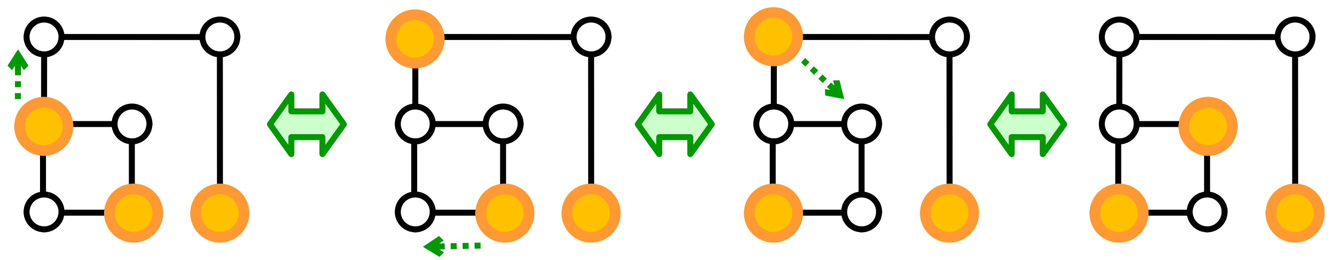
Invitation to Combinatorial Reconfiguration



Takehiro ITO

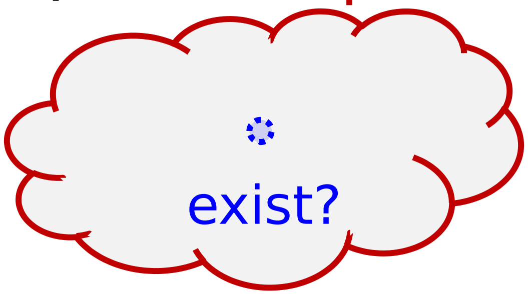
Tohoku University, Japan

SIAM DM 2016 --- June 7, 2016

Combinatorial Reconfiguration

asks the “reachability”/“connectivity” of the solution space.

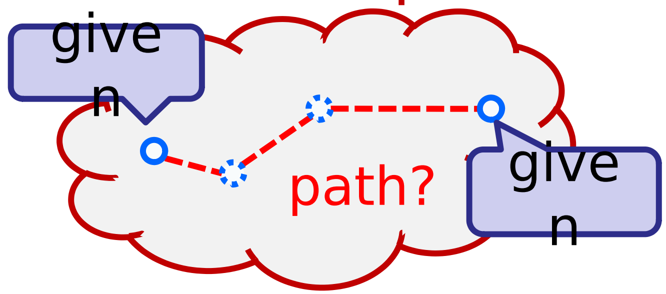
solution space



Search Problem

asks the **existence** of a feasible solution.

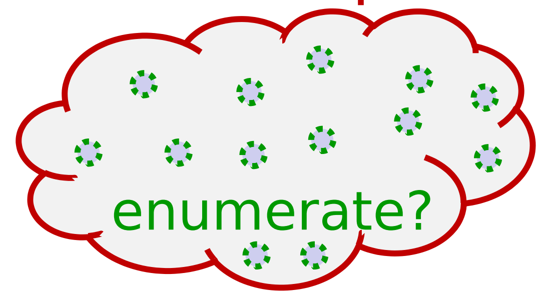
solution space



Reconfiguration Problem

asks the **reachability** between two given feasible solutions

solution space



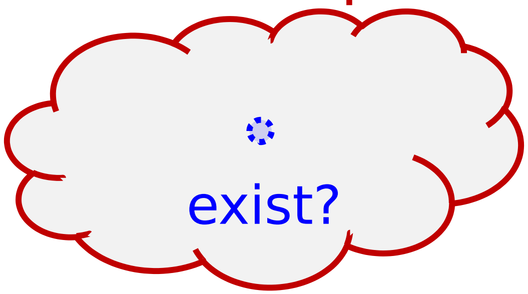
Enumeration Problem

asks to **output ALL** feasible solutions

The concept of reconfiguration problems is located “**between**” standard search problems and enumeration problems.

Search problem

solution space

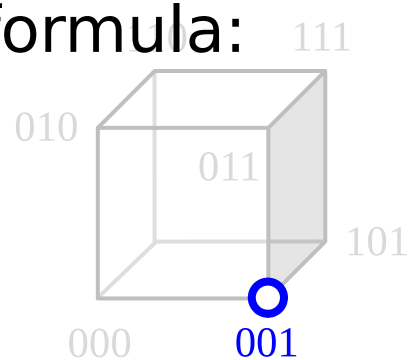


Search Problem asks the **existence** of a feasible solution.

Check if there **exists at least one** feasible solution (i.e., satisfiable truth assignment of)
from candidates of solutions for
variables.

$$f = (x \vee \overline{y}) \wedge (\overline{x} \vee y \vee z) \wedge (\overline{y} \vee \overline{z})$$

ex) SAT formula:

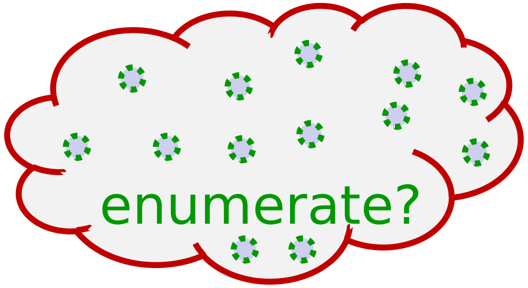


Check if there **exists at least one** feasible solution (i.e., satisfiable truth assignment of f) from 2^n candidates of solutions for n variables.

Enumeration problem

Enumeration Problem asks to output **ALL** feasible solutions

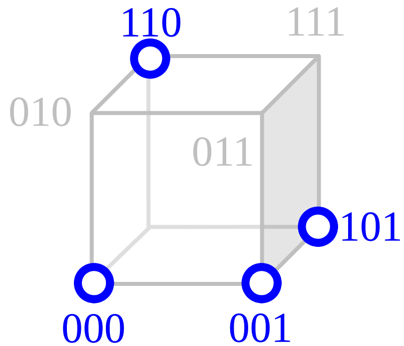
solution space



ex) SAT formula:

$$f = (x \vee \overline{y}) \wedge (\overline{x} \vee y \vee z) \wedge (\overline{y} \vee \overline{z})$$

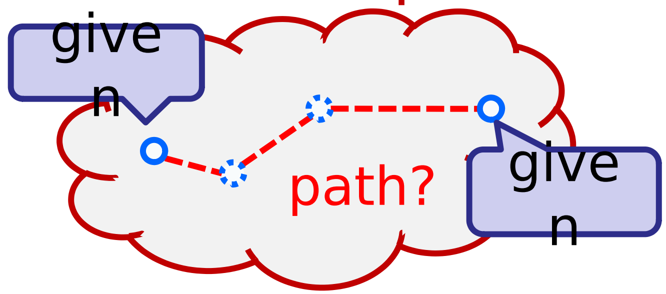
output **all** feasible solutions from candidates of solutions for variables.



Combinatorial Reconfiguration

asks the "reachability"/"connectivity" of the solution space.

solution space



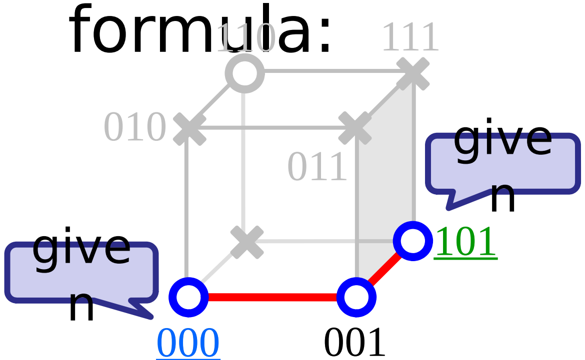
introduce an **adjacency relation** on feasible solutions

two feasible solutions are **given as an input**

Reconfiguration Problem
asks the **reachability** between two given feasible solutions

Hamming distance one
(i.e., flip of a single variable)

ex) SAT formula:



$$f = (x \vee \bar{y}) \wedge (\bar{x} \vee y \vee z) \wedge (\bar{y} \vee \bar{z})$$

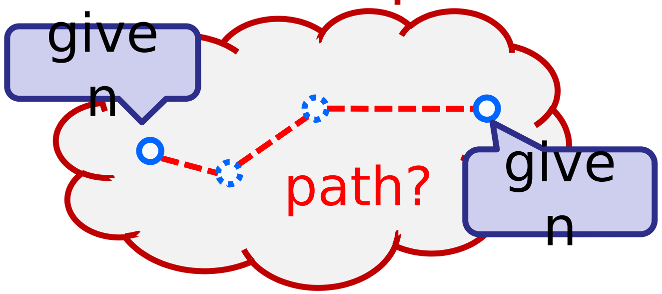
satisfiable!

& **&** $(x, y, z) = (0,0,0)$
$= (1,0,1)$

Find a **sequence of adjacent feasible** solutions among candidates of solutions for variables.
(We do NOT know the feasibility of the other candidates.)

Combinatorial Reconfiguration

solution space



Reconfiguration
Problem
asks the
reachability
between two given
feasible solutions

Reconfiguration is a **decision** problem:

- simply output Yes/No
- actual reconfiguration sequence is not required

Indeed, there are examples such that a **shortest** reconfiguration sequence requires **super-polynomial** length!

