Sliding Token on Bipartite Permutation Graphs

Eli Fox-Epstein[1] Duc A. Hoang[2]
Yota Otachi[2] Ryuhei Uehara[2]

[1]Brown University, USA

[2]JAIST, Japan

Reconfiguration Problems

[Flake & Baum 2002]

[Romanishin, Rus, Gilpin 2013]

[Ito, Demaine, Harvey, Papadimitriou, Sideri, Uehara, Uno 2008]


NIKKEI
東朋テクノロジー
TOHO

Reconfiguration Problems

- Start with some problem with solutions

Reconfiguration Problems

- Start with some problem with solutions (e.g. Rush Hour)

Reconfiguration Problems

- Start with some problem with solutions (e.g. Rush Hour)
- Define legal transformations between solutions

Reconfiguration Problems

- Start with some problem with solutions (e.g. Rush Hour)
- Define legal transformations between solutions
  (legal if solutions differ by sliding one car)

Reconfiguration Problems

- Start with some problem with solutions (e.g. Rush Hour)
- Define legal transformations between solutions
  (legal if solutions differ by sliding one car)
- Question: is there a sequence of transformations between two given solutions?

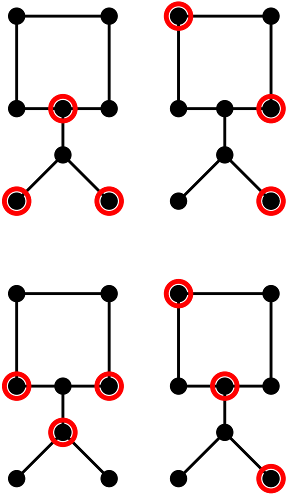
Reconfiguration Problems

- Start with some problem with solutions (e.g. Rush Hour)
- Define legal transformations between solutions
  (legal if solutions differ by sliding one car)
- Question: is there a sequence of transformations between two given solutions? (PSPACE-complete for Rush Hour)

Sliding Token

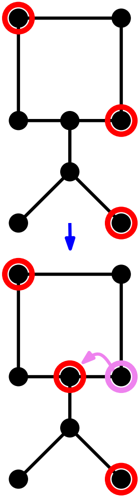
Sliding Token:
a natural, pure problem in Combinatorial Reconfiguration

Sliding Token



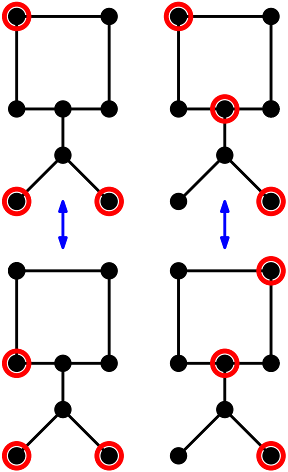
- Classic optimization problem: **Independent Set**

Sliding Token



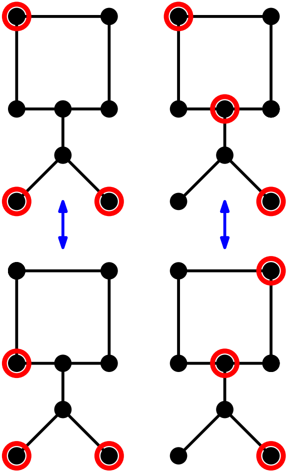
- Classic optimization problem: **Independent Set**
- Reconfiguration moves: “slide” a “token” to a neighbor

Sliding Token



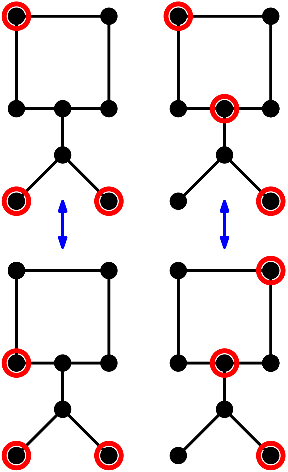
- Classic optimization problem: **Independent Set**
- Reconfiguration moves: “slide” a “token” to a neighbor
- Induces a “reconfiguration graph”

Sliding Token



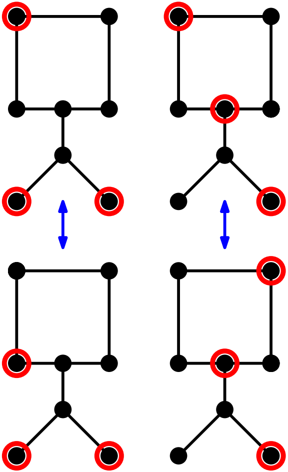
- Classic optimization problem: **Independent Set**
- Reconfiguration moves: "slide" a "token" to a neighbor
- Induces a "reconfiguration graph"
  - Nodes: independent sets

