The Complexity of Change

Jan van den Heuvel

24th BCC, 5 July 2013

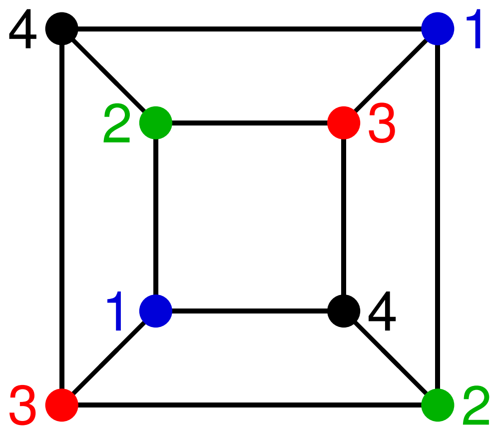
Department of Mathematics
London School of Economics and Political Science

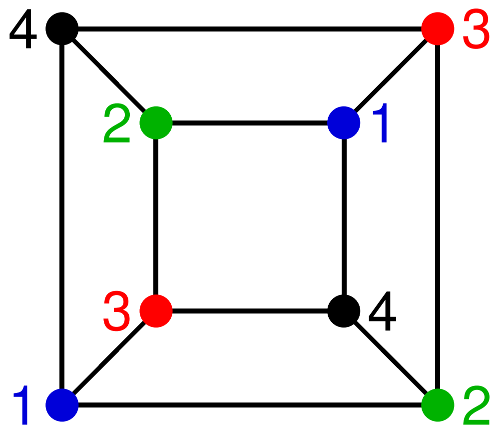
Recolouring planar bipartite graphs

- *Input*: a planar, bipartite graph G,
  and two proper 4-colourings of G

  Question: can we change one 4-colouring to the other one,
  by recolouring 1 vertex at the time,
  while always maintaining a proper 4-colouring?

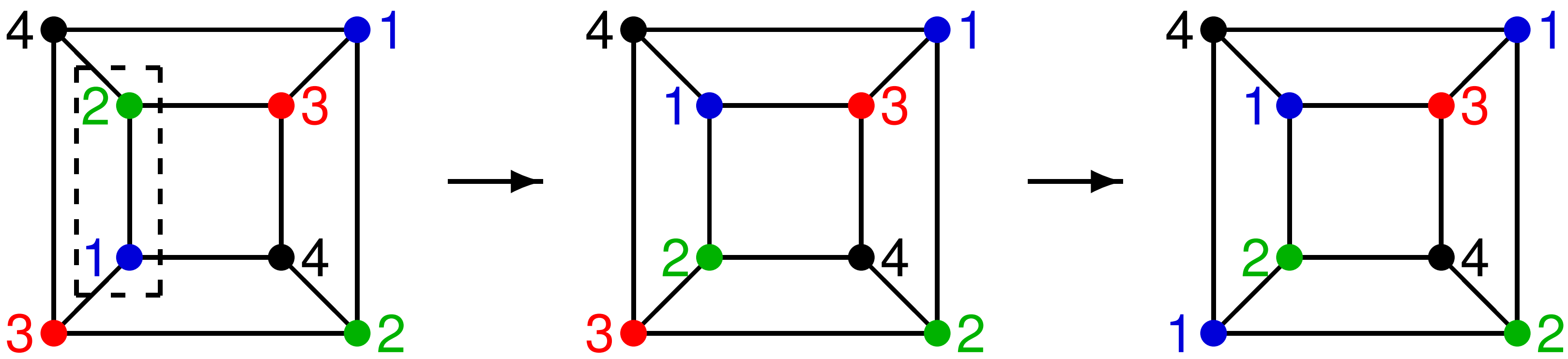
- no, not always:





But what if . . .

- what if we would be able to recolour with Kempe chains?
  - i.e., for any two colours c_1, c_2, we can swap the colours on any (c_1, c_2)-coloured component



→ etc. → to any 4-colouring of the cube!

The two kinds of reconfiguration problems

- A-TO-B-PATH

  Input: some collection of feasible configurations, some collection of allowed transformations, and two feasible configurations *A*, *B*

  Question: can we go from *A* to *B* by a sequence of transformations, so that each intermediate configuration is feasible as well?