Output the answer **without** constructing the solution space

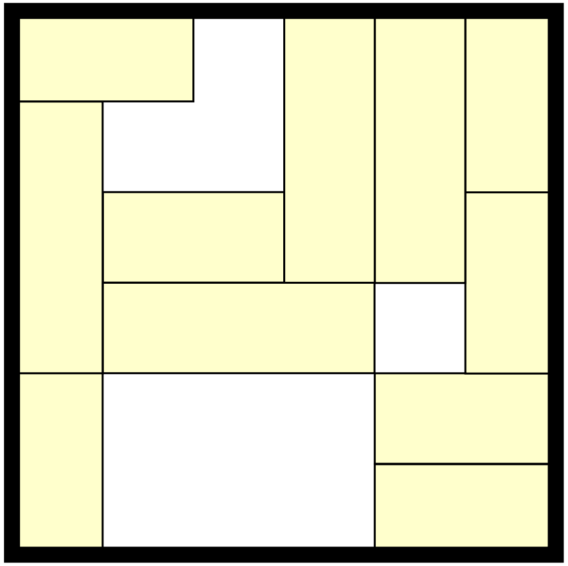
Challenge!!

- solution space can be exponential size w.r.t. the input size
- but, evaluate the running time of algorithm w.r.t. the input size!

Motivations

[Puzzles]

- Sliding block puzzle
- Rubik cube
- 15 puzzle

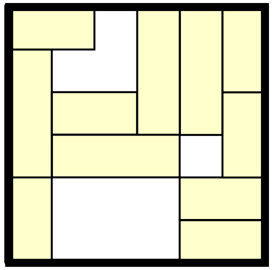


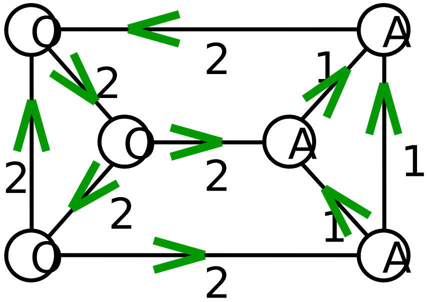
Sliding block puzzle

Motivations

[Puzzles]

- Sliding block puzzle
- Rubik cube
- 15 puzzle

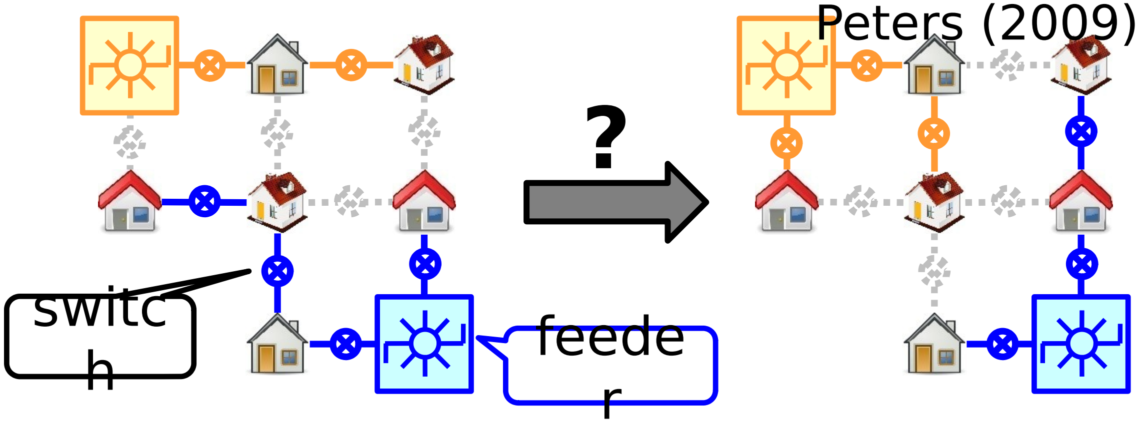




R.A. Hearn, E.D. Demaine. Games, Puzzles, and Computation. A K Peters (2009)

[Power-supply network]

by operating switches, reconfigurable without causing any blackout?



[The Potts model in physics]

= Graph coloring reconfiguration (under Kempe change rule)

In this mini-symposia:

2) C. Feghali "Kempe equivalence of colourings of graphs"

3) J. Salas "Kempe reconfiguration and Potts antiferromagne

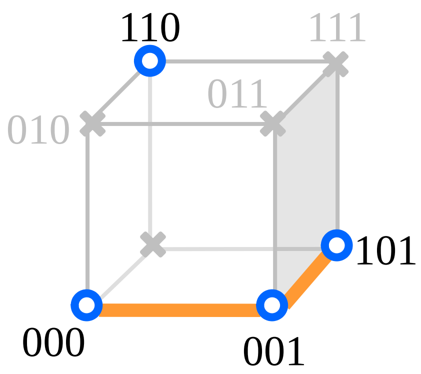
Adjacency relation

Reconfiguration problem = feasible solutions for a search problem instance + adjacency relation

defined by
- application
- most elementary change to solution

(But, there is no clearly-stated rule.)

Example:



Solution space for SAT formula

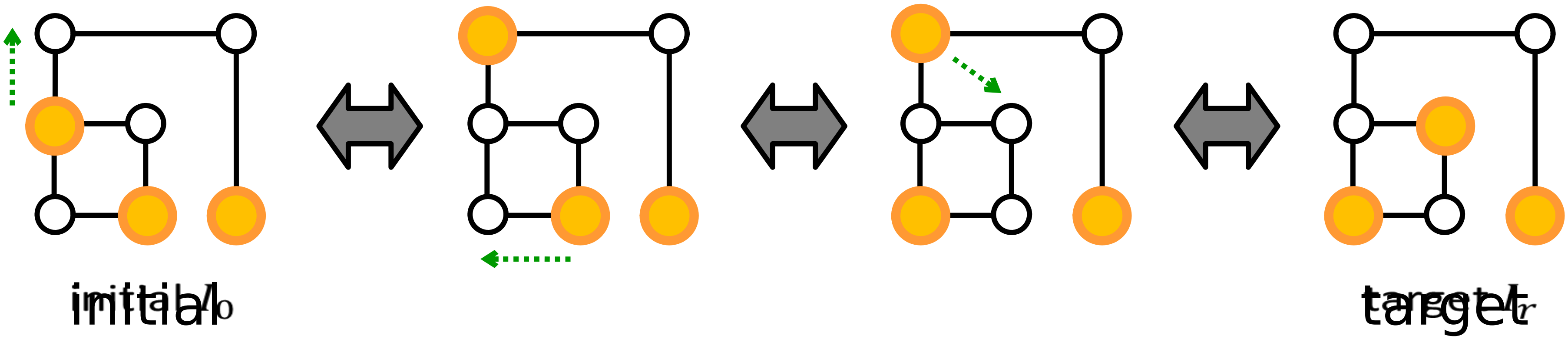
$$f = (x \vee \bar{y}) \wedge (\bar{x} \vee y \vee z) \wedge (\bar{y} \vee \bar{z})$$

[**SAT** reconfiguration]
- feasible solutions: satisfiable truth assignments of f
- adjacency relation: flip of a single variable (Hamming distance one)

Independent set reconfiguration

[**Independent set** reconfiguration (**Token Jumping**)]

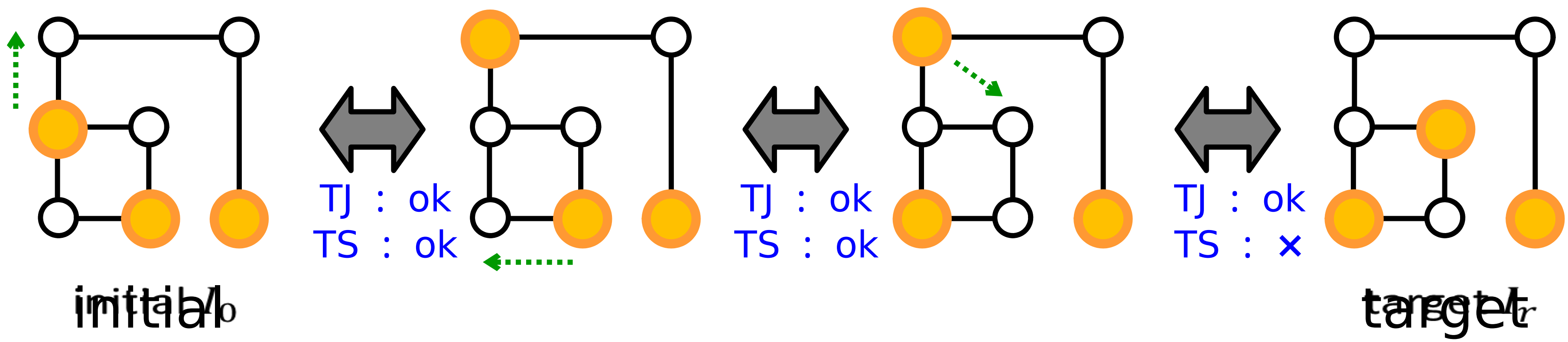
- feasible solutions: independent sets of size exactly
- adjacency relation: **move** a single token



Independent set of a graph:
a vertex subset such that no two vertices are adjacent.
(We regard a **token** is placed on each vertex in an independent set.)