Sliding Token



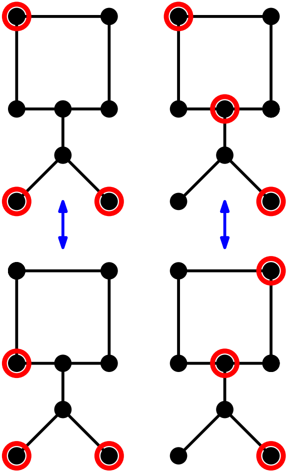
- Classic optimization problem: **Independent Set**
- Reconfiguration moves: “slide” a “token” to a neighbor
- Induces a “reconfiguration graph”
  - Nodes: independent sets
  - Adjacency: one reconfiguration move

Sliding Token



- Classic optimization problem: **Independent Set**
- Reconfiguration moves: "slide" a "token" to a neighbor
- Induces a "reconfiguration graph"
  - Nodes: independent sets
  - Adjacency: one reconfiguration move
  - Notation: $[A]$ is A's connected component

Sliding Token



- Classic optimization problem: **Independent Set**
- Reconfiguration moves: "slide" a "token" to a neighbor
- Induces a "reconfiguration graph"
  - Nodes: independent sets
  - Adjacency: one reconfiguration move
  - Notation: $[A]$ is A's connected component
  - Ask: $B \in [A]$?

A Brief Overview of SLIDING TOKEN's Complexity

- PSPACE-complete on general, AT-free, planar, perfect, and bounded treewidth graphs [Ito, Demaine, Harvey, Papadimitriou, Sideri, Uehara, Uno 2008], [Wrochna 2014]
- Polytime on proper interval graphs, claw-free graphs, forests, cographs [Bonsma, Kamiński, Wronchna 2014], [Demaine, Demaine, F., Hoang, Ito, Ono, Otachi, Uehara, Yamada 2014], [Kamiński, Medvedev, Milanic 2010]
- **??? on bipartite graphs**

A Brief Overview of SLIDING TOKEN's Complexity

- PSPACE-complete on general, AT-free, planar, perfect, and bounded treewidth graphs [Ito, Demaine, Harvey, Papadimitriou, Sideri, Uehara, Uno 2008], [Wrochna 2014]
- Polytime on proper interval graphs, claw-free graphs, forests, cographs [Bonsma, Kamiński, Wronchna 2014], [Demaine, Demaine, F., Hoang, Ito, Ono, Otachi, Uehara, Yamada 2014], [Kamiński, Medvedev, Milanic 2010]
- **??? on bipartite graphs**
  - We give an efficient algorithm on a subclass of bipartite graphs.

Main Result

Algorithm for SLIDING TOKEN on **bipartite permutation** graphs.

Main Result

Algorithm for Sliding Token on **bipartite permutation** graphs.

Given graph G, independent sets A and B,
finds a reconfiguration sequence from A to B
or reports that none exists.

Bipartite Permutation Graphs

$$\{\text{bipartite permutation graphs}\} = \{\text{bipartite graphs}\} \cap \{\text{permutation graphs}\}$$

Bipartite Permutation Graphs

$$\begin{aligned}
&\{\text{bipartite permutation graphs}\} \\
&\quad = \{\text{bipartite graphs}\} \cap \{\text{permutation graphs}\} \\
&\quad = \{\text{bipartite graphs}\} \cap \{\text{tolerance graphs}\}
\end{aligned}$$

Bipartite Permutation Graphs

$$
\begin{aligned}
&\{\text{bipartite permutation graphs}\} \\
&\quad = \{\text{bipartite graphs}\} \cap \{\text{permutation graphs}\} \\
&\quad = \{\text{bipartite graphs}\} \cap \{\text{tolerance graphs}\} \\
&\quad = \{\text{bipartite graphs}\} \cap \{\text{AT-free graphs}\}
\end{aligned}
$$

Bipartite Permutation Graphs

{bipartite permutation graphs}

$= \{$bipartite graphs$\} \cap \{$permutation graphs$\}$

$= \{$bipartite graphs$\} \cap \{$tolerance graphs$\}$

$= \{$bipartite graphs$\} \cap \{$AT-free graphs$\}$

(Sliding Token is PSPACE-hard on AT-free graphs)

Bipartite Permutation Graphs

$$
\begin{aligned}
&\{\text{bipartite permutation graphs}\}\\
&\quad = \{\text{bipartite graphs}\} \cap \{\text{permutation graphs}\}\\
&\quad = \{\text{bipartite graphs}\} \cap \{\text{tolerance graphs}\}\\
&\quad = \{\text{bipartite graphs}\} \cap \{\text{AT-free graphs}\}\\
&\qquad\quad (\textsc{Sliding Token} \text{ is PSPACE-hard on AT-free graphs})\\
&\quad = \dots
\end{aligned}
$$