- PATH-BETWEEN-ALL-PAIRS

  Input: some collection of feasible configurations, and some collection of allowed transformations

  Question: is it possible to do the above for any two feasible configurations *A*, *B*?

The complexity classes we need

- **P**: Polynomial-Time
  - if you are clever, you can find the answer in polynomial time

The complexity classes we need

- **P**: Polynomial-Time
- **NP**: Non-Deterministic Polynomial-Time
  - if the answer is "yes" and you are lucky,
    you can discover the "yes" in polynomial time

The complexity classes we need

- **P**: Polynomial-Time
- **NP**: Non-Deterministic Polynomial-Time
- **PSPACE**: Polynomial-Space
  - if you are clever, you can find the answer using a polynomial amount of memory

The complexity classes we need

- **P**: Polynomial-Time
- **NP**: Non-Deterministic Polynomial-Time
- **PSPACE**: Polynomial-Space
- **NPSPACE**: Non-Deterministic Polynomial-Space
  - if the answer is “yes” and you are lucky, you can discover the “yes” using a polynomial amount of memory

The complexity classes we need

- **P**: Polynomial-Time
- **NP**: Non-Deterministic Polynomial-Time
- **PSPACE**: Polynomial-Space
- **NPSPACE**: Non-Deterministic Polynomial-Space
- (there really should be a class **Constant**, for problems that can be solved by algorithms like "print("yes")" or "print("no")")

The complexity classes we need

- **P**: Polynomial-Time
- **NP**: Non-Deterministic Polynomial-Time
- **PSPACE**: Polynomial-Space
- **NPSPACE**: Non-Deterministic Polynomial-Space

- easy: **P** $\subseteq$ **NP** $\subseteq$ **PSPACE** $\subseteq$ **NPSPACE**
- and in fact: **PSPACE** $=$ **NPSPACE** (Savitch, 1970)

How to describe a problem?

- when being given a particular reconfiguration problem, we don't expect to being told an exhaustive list of all feasible configurations and/or an exhaustive list of all related pairs

- instead we assume we are told:
  - a "description" of all feasible configurations,
  - and a "description" of the allowed transformations

How to describe a problem?

- when being given a particular reconfiguration problem, we don't expect to being told an exhaustive list of all feasible configurations and/or an exhaustive list of all related pairs
- i.e., we assume the input is in the form of two algorithms to decide
  - if a possible configuration is feasible,
  - and if a possible transformation is allowed
- and we assume these algorithms give the correct answer in polynomial time

The complexity of all reconfiguration problems

- **Under these assumptions**

  A-TO-B-PATH and PATH-BETWEEN-ALL-PAIRS are in **NPSPACE**

  (and hence in **PSPACE**)

- suppose we want to decide if we can go from A to B
  - starting from A, "guess" a next configuration A_1
    - check that A_1 is feasible
    - check that going from A to A_1 is an allowed transformation
  - if A_1 is a valid next configuration,
    "forget" A and replace it by A_1
  - repeat those steps until the target configuration B is reached

Reconfiguration of satisfiability problems

- consider some Boolean formula with n variables
  - e.g.: $\varphi = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2})$