Independent set reconfiguration

[**Independent set** reconfiguration (**Token Jumping**)]
- feasible solutions: independent sets of size exactly
- adjacency relation: **move** a single token



[**Independent set** reconfiguration (**Token Sliding**)]
- feasible solutions: independent sets of size exactly
- adjacency relation: **slide** a single token to its neighbor along an edge

* The figure above is a no-instance for Token Slic

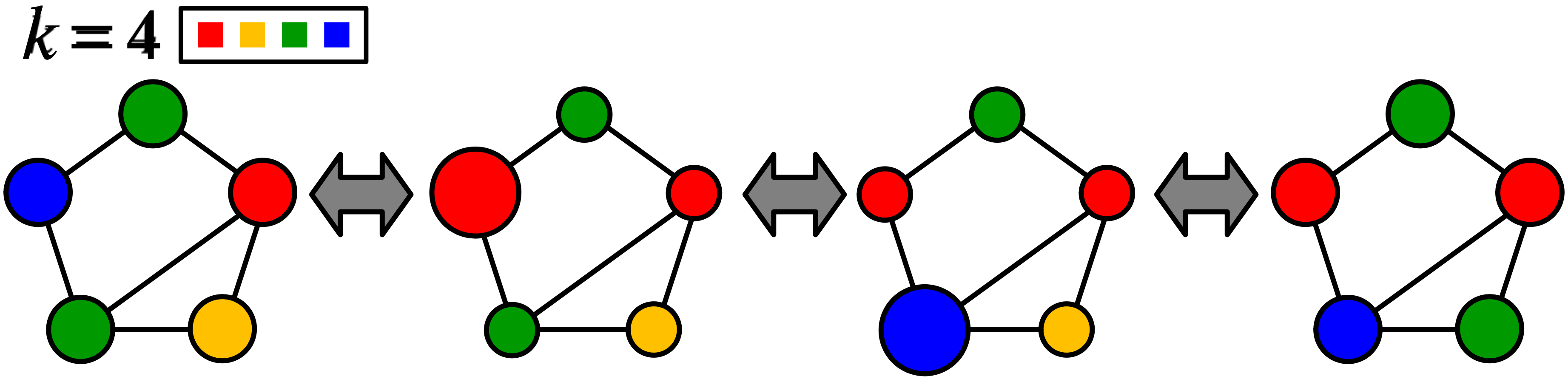
Reachability depends on the choice of adjacency relations. (= the structure of the solution space)

k-coloring reconfiguration

[**-coloring** reconfiguration]

- feasible solutions: -colorings of a graph
- adjacency relation: recoloring a single vertex

$k=4$

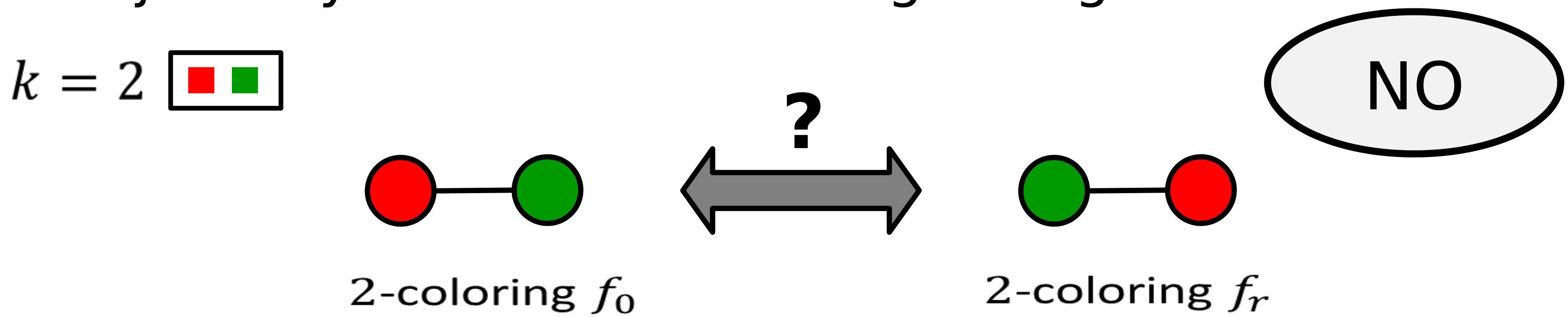


***k*-coloring** of a graph:
using at most k colors, color the vertices of a graph so that any two adjacent vertices receive different colors.

k-coloring reconfiguration

[**-coloring** reconfiguration]

- feasible solutions: -colorings of a graph
- adjacency relation: recoloring a single vertex



[**-coloring** reconfiguration (**Kempe change**)]

- feasible solutions: -colorings of a graph
- adjacency relation: swapping “connected” two color classes

* The figure above is a yes-instance under the Kempe change relation.

k-coloring reconfiguration and its generalization

Coloring reconfiguration is one of the most well-studied problems

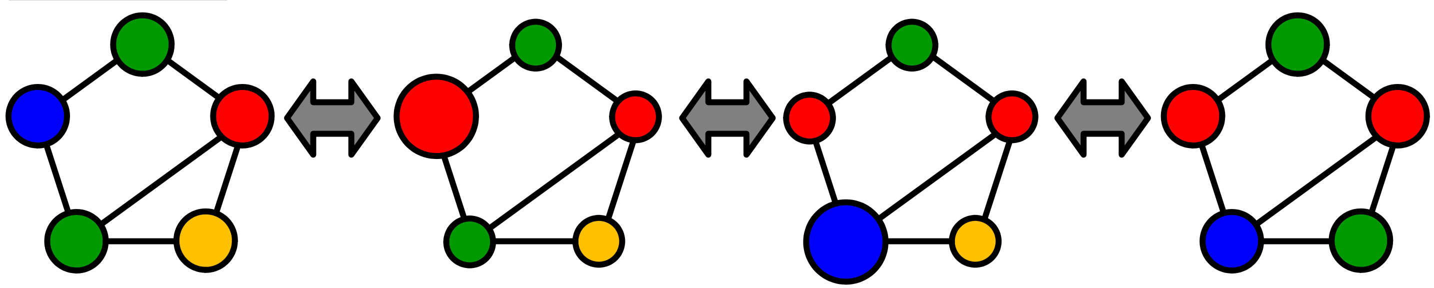
- will appear several times in this talk;
- has been studied for not only different types of adjacency relations but also **“generalized” types of feasible solutions**.

Generalized to Graph Homomorphism

[***k*-coloring** reconfiguration]

- feasible solutions: k-colorings of a graph
- adjacency relation: recoloring a single vertex

$k = 4$



4-coloring f_0 4-coloring f_1 4-coloring f_2 4-coloring f_r

5th talk of this mini-symposia

R. Brewster, S. Mcguinness, B. Moore, **J. Noel**. Reconfiguring graph homomorphisms and colourings.

History of combinatorial reconfiguration (from my viewpoint ...)

[2002 – 2012]

- Negative results (PSPACE-completeness)
- Sufficient conditions for yes-instances
- Algorithms obtained using mostly greedy methods

[2013 – now]

- Broader algorithmic techniques are starting to emerge

  These three years, ≥ 20 papers have been published @ arXiv

  ➔ later presented at ICALP, STACS, ISAAC, SWAT, WADS, etc.

- Algorithm methods capturing the solution space
  - Dynamic programming
  - Fixed-parameter tractability (FPT)

We now have techniques/results for **both** negative & positive sides!

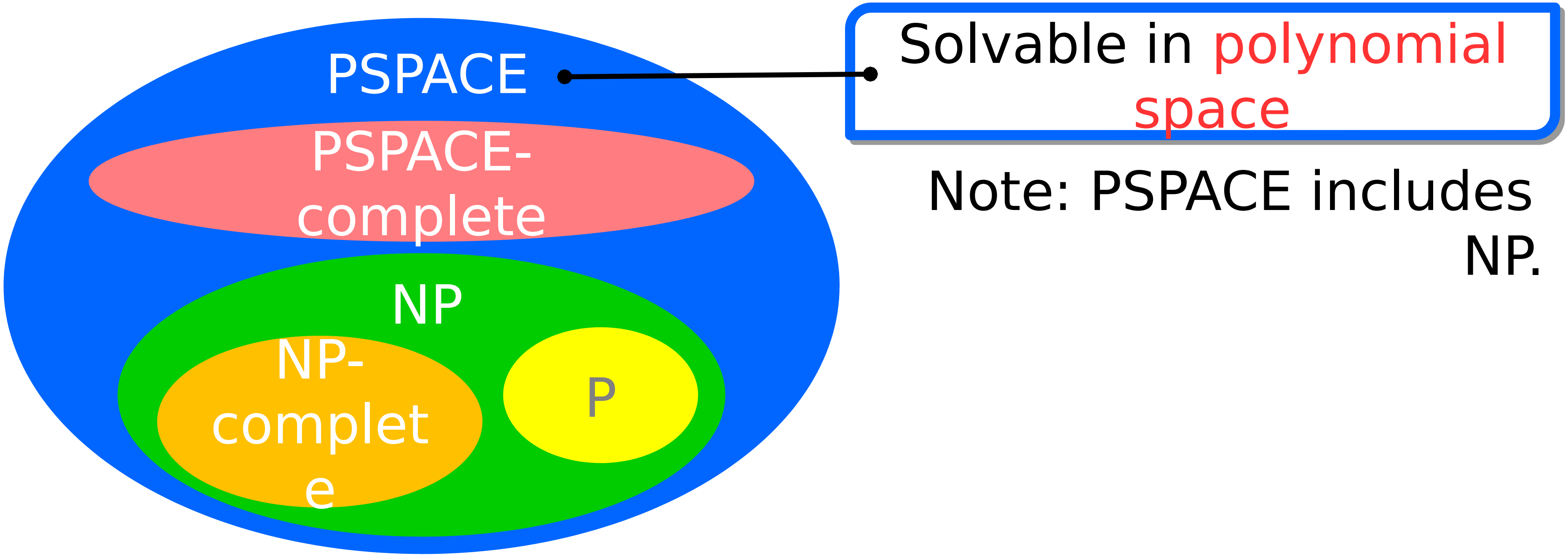
In this talk: I will give an **overview** of these techniques/results quickly!