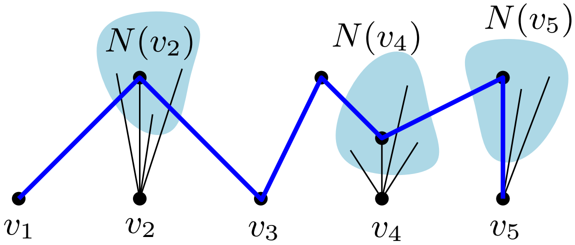
Bipartite permutation graph iff

vertices can be ordered $v_1, v_2, \ldots, v_n$

such that $\forall i \leq j \leq k$,

all paths from v_i to v_k include a vertex of $N[v_j]$.

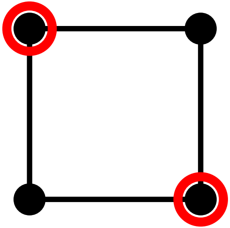
Bipartite permutation graph iff
vertices can be ordered $v_1, v_2, \ldots, v_n$
such that $\forall i \leq j \leq k$,
all paths from v_i to v_k include a vertex of $N[v_j]$.



Algorithmic Tools

Algorithmic Tools

1. Wiggling: greedily moving each token to a neighbor and then eventually back



Algorithmic Tools

1. Wiggling: greedily moving each token to a neighbor and then eventually back
2. Rigid tokens: $R(G, A) = \bigcap[A]$

Algorithmic Tools

1. Wiggling: greedily moving each token to a neighbor and then eventually back
2. Rigid tokens: $R(G, A) = \bigcap [A]$
   - Wiggling finds rigid tokens efficiently

Algorithmic Tools

1. Wiggling: greedily moving each token to a neighbor and then eventually back
2. Rigid tokens: $R(G, A) = \bigcap[A]$
   - Wiggling finds rigid tokens efficiently
   - If $R(G, A) = R(G, B)$, simplify by deleting $N[R(G, A)]$ from graph. Otherwise, A does not reconfigure to B

Algorithmic Tools

1. Wiggling: greedily moving each token to a neighbor and then eventually back
2. Rigid tokens: $R(G, A) = \bigcap[A]$
   - Wiggling finds rigid tokens efficiently
   - If $R(G, A) = R(G, B)$, simplify by deleting $N[R(G, A)]$ from graph. Otherwise, A does not reconfigure to B
   - Now, wlog no rigid tokens

Algorithmic Tools

1. Wiggling: greedily moving each token to a neighbor and then eventually back
2. Rigid tokens: $R(G, A) = \bigcap[A]$
   - Wiggling finds rigid tokens efficiently
   - If $R(G, A) = R(G, B)$, simplify by deleting $N[R(G, A)]$ from graph. Otherwise, A does not reconfigure to B
   - Now, wlog no rigid tokens
3. Targeting: putting a token on a specific vertex

Algorithmic Tools

1. Wiggling: greedily moving each token to a neighbor and then eventually back
2. Rigid tokens: $R(G, A) = \bigcap[A]$
   - Wiggling finds rigid tokens efficiently
   - If $R(G, A) = R(G, B)$, simplify by deleting $N[R(G, A)]$ from graph. Otherwise, A does not reconfigure to B
   - Now, wlog no rigid tokens
3. Targeting: putting a token on a specific vertex
4. **Canonical representatives** of reconfiguration graph's connected components

Canonical Representatives

- Canonical representative A_+ for connected component $[A]$: lexicographically minimum independent set

Canonical Representatives

- Canonical representative A_+ for connected component $[A]$: lexicographically minimum independent set
  - (Minimize max index of vertex in set, then second max, etc.)

Canonical Representatives

- Canonical representative A_+ for connected component $[A]$: lexicographically minimum independent set
  - (Minimize max index of vertex in set, then second max, etc.)
- $B \in [A]$ iff $A_+ = B_+$.

Finding A_+: use DP

Two ideas:

Finding A_+: use DP

Two ideas:

(a) A_+ contains v_1 or vertex of least index in $N(v_1)$; call this u

Finding A_+: use DP

Two ideas:

(a) A_+ contains v_1 or vertex of least index in $N(v_1)$; call this u

(b) Can maneuver a token to u and delete neighborhood without making new rigid tokens

Finding A_+: use DP

Two ideas:

(a) A_+ contains v_1 or vertex of least index in $N(v_1)$; call this u

(b) Can maneuver a token to u and delete neighborhood without making new rigid tokens

- Formally: $R(G \setminus N[u], I \setminus \{u\}) = \emptyset$ where $u \in I \in [A]$

Finding A_+: use DP

Two ideas:

(a) A_+ contains v_1 or vertex of least index in $N(v_1)$; call this u

(b) Can maneuver a token to u and delete neighborhood without making new rigid tokens