    whose set of satisfying assignments is

    $$\{\, (0,0,0),\ (0,1,0),\ (0,1,1),\ (1,0,0),\ (1,0,1)\,\}$$

- the allowed transformation is: change one bit x_i at the time

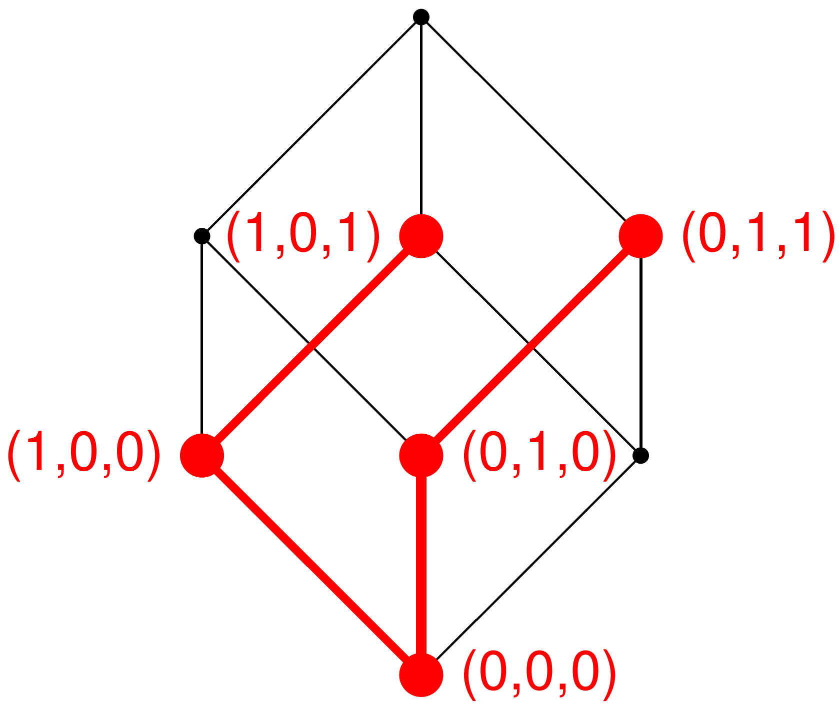
- natural questions:
  - is the set of all satisfying assignments connected?
  - given two satisfying assignments, can you go from one to the other?

Reconfiguration of satisfiability problems

- for a Boolean formula φ, the set of satisfying assignments is an induced subgraph of the n-dimensional hypercube

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2})$$

corresponds to:



Deciding satisfiability problems

- Schaefer (1978) considered “types” of Boolean formulas that can be defined using certain logical relations
- depending on what logical relations are allowed:
  - the decision problem whether or not a Boolean formula is satisfiable is always either in **P** or **NP-complete**

Deciding satisfiability problems

- Schaefer (1978) considered “types” of Boolean formulas that can be defined using certain logical relations

- Gopalan, Kolaitis, Maneva & Papadimitriou (2009) tried to use the same set-up to prove results on:
  - given the type of logical relations allowed
    - what is the complexity of deciding A-TO-B-PATH for two satisfying assignments of some Boolean formula?
    - and what is the complexity of PATH-BETWEEN-ALL-PAIRS

      (i.e., when is the set of satisfying assignments a connected subgraph of the hypercube)?

Reconfiguration of satisfiability problems

Theorem (Gopalan, Kolaitis, Maneva & Papadimitriou, 2009)

for Boolean formulas formed from some fixed set of logical relations:

- A-TO-B-PATH for two satisfying assignments of some Boolean formula is either in **P** or **PSPACE-complete**
  - the boundary between the two classes is different from the boundary between **P** and **NP-complete** for satisfiability

Reconfiguration of satisfiability problems

Theorem (Gopalan, Kolaitis, Maneva & Papadimitriou, 2009)

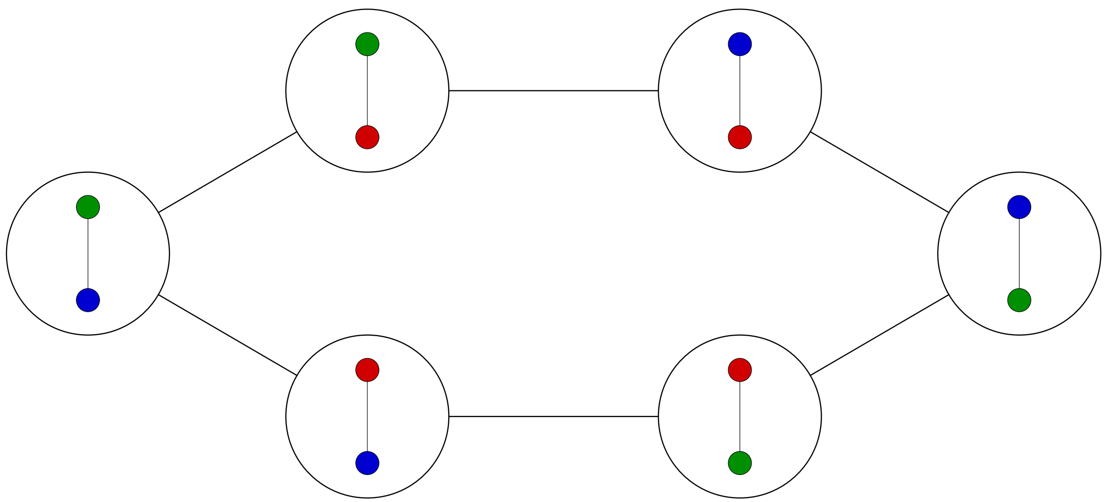
for Boolean formulas formed from some fixed set of logical relations:

- A-TO-B-PATH for two satisfying assignments of some Boolean formula is either in **P** or **PSPACE-complete**
- for the cases that A-TO-B-PATH is **PSPACE-complete**, PATH-BETWEEN-ALL-PAIRS is also **PSPACE-complete**
- in the cases that A-TO-B-PATH is in **P**, PATH-BETWEEN-ALL-PAIRS can be in **P**, in **coNP**, or **coNP-complete**
  - the boundaries between the classes are far from clear

Reconfiguration of graph colourings

- - we consider proper colourings using a fixed number of colours
  - and we are allowed to recolour one vertex at the time

- **example**: the structure of 3-colourings of K_3



Reconfiguration of graph colourings

- k-Colour-A-to-B-Path

  Input: a graph G,
  and two k-colourings A and B of G

  Question: can we go from A to B
  by recolouring one vertex at the time,
  always maintaining a proper k-colouring?

- k-Colour-Path-between-All-Pairs

  Input: a graph G

  Question: can we go between any two k-colourings
  in the manner above?

Reconfiguration of graph colourings

Recall

- if $k = 2$, then deciding if a graph is k-colourable is in **P**
- if $k \geq 3$, then deciding if a graph is k-colourable is **NP-complete**

Theorem

- if $k = 2, 3$, then K-COLOUR-A-TO-B-PATH is in **P**

  (Cereceda, vdH & Johnson, 2011)

- if $k \geq 4$, then K-COLOUR-A-TO-B-PATH is **PSPACE-complete**

  (Bonsma, Cereceda, 2009)

Reconfiguration of graph colourings

Completely trivial

restricted to bipartite, planar graphs:

- for any $k \geq 2$, deciding if a graph is k-colourable is in **Constant**:

  print(“yes”)

Reconfiguration of graph colourings

Completely trivial

restricted to bipartite, planar graphs:

- for any $k \geq 2$, deciding if a graph is k-colourable is in **Constant**

Theorem

restricted to bipartite, planar graphs:

- if $k = 2, 3$, then K-COLOUR-A-TO-B-PATH is in **P**

  (Cereceda, vdH & Johnson, 2011)

- if $k = 4$, then K-COLOUR-A-TO-B-PATH is **PSPACE-complete**

  (Bonsma, Cereceda, 2009)

- if $k \geq 5$, then K-COLOUR-A-TO-B-PATH is in **Constant** ("yes")

Theorem

restricted to bipartite graphs:

- if $k = 2$, then k-COLOUR-PATH-BETWEEN-ALL-PAIRS is in **P**:

  if no edges then print("yes"), else print("no")

- if $k = 3$,
  then k-COLOUR-PATH-BETWEEN-ALL-PAIRS is **coNP-complete**
  (Cereceda, vdH & Johnson, 2009)

- if $k \geq 4$, then the complexity of
  k-COLOUR-PATH-BETWEEN-ALL-PAIRS is unknown

Reconfiguration of graph colourings

Theorem

restricted to bipartite, planar graphs:

- if $k = 3$,
  then K-COLOUR-PATH-BETWEEN-ALL-PAIRS is in **P**
  (Cereceda, vdH & Johnson, 2009)
- if $k = 4$, then the complexity of
  K-COLOUR-PATH-BETWEEN-ALL-PAIRS is unknown
- if $k \geq 5$,
  then K-COLOUR-PATH-BETWEEN-ALL-PAIRS is in **Constant**:

  print("yes")