PSPACE-completeness

[From the viewpoint of algorithm designer]

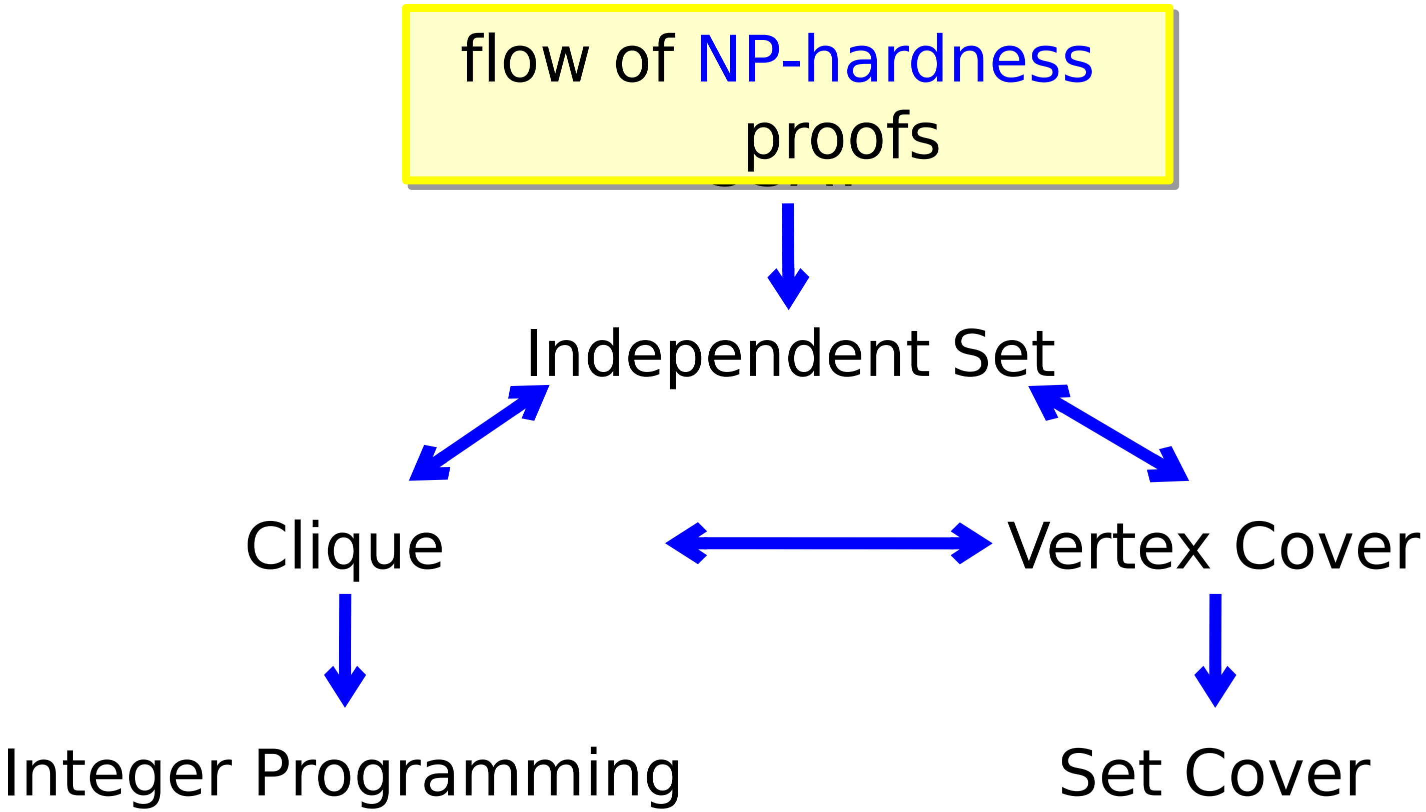
If a reconfiguration problem is **PSPACE-complete**, then

1. no polynomial-time algorithm under $\mathrm{P} \neq \mathrm{NP}$; and
2. exists a yes-instance whose **shortest** reconfiguration sequence requires **super-polynomial length** under $\mathrm{NP} \neq \mathrm{PSPACE}$.



Note: PSPACE includes NP.

PSPACE-hardness



PSPACE-hardness

For many NP-complete search problems, we can show the PSPACE-hardness of their reconfigurations by following the “flow” of NP-hardness reductions (with noting that they preserve the reachability).

flow of PSPACE-hardness proofs

Reconf.

PSPACE-complete [GKMP09]

Independent Set Reconf.

Clique Reconf. ⟷ Vertex Cover Reconf.

Integer Programming Reconf.

Set Cover Reconf.

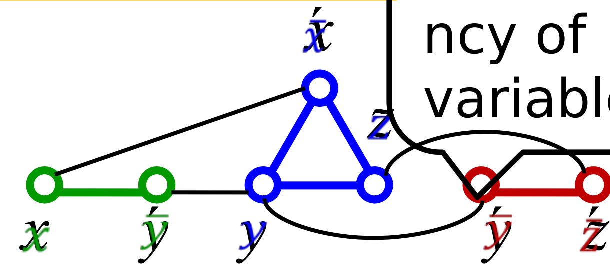
[GKMP09] P. Gopalan, P.G. Kolaitis, E.N. Maneva, C.H. Papadimitriou. The connectivity of Boolean satisfiability: computational and structural dichotomies. SIAM J. Computing 38, pp. 2330-2355 (2009)

Reduction for preserving the reachability

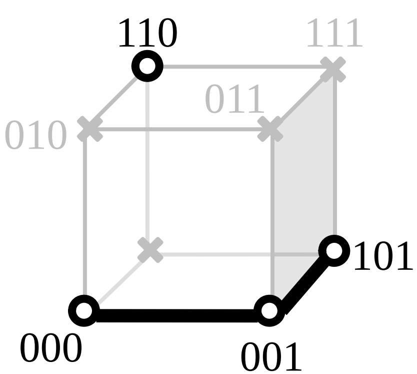
3SAT —reduction for NP-hardness→ Independent set

preserve consistency of variables

$$f = (x \lor \bar{y}) \land (\bar{x} \lor y \lor z) \land (\bar{y} \lor \bar{z})$$

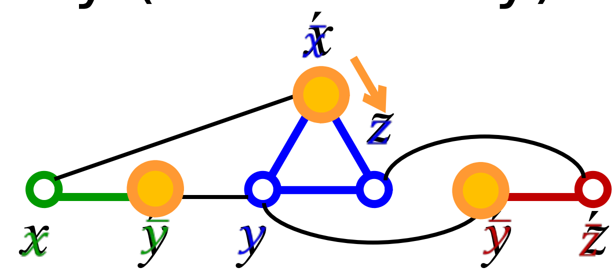


This reduction is correct for NP-hardness,
but does not preserve the connectivity (reachability) of solution space.



Solution space for SAT formula f

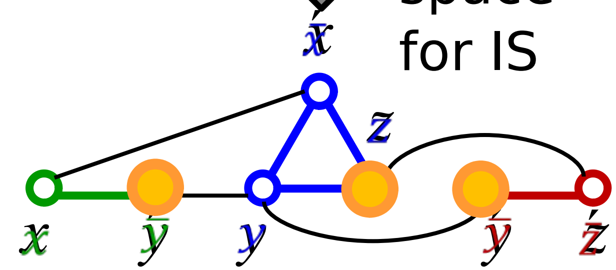
<u>**0**</u>0<u>**0**</u> 001 ← z: don't care



✖ <u>**NOT**</u> adjacent in solution space for SAT

adjacent in solution space for IS

<u>**1**</u>0<u>**1**</u> 001 ← x: don't care

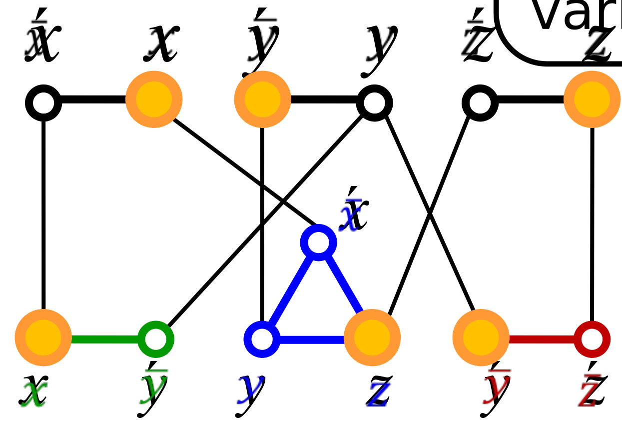


Reduction for preserving the reachability

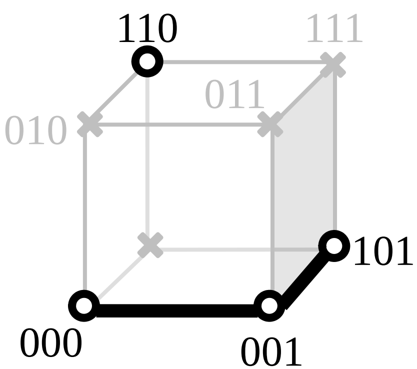
3SAT $\xrightarrow{\text{reduction for NP-hardness}}$ Independent set

preserve consistency of variables

$$f = (x \vee \acute{y}) \wedge (\acute{x} \vee y \vee z) \wedge (\acute{y} \vee \acute{z})$$