- Formally: $R(G \setminus N[u], I \setminus \{u\}) = \emptyset$ where $u \in I \in [A]$

Finding A_+: use DP

Two ideas:

(a) A_+ contains v_1 or vertex of least index in $N(v_1)$; call this u

(b) Can maneuver a token to u and delete neighborhood without making new rigid tokens

- Formally: $R(G \setminus N[u], I \setminus \{u\}) = \emptyset$ where $u \in I \in [A]$

Strategy:

- DP guesses least index of tokens in A_+ and uses Targeting to put a token there

Finding A_+: use DP

Two ideas:

(a) A_+ contains v_1 or vertex of least index in $N(v_1)$; call this u

(b) Can maneuver a token to u and delete neighborhood without making new rigid tokens
  - Formally: $R(G \setminus N[u], I \setminus \{u\}) = \emptyset$ where $u \in I \in [A]$

Strategy:

- DP guesses least index of tokens in A_+ and uses Targeting to put a token there
- That token will never move again

Finding A_+: use DP

Two ideas:

(a) A_+ contains v_1 or vertex of least index in $N(v_1)$; call this u

(b) Can maneuver a token to u and delete neighborhood without making new rigid tokens

- Formally: $R(G \setminus N[u], I \setminus \{u\}) = \emptyset$ where $u \in I \in [A]$

Strategy:

- DP guesses least index of tokens in A_+ and uses Targeting to put a token there
- That token will never move again
- By (b), this does not cause any tokens to go rigid

Finding A_+: use DP

Two ideas:

(a) A_+ contains v_1 or vertex of least index in $N(v_1)$; call this u

(b) Can maneuver a token to u and delete neighborhood without making new rigid tokens

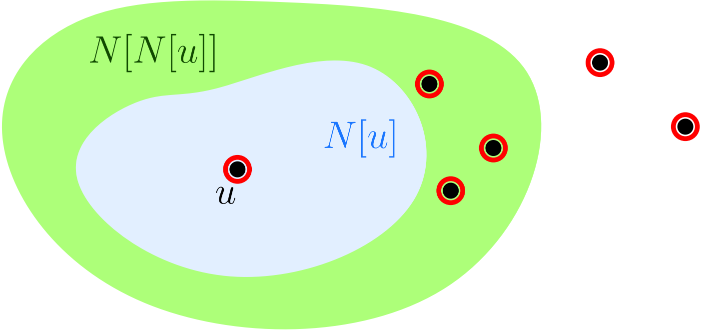
- Formally: $R(G \setminus N[u], I \setminus \{u\}) = \emptyset$ where $u \in I \in [A]$

Strategy:

- DP guesses least index of tokens in A_+ and uses Targeting to put a token there
- That token will never move again
- By (b), this does not cause any tokens to go rigid
- Repeat, pretending we deleted the token and neighborhood

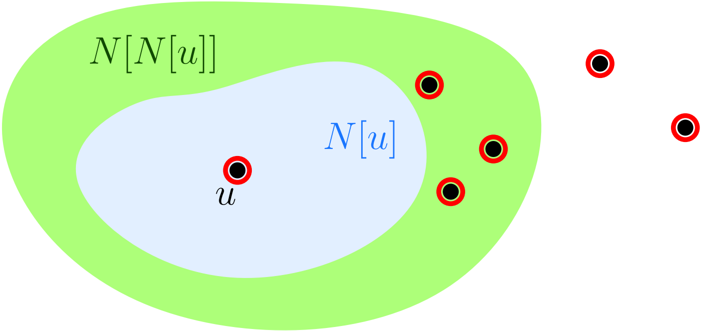
Proof sketch of idea (b): $R(G \setminus N[u], I \setminus \{u\}) = \emptyset$

u: vertex of least index in A_+



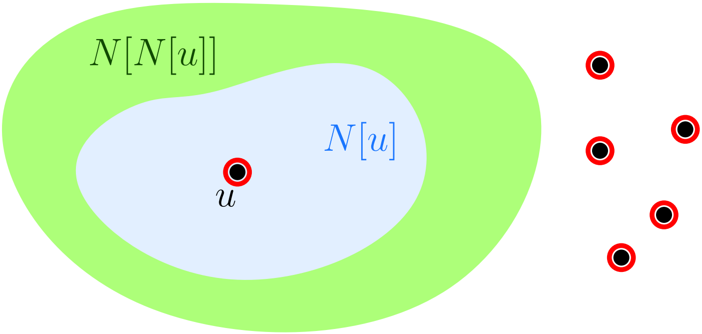
Proof sketch of idea (b): $R(G \setminus N[u], I \setminus \{u\}) = \emptyset$

First, push other tokens away from u (extreme case analysis)