Reconfiguration with Kempe chains

- a Kempe chain transformation of a k-colouring of a graph involves:
  - for two colours c_1, c_2,
    swap the colours on a (c_1, c_2)-coloured component

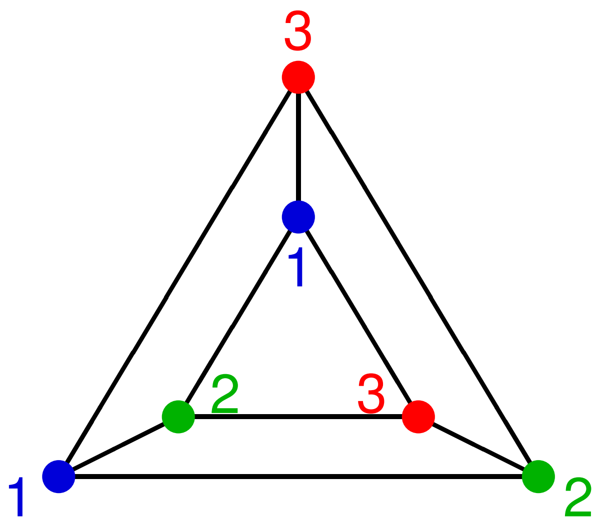
Theorem (Burton Jr. & Henley, 1997; many times since then)

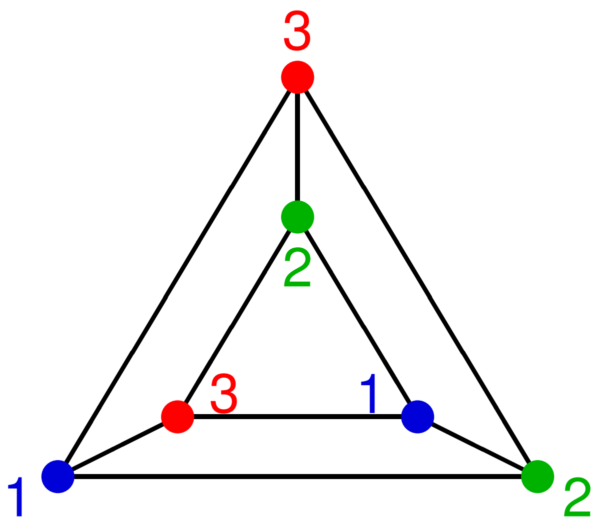
- if G is bipartite, then for any k: any two k-colourings are connected by Kempe chain transformations

Reconfiguration with Kempe chains

- a Kempe chain transformation of a k-colouring of a graph involves:
  - for two colours c_1, c_2,
    swap the colours on a (c_1, c_2)-coloured component

- not true if G is not bipartite:





Reconfiguration with Kempe chains

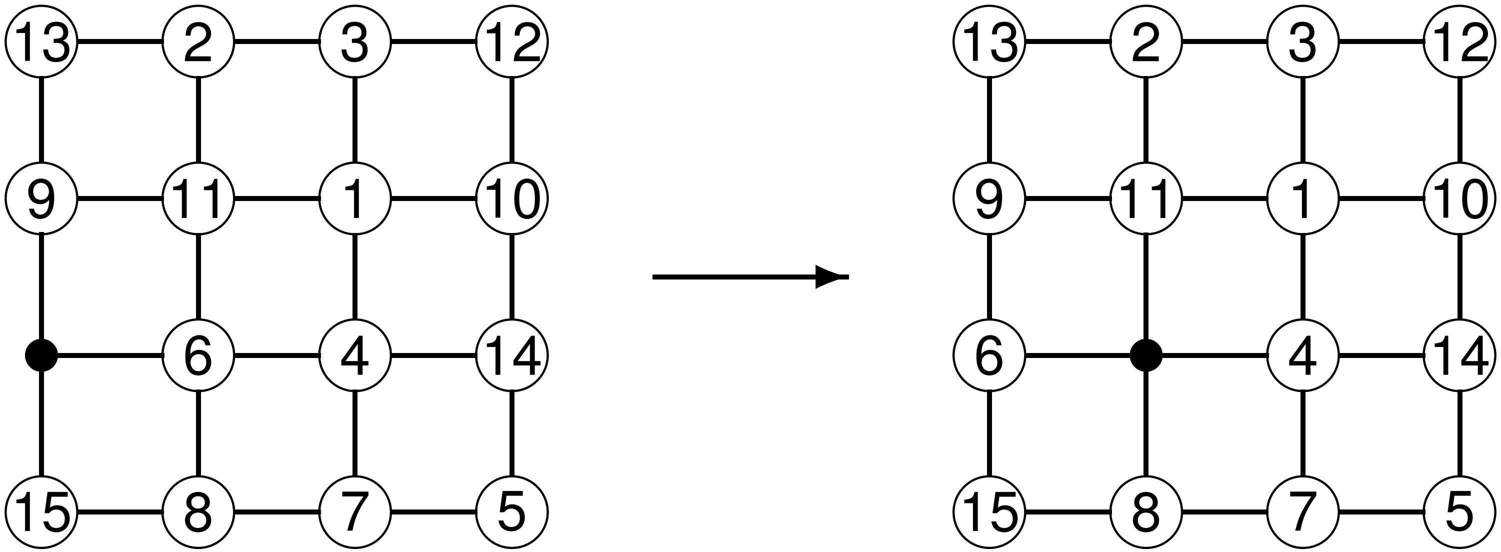
- we know some classes of graphs whose k-colourings are connected using Kempe chains
- **example**:

  planar graphs with $k = 5$ (best possible, not true for $k = 4$)

  (Meyniel, 1978)
- but no results are known about the complexity of deciding this when it is not always "yes",
  - not even for specific graph classes
- the same holds for the "path between colourings" version of the decision problem

Sliding token puzzles

- we can interpret the 15-puzzle as a problem involving moving tokens on a given graph:



Sliding token puzzles

- so what happens if we would play this on other graphs?

- for a given graph G on n vertices,

  define **puz(G)** as the graph that has:

  - nodes: all possible placements of $n - 1$ tokens on G
  - adjacency: sliding one token along an edge of G to an empty vertex

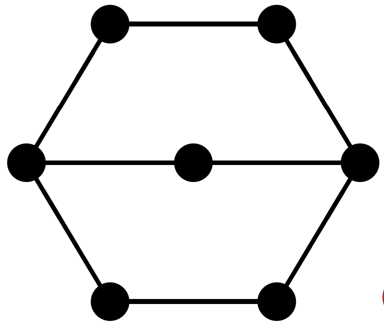
- and our standard decision problems become:

  - are two token configurations in one component of puz(G)?
  - is puz(G) connected?

Sliding token puzzles

Theorem (Wilson, 1974)

- if G is a 2-connected graph, then puz(G) is connected, except if:
  - G is a cycle on $n \geq 4$ vertices
    (then puz(G) has $(n-2)\boldsymbol{!}$ components)
  - G is bipartite different from a cycle
    (then puz(G) has 2 components)
  - G is the exceptional graph $\boldsymbol{\Theta}_0$ (puz($\boldsymbol{\Theta}_0$) has 6 components)



$\boldsymbol{\Theta}_0$

Generalised sliding token puzzles

- **what would happen if**:
  - we have fewer than $n - 1$ tokens (i.e., more empty vertices)?
  - and/or not all tokens are the same?
- so suppose we have a set $(k_1, k_2, \ldots, k_p)$ of labelled tokens
  - meaning: k_1 tokens with label 1, k_2 tokens with label 2, etc.
  - tokens with the same label are indistinguishable
  - we can assume that $k_1 \geq k_2 \geq \cdots \geq k_p$ and their sum is at most $n - 1$
- the corresponding graph of all token configurations on G is denoted by **puz**$(G; k_1, \ldots, k_p)$

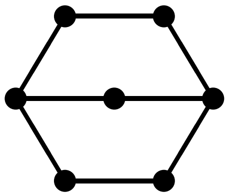
Generalised sliding token puzzles

Theorem (Brightwell, vdH & Trakultraipruk, 2013+)