No “don’t care” variable

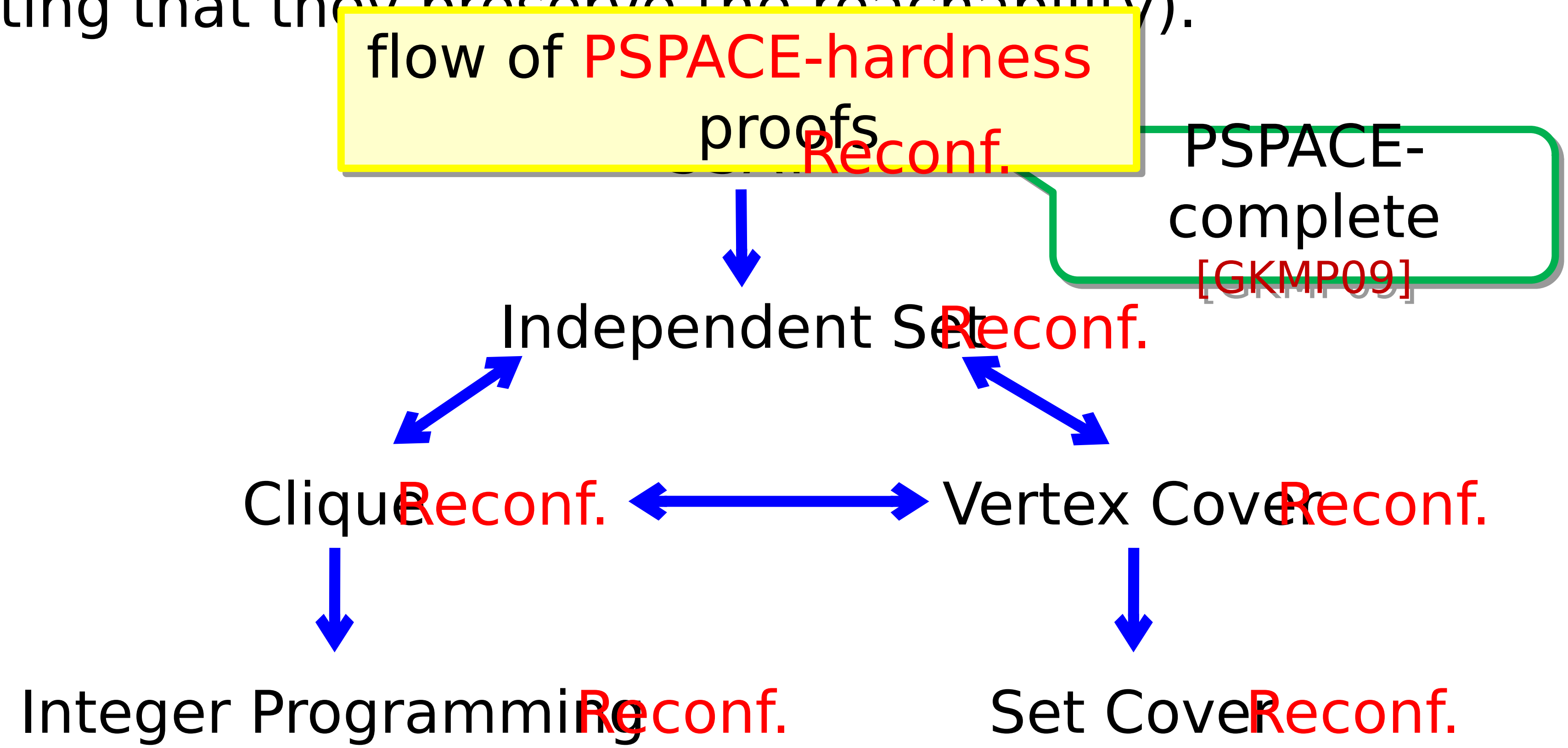


Solution space for SAT formula f

- Every independent set corresponds to exactly one satisfiable truth assignment of
- This reduction preserves the reachability of solutions spaces. (Details omitted)

PSPACE-hardness

For many NP-complete search problems, we can show the PSPACE-hardness of their reconfigurations by following the “flow” of NP-hardness reductions (with noting that they preserve the reachability).



[GKMP09] P. Gopalan, P.G. Kolaitis, E.N. Maneva, C.H. Papadimitriou. The connectivity of Boolean satisfiability: computational and structural dichotomies. SIAM J. Computing 38, pp. 2330-2355 (2009)

History of combinatorial reconfiguration (from my viewpoint ...)

[2002 – 2012]

- Negative results (PSPACE-completeness)
- Sufficient conditions for yes-instances
- Algorithms obtained using mostly greedy methods

[2013 – now]

- Broader algorithmic techniques are starting to emerge

  These three years, ≥ 20 papers have been published @ arXiv

  ➔ later presented at ICALP, STACS, ISAAC, SWAT, WADS, etc.

- Algorithm methods capturing the solution space
  - Dynamic programming
  - Fixed-parameter tractability (FPT)

We now have techniques/results for **both** negative & positive sides!

In this talk: I will give an **overview** of these techniques/results quickly!

Sufficient condition for -coloring reconfiguration

Showing when the solution space consists of a **single connected component**

[**-coloring** reconfiguration]

- feasible solutions: -colorings of a graph
- adjacency relation: recoloring a single vertex

Theorem: For an instance of -coloring reconfiguration, if , then it is a yes-instance.

degeneracy = coloring number

ex) Every planar graph satisfies , and hence

Thus, any two 7-colorings of a planar graph is a yes-instance.

Note: there are graphs whose chromatic # is .
In this sense, (roughly speaking) this theorem says that **only one additional color** is **sufficient to connect all colorings** of .

P. Bonsma, L. Cereceda. Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. Theoretical

Sufficient condition: Other examples

[H-coloring reconfiguration]

Several sufficient conditions on # of colors **where restricted to graph classes**. In particular, the diameters of solution spaces can be bounded by a polynomial length (quadratic)

[H-coloring reconfiguration]
Several sufficient conditions on # of colors are given **when restricted to graph classes**. In particular, the diameters of solution spaces can be bounded by a polynomial length (quadratic)

[BB13] M. Bonamy, N. Bousquet. Recoloring **bounded treewidth graphs**. Electronic Notes in Discrete Mathematics 44, pp. 257-262 (2013)

[BJLPP14] M. Bonamy, M. Johnson, I. Lignos, V. Patel, D. Paulusma. Reconfiguration graphs for vertex colourings of **chordal** and **chordal bipartite graphs**. J. Combinatorial Optimization 27, pp. 132-143 (2014)

For trees,
- constant # of additional colors
- polynomial diameter

[H-edge-coloring reconfiguration]

- feasible solutions: k-**edge**-colorings of a graph G
- adjacency relation: recoloring a single **edge**

[H-edge-coloring reconfiguration]
- feasible solutions: **edge**-colorings of a graph
- adjacency relation: recoloring a single **edge**

[IKD12] T. Ito, M. Kamiński, E.D. Demaine. Reconfiguration of list edge-colorings in a graph.

History of combinatorial reconfiguration (from my viewpoint ...)

[2002 – 2012]

- Negative results (PSPACE-completeness)
- Sufficient conditions for yes-instances
- Algorithms obtained using mostly greedy methods

[2013 – now]

- Broader algorithmic techniques are starting to emerge

  These three years, ≥ 20 papers have been published @ arXiv

  ➔ later presented at ICALP, STACS, ISAAC, SWAT, WADS, etc.

- Algorithm methods capturing the solution space
  - ➢ Dynamic programming
  - ➢ Fixed-parameter tractability (FPT)

[2013 – now]

- Broader algorithmic techniques are starting to emerge

  These three years, papers have been published @ arXiv

  later presented at ICALP, STACS, ISAAC, SWAT, WADS, etc.

- Algorithm methods capturing the solution space
  - ➢ Dynamic programming
  - ➢ Fixed-parameter tractability (FPT)

We now have techniques/results for **both** negative & positive sides!

In this talk: I will give an **overview** of these techniques/results quickly!

Greedy algorithm

Idea: Take the symmetric difference between two given solutions, and transform the difference one by one

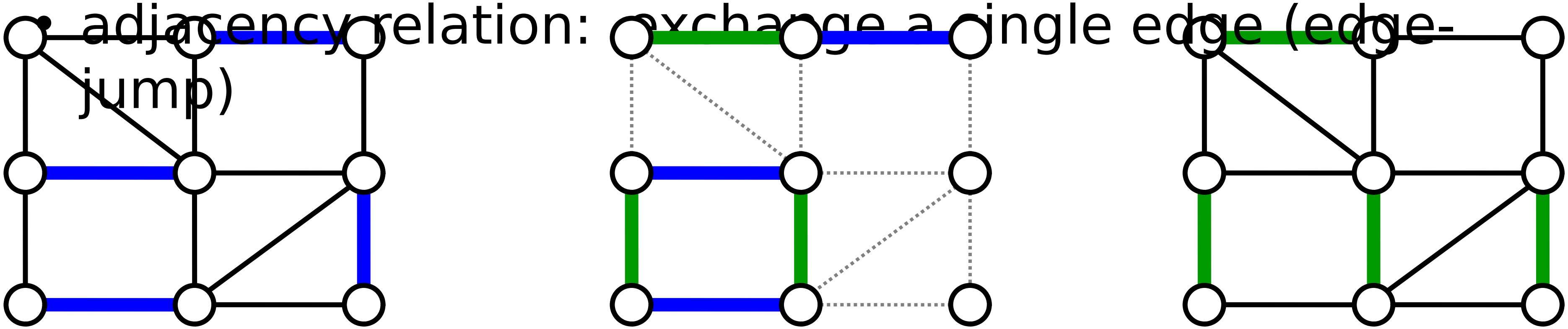
Ensure: feasibility of intermediate solutions

- “no” if we cannot obtain a reconfiguration by this way

[**Matching** reconfiguration]

- feasible solutions: matchings of a graph with cardinality exactly k
- adjacency relation: exchange a single edge (edge-jump)

[Matching reconfiguration]
• feasible solutions: matchings of a graph with cardinality exactly
• adjacency relation: exchange a single edge (edge-jump)



Matching M_0 Symmetric difference Matching M_r

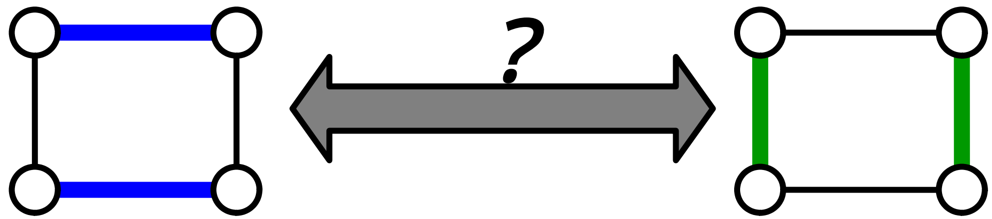
$$M_0 \triangle M_r = (M_0 \setminus M_r) \cup (M_r \setminus M_0)$$

Greedy algorithm for matching reconfiguration

[**Matching** reconfiguration]

- feasible solutions: matchings of a graph with cardinality exactly
- adjacency relation: exchange a single edge (edge-jump)

[Matching reconfiguration]
- feasible solutions: matchings of a graph with cardinality exactly k
- adjacency relation: exchange a single edge (edge-jump)



Matching M_0 Matching M_r

NO

We can characterize the no-instances using the Edmonds–Gallai decomposition, and solve the problem in polynomial time.