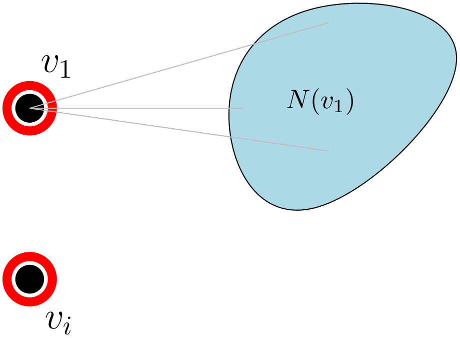


Proof sketch of idea (b): $R(G \setminus N[u], I \setminus \{u\}) = \emptyset$

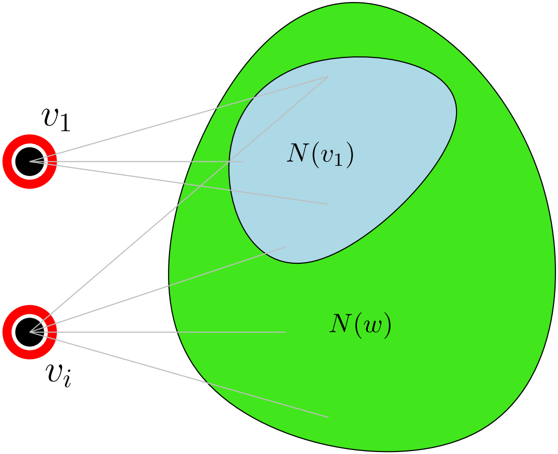
First, push other tokens away from u (extreme case analysis)



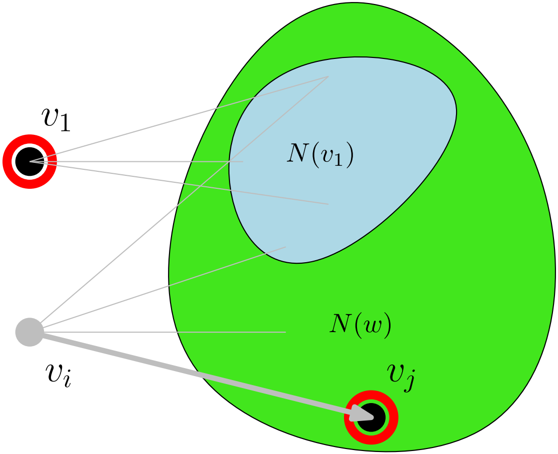
Case: $v_1, v_i \in I$ with $i <$ min index of neighbor of v_1



Case: $v_1, v_i \in I$ with $i <$ min index of neighbor of v_1

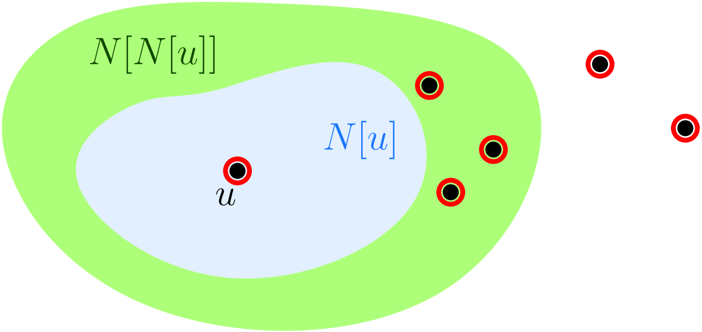


Case: $v_1, v_i \in I$ with $i <$ min index of neighbor of v_1



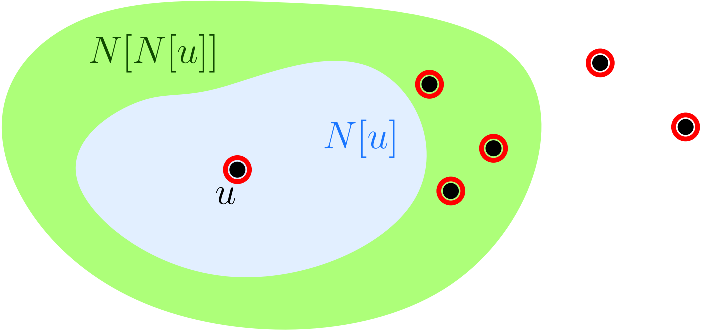
Idea (b) proof:

u: vertex of least index in I_+



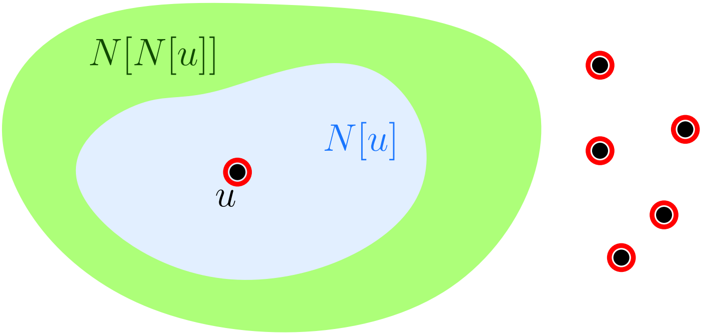
Idea (b) proof:

First, push other tokens away from u (extreme case analysis)



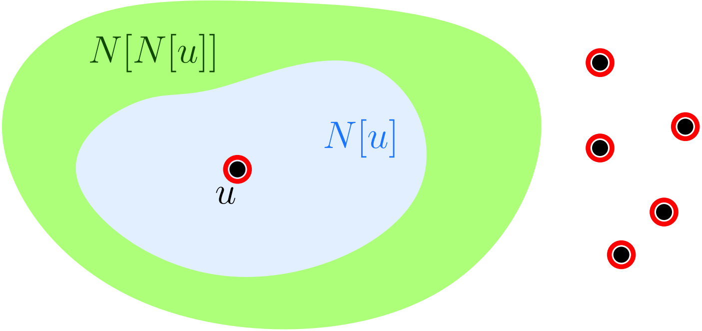
Idea (b) proof:

First, push other tokens away from u (extreme case analysis)



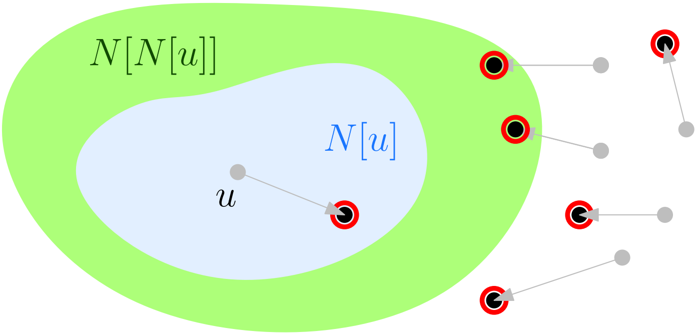
Idea (b) proof:

"Wiggle" everything



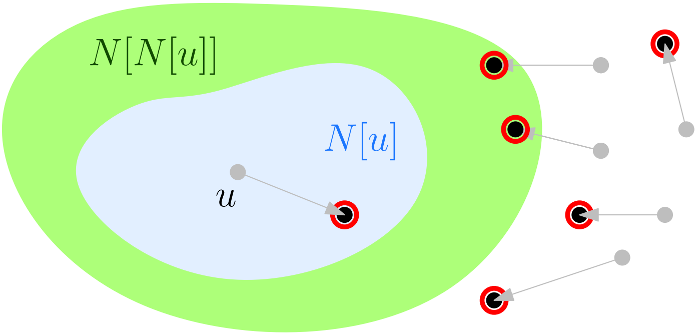
Idea (b) proof:

"Wiggle" everything



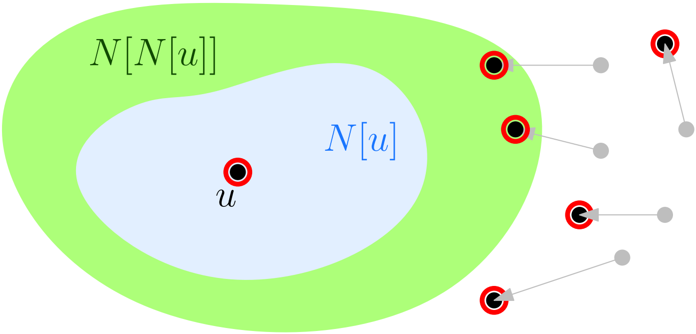
Idea (b) proof:

Now edit sequence so token stays on u



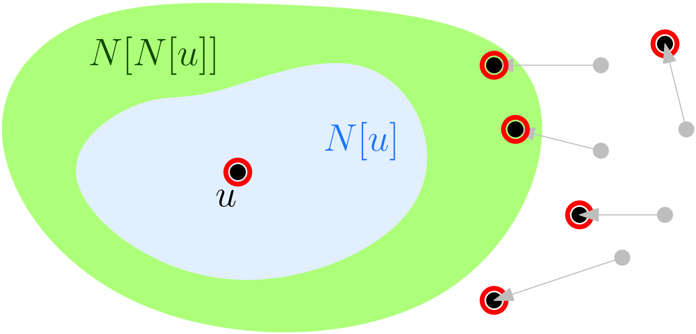
Idea (b) proof:

Now edit sequence so token stays on u



Idea (b) proof:

This sequence witnesses that nothing is rigid after deleting $N[u]$



Dynamic program essentially tracks how to pack most tokens onto vertices v_1 through v_i for all i.