- G a graph on n vertices, $(k_1, k_2, \dots, k_p)$ a token set,

  then $\mathrm{puz}(G; k_1, \dots, k_p)$ is connected, except if:

  - G is not connected
  - G is a path and $p \geq 2$
  - G is a cycle, and $p \geq 3$, or $p = 2$ and $k_2 \geq 2$
  - G is a 2-connected, bipartite graph with token set $(1^{(n-1)})$
  - G is the exceptional graph Θ_0 with token set $(2, 2, 2)$, $(2, 2, 1, 1)$, $(2, 1, 1, 1, 1)$ or $(1, 1, 1, 1, 1, 1)$



Generalised sliding token puzzles

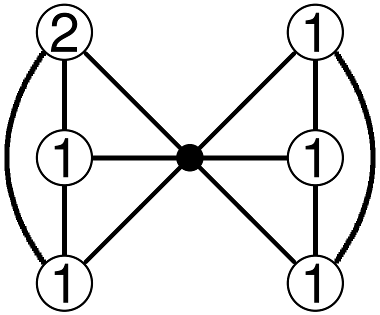
<u>**Theorem**</u> (Brightwell, vdH & Trakultraipruk, 2013+)

- G a graph on n vertices, $(k_1, k_2, \ldots, k_p)$ a token set,
  then $\mathrm{puz}(G; k_1, \ldots, k_p)$ is connected, except if:
  - G is not connected
  - G is a path and $p \geq 2$
  - G is a cycle, and $p \geq 3$, or $p = 2$ and $k_2 \geq 2$
  - G is a 2-connected, bipartite graph with token set $(1^{(n-1)})$
  - G is the exceptional graph $\mathbf{\Theta}_0$ with some bad token sets
  - G has connectivity one, $p \geq 2$ and there is a "separating path preventing tokens from moving between blocks"

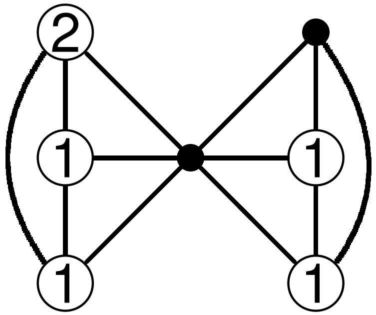
Generalised sliding token puzzles

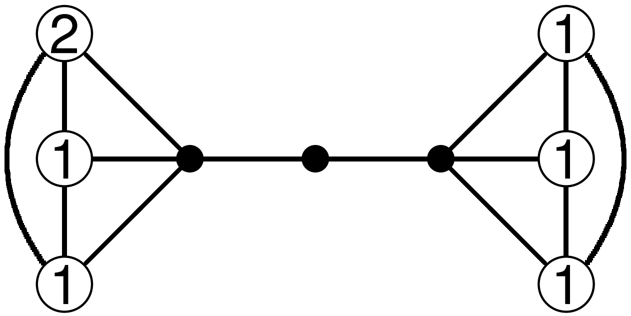
- “separating paths” in graphs of connectivity one:

bad:

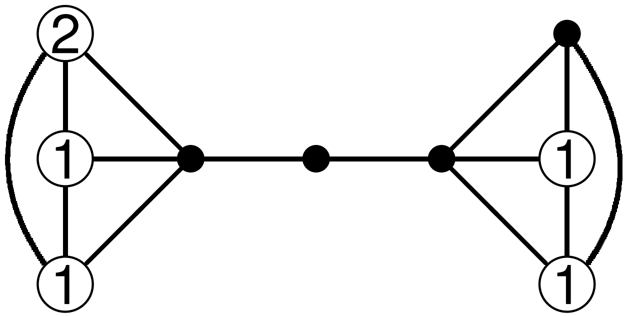


good:





bad



good

Generalised sliding token puzzles

- we can also characterise:
  - given a graph G, token set $(k_1, \ldots, k_p)$, and two token configurations on G,
  - are the two configurations in the same component of $\mathrm{puz}(G; k_1, \ldots, k_p)$?
- so recognising connectivity properties of $\mathrm{puz}(G; k_1, \ldots, k_p)$ is easy
- so can we say something about the number of steps we would need?