T. Ito, E.D. Demaine, N.J.A. Harvey, C.H. Papadimitriou, M. Sideri, R. Uehara, Y. Uno.
On the complexity of reconfiguration problems.
Theoretical Computer Science 412, pp. 1054-1065 (2011).

Greedy algorithm: Other examples

[**Independent set** reconfiguration (**Token Jumping**)]

- feasible solutions: independent sets of size exactly
- adjacency relation: move a single token

[Independent set reconfiguration (Token Jumping)]
• feasible solutions: independent sets of size exactly k
• adjacency relation: move a single token

Theorem: Token Jumping is solvable in linear time for even-hole-free graphs.

M. Kamiński, P. Medvedev, M. Milanič. Complexity of independent set reconfigurability problems. Theoretical Computer Science 439, pp. 9-15 (2012)

[**Minimum spanning tree** reconfiguration]

- feasible solutions: minimum spanning trees of a weighted graph

Theorem: Minimum spanning tree reconfiguration is solvable in polynomial time for any graph.

T. Ito, E.D. Demaine, N.J.A. Harvey, C.H. Papadimitriou, M. Sideri, R. Uehara, Y. Uno. On the complexity of reconfiguration problems. Theoretical Computer Science 412, pp. 1054-1065 (2011)

History of combinatorial reconfiguration (from my viewpoint …)

[2002 – 2012]

- Negative results (PSPACE-completeness)
- Sufficient conditions for yes-instances
- Algorithms obtained using mostly greedy methods

[2013 – now]

- Broader algorithmic techniques are starting to emerge

  These three years, ≥ 20 papers have been published @ arXiv

  ➔ later presented at ICALP, STACS, ISAAC, SWAT, WADS, etc.

- Algorithm methods capturing the solution space
  - ➢ Dynamic programming
  - ➢ Fixed-parameter tractability (FPT)

[2013 – now]

- Broader algorithmic techniques are starting to emerge

  These three years, papers have been published @ arXiv

  later presented at ICALP, STACS, ISAAC, SWAT, WADS, etc.

- Algorithm methods capturing the solution space
  - ➢ Dynamic programming
  - ➢ Fixed-parameter tractability (FPT)

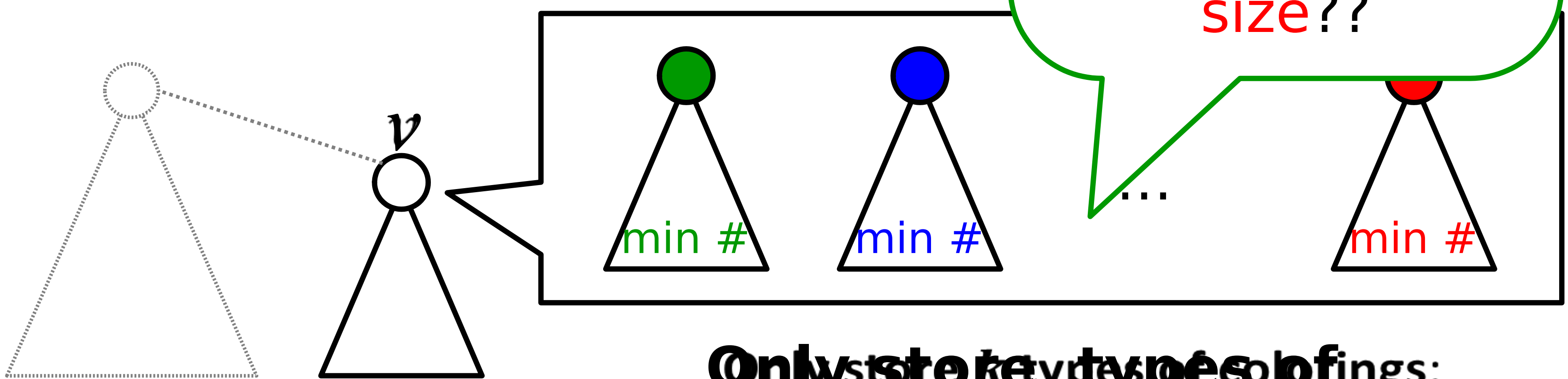
We now have techniques/results for **both** negative & positive sides!

In this talk: I will give an **overview** of these techniques/results quickly!

Dynamic Programming

It is a natural idea to try the **DP method** for reconfiguration.
However, only a few positive results on DP method.

How to store the information about "reachability" within a polynomial size??



Ex: k-coloring of a tree (as a search problem)

Only store k types of colorings:
The min # of colors under the assumption that v is colored with c_i

Ex: coloring of a tree (as a search problem)

Only store types of colorings:
The min # of colors under the assumption that is colored

DP algorithm for list coloring reconfiguration

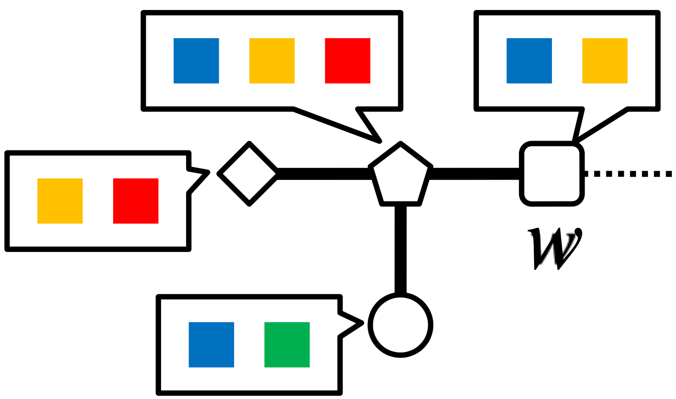
[List coloring reconfiguration]

- feasible solutions: list colorings of a graph G
- adjacency relation: recoloring a single vertex

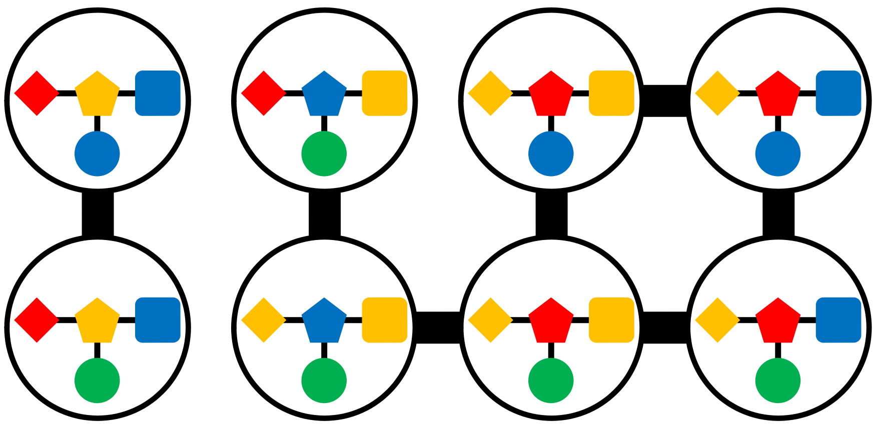
[List coloring reconfiguration]

- feasible solutions: list colorings of a graph
- adjacency relation: recoloring a single vertex

Subtree T_w
Subtree



Solution space for T_w
Solution space for



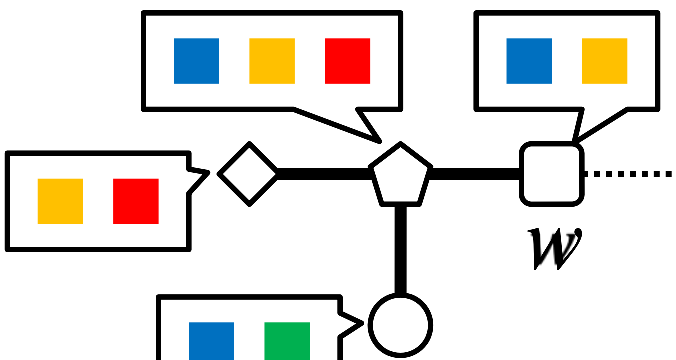
DP algorithm for list coloring reconfiguration

[**List coloring** reconfiguration]

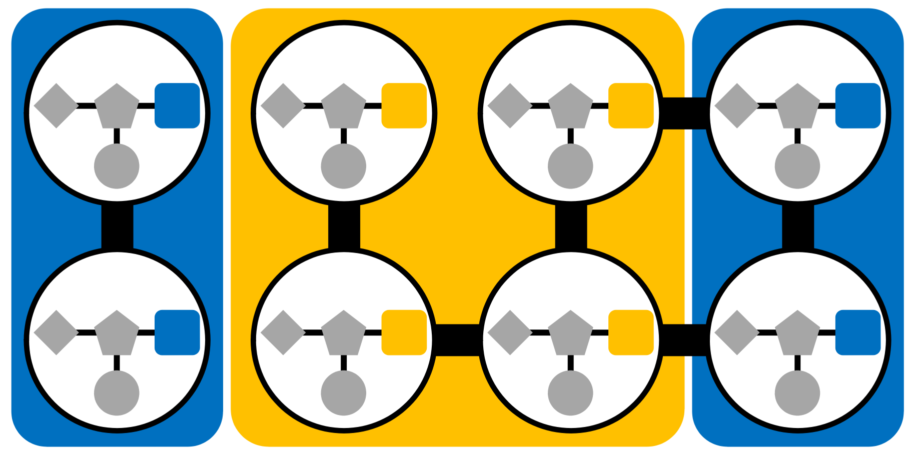
- feasible solutions: list colorings of a graph G
- adjacency relation: recoloring a single vertex

Focus on the color assigned to the vertex w which is adjacent to the outside T_w ...

