1, \dots, u_k\}$ be an independent set.

Definition

$R(v, i)$: rightmost possible first vertex of an IS we can reach from $\{u_i, \dots, u_k\}$ in G_v.

Lemma : $R(v, i)$ can be computed in polynomial time.

- $R(v, k)$ can be computed in polynomial time (rightmost vertex in the component of u_k in G_v).
- Otherwise, repeat :
  - Access to $y = R(u_i, i + 1)$ (induction).
  - z : leftmost vertex we can reach from u_i in $G_v \setminus N(y)$.
  - $u_i \leftarrow z$.

Dynamic programming

G_u is the graph at the right of u, i.e. :

- without vertices strictly before u,
- without vertices that intersect u.

Let $I = \{u_1, \ldots, u_k\}$ be an independent set.

Definition

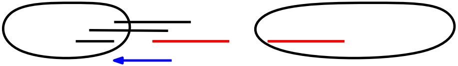
$R(v, i)$: rightmost possible first vertex of an IS we can reach from $\{u_i, \ldots, u_k\}$ in G_v.

Lemma : $R(v, i)$ can be computed in polynomial time.

- $R(v, k)$ can be computed in polynomial time (rightmost vertex in the component of u_k in G_v).
- Otherwise, repeat :
  - Access to $y = R(u_i, i+1)$ (induction).
  - z : leftmost vertex we can reach from u_i in $G_v \setminus N(y)$.
  - $u_i \leftarrow z$.

Complexity : $\mathcal{O}(n \cdot m)$.

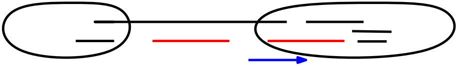
Algorithm for TS-Connectivity



We repeat the following procedure on "any" independent set I :

- Push the first vertex to the left.

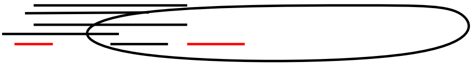
Algorithm for TS-Connectivity



We repeat the following procedure on "any" independent set I :

- Push the first vertex to the left.
- Push the independent set minus its first vertex to the right.

Algorithm for TS-Connectivity



We repeat the following procedure on “any” independent set I :

- Push the first vertex to the left.
- Push the independent set minus its first vertex to the right.

- If the leftmost vertex is the first vertex of the LIS, apply induction (with $k \leftarrow k-1$).

Algorithm for TS-Connectivity

We repeat the following procedure on “any” independent set I :

- Push the first vertex to the left.
- Push the independent set minus its first vertex to the right.
- If the leftmost vertex is the first vertex of the LIS, apply induction (with $k \leftarrow k - 1$).

Algorithm for TS-Connectivity

We repeat the following procedure on “any” independent set I :

- Push the first vertex to the left.
- Push the independent set minus its first vertex to the right.

- If the leftmost vertex is the first vertex of the LIS, apply induction (with $k \leftarrow k - 1$).
- If no vertex of the independent set has moved, answer NO (we cannot reach the LIS).

Algorithm for TS-Connectivity

We repeat the following procedure on "any" independent set I :

- Push the first vertex to the left.
- Push the independent set minus its first vertex to the right.
- If the leftmost vertex is the first vertex of the LIS, apply induction (with $k \leftarrow k - 1$).
- If no vertex of the independent set has moved, answer NO (we cannot reach the LIS).

Computation ?

Using a slightly more complicated dynamic programming algorithm.

Conclusion and open problems

- Complexity of the TS-Reachability on split graphs ? on chordal graphs ?
- Complexity of the TS problems on more general intersection graphs ?
- What about the minimum length sequence ?

Conclusion and open problems

- Complexity of the TS-Reachability on split graphs ? on chordal graphs ?
- Complexity of the TS problems on more general intersection graphs ?
- What about the minimum length sequence ?

Thanks for your attention !