Dynamic program essentially tracks how to pack most tokens onto vertices v_1 through v_i for all i.

For entry corresponding to v_i, the DP

- guesses greatest $j < i$ containing a token ($O(n)$ guesses)

Dynamic program essentially tracks how to pack most tokens onto vertices v_1 through v_i for all i.

For entry corresponding to v_i, the DP

- guesses greatest $j < i$ containing a token ($O(n)$ guesses)
- places a token on that vertex ($O(n)$ time);

Dynamic program essentially tracks how to pack most tokens onto vertices v_1 through v_i for all i.

For entry corresponding to v_i, the DP

- guesses greatest $j < i$ containing a token ($O(n)$ guesses)
- places a token on that vertex ($O(n)$ time);
- deletes the neighborhood; and

Dynamic program essentially tracks how to pack most tokens onto vertices v_1 through v_i for all i.

For entry corresponding to v_i, the DP

- guesses greatest $j < i$ containing a token ($O(n)$ guesses)
- places a token on that vertex ($O(n)$ time);
- deletes the neighborhood; and
- checks for rigidity ($O(n)$ time).

Dynamic program essentially tracks how to pack most tokens onto vertices v_1 through v_i for all i.

For entry corresponding to v_i, the DP

- guesses greatest $j < i$ containing a token ($O(n)$ guesses)
- places a token on that vertex ($O(n)$ time);
- deletes the neighborhood; and
- checks for rigidity ($O(n)$ time).

Dynamic program essentially tracks how to pack most tokens onto vertices v_1 through v_i for all i.

For entry corresponding to v_i, the DP

- guesses greatest $j < i$ containing a token ($O(n)$ guesses)
- places a token on that vertex ($O(n)$ time);
- deletes the neighborhood; and
- checks for rigidity ($O(n)$ time).

Overall, $O(n^3)$ time.

Question: **is there hope for generalization?**

Question: **is there hope for generalization?**

- Wiggling, Targeting apply to bipartite graphs.

Question: **is there hope for generalization?**

- Wiggling, Targeting apply to bipartite graphs.
- Canonical representatives seem hard to generalize: permutation graphs have nice linear structure.

Question: **is there hope for generalization?**

- Wiggling, Targeting apply to bipartite graphs.
- Canonical representatives seem hard to generalize: permutation graphs have nice linear structure.
- Cannot naively put a token on some vertex and delete the neighborhood

Thanks

Bibliography I

H. Bandelt and H. M. Mulder.
Distance-hereditary graphs.
Journal of Combinatorial Theory, Series B, 41(2):182–208, 1986.

H. L. Bodlaender.
A Partial *k*-Arboretum of Graphs with Bounded Treewidth.
Theor. Comput. Sci., 209(1-2):1–45, 1998.

M. Bonamy and N. Bousquet.
Recoloring bounded treewidth graphs.
Electronic Notes in Discrete Mathematics, 44:257–262, 2013.

M. Bonamy and N. Bousquet.
Recoloring graphs via tree decompositions.
CoRR, abs/1403.6386, 2014.

M. Bonamy and N. Bousquet.
Reconfiguring independent sets in cographs.
CoRR, abs/1406.1433, 2014.

M. Bonamy, M. Johnson, I. Lignos, V. Patel, and D. Paulusma.
On the diameter of reconfiguration graphs for vertex colourings.
Electronic Notes in Discrete Mathematics, 38:161–166, 2011.

Bibliography II

M. Bonamy, M. Johnson, I. Lignos, V. Patel, and D. Paulusma.
Reconfiguration graphs for vertex colourings of chordal and chordal bipartite graphs.
Journal of Combinatorial Optimization, 27(1):132–143, 2014.

P. Bonsma.
Shortest Path Reconfiguration is PSPACE-hard.
CoRR, abs/1009.3217, 2010.

P. Bonsma.
The complexity of rerouting shortest paths.
Theoretical computer science, 510:1–12, 2013.

P. Bonsma.
Independent set reconfiguration in cographs.
CoRR, abs/1402.1587, 2014.

P. Bonsma and L. Cereceda.
Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances.
In *Mathematical Foundations of Computer Science 2007*, volume 4708 of *Lecture Notes in Computer Science*, pages 738–749. Springer Berlin Heidelberg, 2007.

P. S. Bonsma, M. Kaminski, and M. Wrochna.
Reconfiguring independent sets in claw-free graphs.
In *Algorithm Theory - SWAT 2014*, pages 86–97, 2014.

Bibliography III

C. Calabro.
The exponential complexity of satisfiability problems.
PhD thesis, UC San Diego, 2009.

B. Courcelle.
The monadic second-order logic of graphs. I. Recognizable sets of finite graphs.
Information and computation, 85(1):12–75, 1990.