Subtree T_w



Solution space for T_w



Need to store such a reachability information within a polynomial size

All four colorings assign blue to w, but they belong to two different components in the solution space.

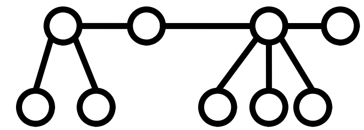
DP algorithm for list coloring reconfiguration

[**List coloring** reconfiguration]
[**List coloring** reconfiguration]

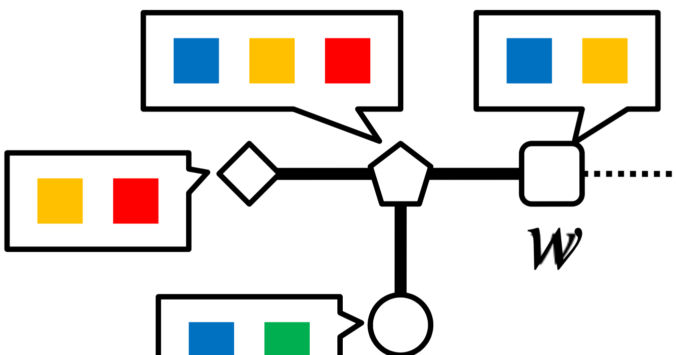
- feasible solutions: list colorings of a graph G
- adjacency relation: recoloring a single vertex

- feasible solutions: list colorings of a graph
- adjacency relation: recoloring a single vertex

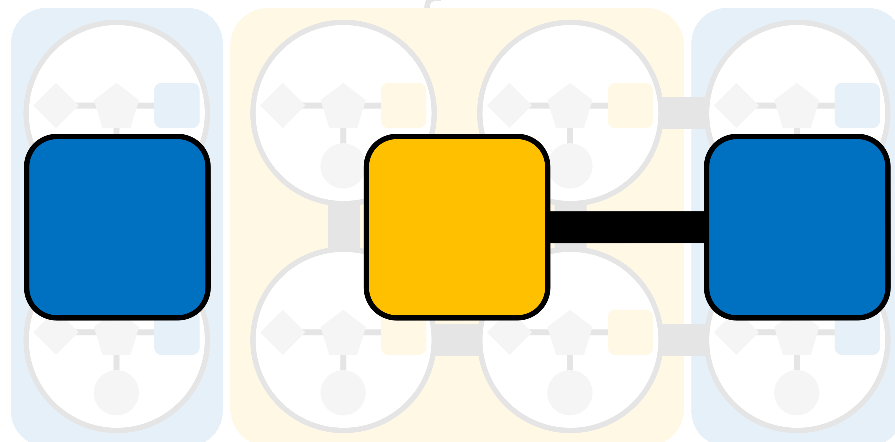
Theorem: List coloring reconfiguration is solvable in polynomial time for caterpillars.



Subtree T_w



Contracted solution space for T_w



T. Hatanaka, T. Ito, X. Zhou. The list coloring reconfiguration problem for bounded pathwidth graphs. IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences E98-A, pp. 1168-1178 (2015)

DP algorithm: Other examples

[**Shortest path** reconfiguration]

- feasible solutions: shortest paths of an unweighted graph G
- adjacency relation: switch a single intermediate vertex

Theorem: Shortest path reconfiguration is solvable in polynomial time for unweighted planar graphs.

P. Bonsma. Rerouting shortest paths in planar graphs. Proc. FSTTCS 2012, LIPIcs 18, pp. 337-349 (2012)

[**k-coloring** reconfiguration]

- feasible solutions: k-colorings of a graph G
- adjacency relation: recoloring a single vertex

Theorem: k-coloring reconfiguration is solvable in polynomial time for $(k-2)$-connected chordal graphs.

polynomial time for connected chordal graphs.

P. Bonsma, D. Paulusma. Using contracted solution graphs for solving reconfiguration problems. arXiv:1509.06357 (2015)

History of combinatorial reconfiguration (from my viewpoint …)

[2002 – 2012]

- Negative results (PSPACE-completeness)
- Sufficient conditions for yes-instances
- Algorithms obtained using mostly greedy methods

[2013 – now]

- Broader algorithmic techniques are starting to emerge

  These three years, $\geq$ 20 papers have been published @ arXiv

  ➔ later presented at ICALP, STACS, ISAAC, SWAT, WADS, etc.

- Algorithm methods capturing the solution space
  - ➢ Dynamic programming
  - ➢ Fixed-parameter tractability (FPT)

[2013 – now]
- Broader algorithmic techniques are starting to emerge
These three years, papers have been published @ arXiv
later presented at ICALP, STACS, ISAAC, SWAT, WADS, etc
- Algorithm methods capturing the solution space
➢ Dynamic programming
➢ Fixed-parameter tractability (FPT)

We now have techniques/results for **both** negative & positive sides!

In this talk: I will give an **overview** of these techniques/ results quickly!

FPT algorithms for reconfiguration problems

Solution space has exponential size

Previous polynomial-time algorithms:

Difficult to characterize the no-instances.

➔ In FPT algorithms, this can be done by the brute-force manner!

Outline of FPT algorithms

1. Give a sufficient condition for a yes-instance;
2. Based on the sufficient condition, kernelize a given instance into an FPT size; and
3. Construct the solution space for the kernelized instance by the

characterizing yes-instances is non-trivial

trivial part

Solution space for the kernelized instance has an **FPT size**.

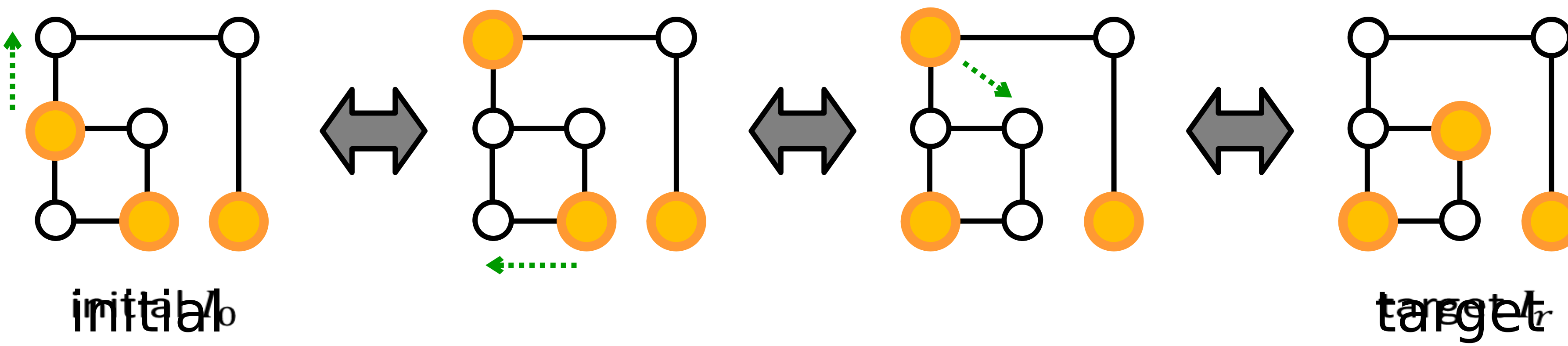
➔ We can enumerate all possible reconfiguration sequences.

Note: Answering "no" happens only in this step.

FPT algorithm for Token Jumping

[**Independent set** reconfiguration (**Token Jumping**)]

- feasible solutions: independent sets of size exactly k
- adjacency relation: move a single token



FPT algorithm for Token Jumping on general graphs

Parameter: $k + d$

k: bound on # of tokens $|I_0| = |I_r|$

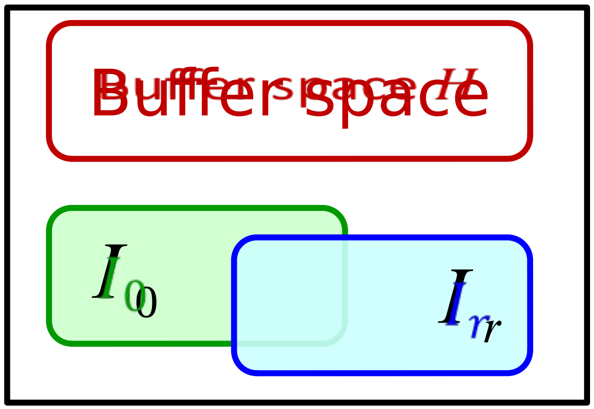
d: bound on maximum degree $\Delta(G)$ of a graph G

FPT algorithm for Token Jumping

Parameter: $k + d$ (# of tokens $|I_0| = |I_r| \le k$ and max degree $\Delta(G) \le d$)

1. Give a sufficient condition for a yes-instance

If $|V(G)| \ge 3k(d+1)$, then it is a yes-instance.



Graph G

Delete all vertices in $I_0 \cup I_r$ and their neighbors from G. Then, the remaining graph H is the "**safe place**" from $I_0 \cup I_r$.

$$|V(G)| \ge 3k(d+1)$$

$k(d+1) \leftarrow I_0$ and its neighbors

$k(d+1) \leftarrow I_r$ and its neighbors

$$|V(H)| \ge k(d+1)$$

H has an independent set I^* of size $\ge k$

Since H has an independent set I^* of size $\ge k$, we can use it as a buffer space, that is, I_0 and I_r are reconfigurable via I^*.

FPT algorithm for Token Jumping

Parameter: $k + d$ (# of tokens $|I_0| = |I_r| \le k$ and max degree $\Delta(G) \le d$)

1. Give a **sufficient condition** for a yes-instance

   If $|V(G)| \ge 3k(d+1)$, then it is a yes-instance.

   ➔ Output "yes" if G satisfies the condition.

2. **Kernelize** a given instance into an FPT size

   ➔ This step is executed **only when** $|V(G)| < 3k(d+1)$

   ➔ Thus, G is of an FPT size already

3. Construct the **solution space** by the brute-force manner

   ➔ # of independent sets in G of size exactly k can be bounded by

   $$O(|V(G)|^k) < O\left((3k(d+1))^k\right)$$

   ➔ Solution space has an **FPT size**, and be constructed in **FPT time**.

   (We can check the reachability between I_0 and I_r by a breadth-first search.)

Parameterized complexity of Token Jumping

[**Independent set** reconfiguration (**Token Jumping**)]

- feasible solutions: independent sets of size exactly k
- adjacency relation: move a single token

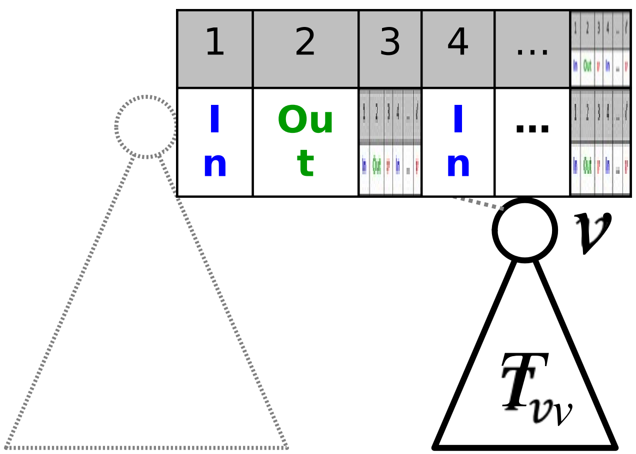
Parameter	Graph class	Result
# of tokens + max degree	general	FPT [IKOSUY14]
# of tokens **only**	general	W[1]-hard [IKOSUY14]
	nowhere dense, bounded degeneracy (includes planar, bounded treewidth)	FPT [LMPRS15] [IKO14] also shows FPT for planar

[IKOSUY14] T. Ito, M. Kamiński, H. Ono, A. Suzuki, R. Uehara, K. Yamanaka. On the parameterized complexity for token jumping on graphs. Proc. TAMC 2014, LNCS 8402, pp. 341-351 (2014)
[LMPRS15] D. Lokshtanov, A.E. Mouawad, F. Panolan, M.S. Ramanujan, S. Saurabh. Reconfiguration on sparse graphs. Proc. WADS 2015, LNCS 9214, pp. 398-409 (2015)
[IKO14] T. Ito, M. Kamiński, H. Ono. Fixed-parameter tractability of token jumping on planar graphs. Proc. ISAAC 2014, LNCS 8889, pp. 208-219 (2014)

FPT algorithms with length parameter

DP methods work nicely when the **length ℓ of a sequence** is taken as the parameter.

DP methods work nicely when the **length of a sequence** is taken as the parameter.



Token Jumping for trees

(solvable in P, though)

Store **what happens at the i-th step**, $i \in \{1,2,\ldots,\ell\}$, of a reconfiguration sequence by distinguishing the following **three**:

- touched token on the separator v
- touched token on a vertex inside T_v
- touched token on a vertex outside T_v

➔ all possible patterns can be bounded by 3^ℓ, and hence the size of DP tables can be bounded by an **FPT size**.

Store **what happens at the -th step** of a reconfiguration sequence by distinguishing the following **three**:
- touched token on the separator
- touched token on a vertex inside
- touched token on a vertex outside
➔ all possible patterns can be bounded by , and hence the size of DP tables can be bounded by an **FPT size**

Thm: For every search problem expressible by the Monadic Second-Order Logic, its reconfiguration is in FPT when parameterized by treewidth and the length ℓ of a reconfiguration sequence.

in FPT when parameterized by treewidth and the length of a reconfiguration sequence.

tree decompositions. Proc. IPEC 2014, LNCS 8894, pp. 246-257 (2014)

Future work (from my viewpoint …)

[2002 – 2012]

- Negative results (PSPACE-completeness)
- Sufficient conditions for yes-instances
- Algorithms obtained using mostly greedy methods

[2013 – now]

- Algorithm methods capturing the solution space
  - Dynamic programming
  - Fixed-parameter tractability (FPT)

We now have techniques/results for **both** negative & positive sides!

[General Question]

- Clarify **relationships** on complexity between search problems and their reconfiguration problems?

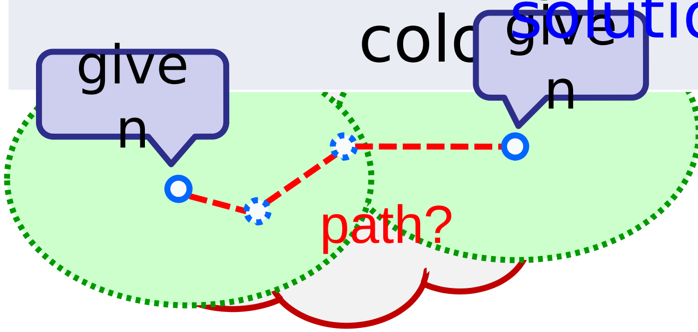
Search problem vs its reconfiguration proble m

For many **NP-complete** search problems,
their reconfiguration problems are **PSPACE-complete**.

	Search	Reconfiguration
3-coloring	NP-complete	P
L(2,1)-labeling with colo...		

Difficult to find one solution

Easy to check the



There are exponentially many solutions,
but each connected component of the solution space is of polynomial size.

Our advantage: initial & target solutions are given us an input

➔ check only polynomial number of solutions **around th...**

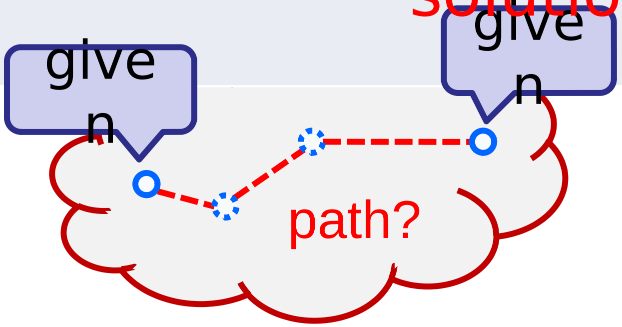
OPEN: Is there a reconfiguration problem which is in P, but
its solution space has a super-polynomial ...

Search problem vs its reconfiguration problem

For several search problems in **P**,
their reconfiguration problems are also in **P**. But, …

	Search	Reconfiguration
4-coloring for bipartite graphs	P	PSPACE-complete
shortest path	P	PSPACE-complete

Easy to find one solution ⟺ Difficult to check the reachability



So far, I don't have intuitive explanations to what makes these problems difficult in reconfiguration…

Future work (from my viewpoint …)

[2002 – 2012]

- Negative results (PSPACE-completeness)
- Sufficient conditions for yes-instances
- Algorithms obtained using mostly greedy methods

[2013 – now]

- Algorithm methods capturing the solution space
  - Dynamic programming
  - Fixed-parameter tractability (FPT)

We now have techniques/results for **both** negative & positive sides!

[General Question]

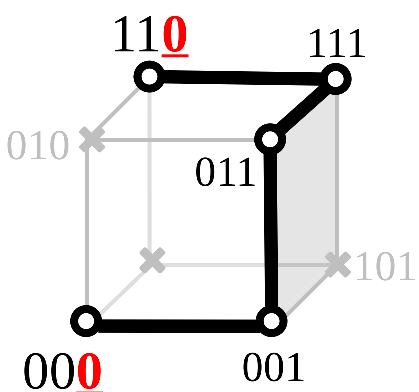
- Clarify **relationships** on complexity between search problems and their reconfiguration problems?
- Give a (sufficient) condition for which the **DP method** yields a polynomial-time algorithm?
- **Shortest** variant?

Shortest variant

asks for the length of a **shortest** reconfiguration sequence.

[2015 – now]

- Algorithms for **shortest** variant, capturing “**detours**”
  - SAT reconfiguration [MNPR15]
  - Independent set reconfiguration (Token Sliding) for caterpillars [YU16]



Solution space
for a SAT formula

[Difficult point]

Even though in both initial & target, we need to flip once for the feasibility.

Almost all previously known algorithms for shortest variants touch only the symmetric difference

□ no detour.

4th talk of this mini-symposia

[MNPR15] **A.E. Mouawad**, N. Nishimura, V. Pathak, V. Raman. Shortest reconfiguration paths in the solution space of Boolean formulas. Proc. ICALP 2015, LNCS 9134, pp. 985-996 (2015)

[YU16] T. Yamada, R. Uehara. Shortest reconfiguration of sliding tokens on a caterpillar. Proc. WALCOM 2016, LNCS 9627, pp. 236-248 (2016)

Conclusion

[2002 – 2012]

- Negative results (PSPACE-completeness)
- Sufficient conditions for yes-instances
- Algorithms obtained using mostly greedy methods

[2013 – now]

- Algorithm methods capturing the solution space
  - Dynamic programming
  - Fixed-parameter tractability (FPT)

[2015 – now]

- Algorithms for shortest variant, capturing “detours”
  - SAT reconfiguration
  - Independent set reconfiguration (Token Sliding) for caterpillars

We now have techniques/results for **both** negative & positive sides!

... but, we still have several interesting open problems!!