E. D. Demaine, M. L. Demaine, E. Fox-Epstein, D. A. Hoang, T. Ito, H. Ono, Y. Otachi, R. Uehara, and T. Yamada.
Polynomial-time algorithm for sliding tokens on trees.
In *Algorithms and Computation*, volume 8889 of *Lecture Notes in Computer Science*, pages 389–400. Springer International Publishing, 2014.

P. Gopalan, P. G. Kolaitis, E. Maneva, and C. H. Papadimitriou.
The Connectivity of Boolean Satisfiability: Computational and Structural Dichotomies.
SIAM Journal on Computing, 38(6):2330–2355, 2009.

R. A. Hearn.
Games, Puzzles, and Computation.
PhD thesis, Cambridge, MA, USA, 2006.

Bibliography IV

R. A. Hearn and E. D. Demaine.
PSPACE-completeness of Sliding-block Puzzles and Other Problems Through the Nondeterministic Constraint Logic Model of Computation.
Theor. Comput. Sci., 343(1-2):72–96, October 2005.

T. Ito and E. D. Demaine.
Approximability of the subset sum reconfiguration problem.
Journal of Combinatorial Optimization, 28(3):639–654, 2014.

T. Ito, E. D. Demaine, N. J. A. Harvey, C. H. Papadimitriou, M. Sideri, R. Uehara, and Y. Uno.
On the complexity of reconfiguration problems.
In *Algorithms and Computation*, volume 5369 of *Lecture Notes in Computer Science*, pages 28–39. Springer Berlin Heidelberg, 2008.

T. Ito, M. Kamiński, and E. D. Demaine.
Reconfiguration of list edge-colorings in a graph.
In *Algorithms and Data Structures*, volume 5664 of *Lecture Notes in Computer Science*, pages 375–386. Springer Berlin Heidelberg, 2009.

T. Ito, M. Kamiński, H. Ono, A. Suzuki, R. Uehara, and K. Yamanaka.
On the parameterized complexity for token jumping on graphs.
In *Theory and Applications of Models of Computation*, volume 8402 of *Lecture Notes in Computer Science*, pages 341–351. Springer International Publishing, 2014.

Bibliography V

Takehiro Ito, Kazuto Kawamura, Hirotaka Ono, and Xiao Zhou.
Reconfiguration of list l(2,1)-labelings in a graph.
In *Algorithms and Computation*, volume 7676 of *Lecture Notes in Computer Science*, pages 34–43. Springer Berlin Heidelberg, 2012.

Takehiro Ito, Kazuto Kawamura, and Xiao Zhou.
An improved sufficient condition for reconfiguration of list edge-colorings in a tree.
In *Theory and Applications of Models of Computation*, volume 6648 of *Lecture Notes in Computer Science*, pages 94–105. Springer Berlin Heidelberg, 2011.

M. Kamiński, P. Medvedev, and M. Milani.
Complexity of independent set reconfigurability problems.
Theor. Comput. Sci., 439:9–15, June 2012.

M. Kamiński, P. Medvedev, and M. Milanic.
Shortest paths between shortest paths and independent sets.
CoRR, abs/1008.4563, 2010.

K. Makino, S. Tamaki, and M. Yamamoto.
An exact algorithm for the Boolean connectivity problem for k-CNF.
In *Theory and Applications of Satisfiability Testing–SAT 2010*, pages 172–180. Springer, 2010.

Bibliography VI

M. Mézard, G. Parisi, and R. Zecchina.
Analytic and algorithmic solution of random satisfiability problems.
Science, 297(5582):812–815, 2002.

A. E. Mouawad, N. Nishimura, V. Raman, N. Simjour, and A. Suzuki.
On the parameterized complexity of reconfiguration problems.
In *Parameterized and Exact Computation*, pages 281–294. Springer, 2013.

A. E. Mouawad, N. Nishimura, V. Raman, and M. Wrochna.
Reconfiguration over tree decompositions.
In *Parameterized and Exact Computation*, volume 8894 of *Lecture Notes in Computer Science*, pages 246–257. Springer International Publishing, 2014.

T. J. Schaefer.
The complexity of satisfiability problems.
In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226. ACM, 1978.

J. Spinrad, A. Brandstädt, and L. Stewart.
Bipartite permutation graphs.
Discrete Applied Mathematics, 18(3):279–292, 1987.

A. P. Sprague.
Recognition of bipartite permutation graphs.
Congressus Numerantium, 62:151–161, 1995.

Bibliography VII

W. van Wezel and R. Jorna.
Cognition, tasks and planning: supporting the planning of shunting operations at the netherlands railways.
Cognition, Technology & Work, 11(2):165–176, 2009.

M. Wrochna.
Reconfiguration in bounded bandwidth and treedepth.
CoRR, abs/1405.0847, 2014.