The length of sliding token paths

- SHORTEST-A-TO-B-TOKEN-MOVES

  Input: a graph G, a token set $(k_1, \ldots, k_p)$, two token configurations A and B on G, and a positive integer N

  Question: can we go from A to B in at most N steps?

Theorem (Goldreich, 1984-2011)

- restricted to the case that there are $n - 1$ different tokens, SHORTEST-A-TO-B-TOKEN-MOVES is **NP-complete**

The length of sliding token paths

Theorem

- restricted to the case that all tokens are the same,

  SHORTEST-A-TO-B-TOKEN-MOVES is in **P**

sketch of proof

- let $U = \{u_1, \ldots, u_q\}$ be the vertices containing a token in configuration A

  and $V = \{v_1, \ldots, v_q\}$ is that set for configuration B

- form a complete bipartite graph with bipartation $U \cup V$

- give each edge $u_i v_j$ a weight w_{ij} equal to the length of the shortest path from u_i to v_j in G

The length of sliding token paths

sketch of proof

- let $U = \{u_1, \ldots, u_q\}$ be the vertices containing a token in configuration A

  and $V = \{v_1, \ldots, v_q\}$ is that set for configuration B
- form a complete bipartite graph with bipartation $U \cup V$
- give each edge $u_i v_j$ a weight w_{ij} equal to the length of the shortest path from u_i to v_j in G
- find a matching M of minimum weight in the bipartite graph,
  - say it has total weight W
- then you can from A to B in exactly W steps (and not fewer!)
  - although not necessarily by the paths indicated by M !

The length of sliding token paths

- - all tokens different:

    SHORTEST-A-TO-B-TOKEN-MOVES is **NP-complete**

  - all tokens the same:

    SHORTEST-A-TO-B-TOKEN-MOVES is in **P**

- so when does the complexity change?

Theorem (vdH & Trakultraipruk, 2013+)

- restricted to the case that there is just one special token and all others are the same:

  SHORTEST-A-TO-B-TOKEN-MOVES is already **NP-complete**

The length of sliding token paths

- the proof uses ideas of the proof of

Theorem (Papadimitriou, Raghavan, Sudan & Tamaki, 1994)

- SHORTEST-ROBOT-MOTION-WITH-ONE-ROBOT is **NP-complete**

- **Robot Motion** problems on graphs are sliding token problems,
  - with some special tokens (the robots)
    - that have to end in specified positions
  - all other tokens are just obstacles
    - and it is not important where those are at the end

A final puzzle: Rush Hour

- RUSH-HOUR

  Input: some rectangular board,
  a configuration of cars on that board,
  and one special car

  Question: is it possible to get the special car moving?

Theorem

- RUSH-HOUR is **PSPACE-complete** (Flake & Baum, 2002)
- RUSH-HOUR remains **PSPACE-complete** even if all cars have length two (Tromp & Cilibrasi, 2005)

How to prove a decision problem is PSPACE-complete?

standard method:

- reduction to the basic PSPACE-complete problem: QUANTIFIED-SAT:

$$\forall x_1 \, \exists y_1 \, \forall x_2 \, \exists y_2 \, \cdots \, \forall x_n \, \exists y_n \, \varphi(x_1, y_1, \ldots, x_n, y_n)$$

  for some Boolean formula $\varphi(x_1, y_1, \ldots, x_n, y_n)$

- Hearn & Demain (2005) developed an approach that is often much easier to use
  - first step: show that a QUANTIFIED-SAT formula can be represented by certain logical circuits

NCL machines

a **Non-Deterministic Constraint Logic machine**
has the following elements:

- it is an undirected graph,
  with non-negative weights on the vertices and edges
- a feasible configuration is an orientation of the edges,
  such that for each vertex:
  - the sum of the incoming edge-weights
    is at least the weight of the vertex
- a move is reversing the orientation of an edge
  (making sure the new configuration is still feasible)

NCL machines

- NCL-CONFIGURATION-TO-EDGE

  Input: an NCL machine, a feasible configuration, and a special edge of the underlying graph

  Question: is there a sequence of moves that reverses the orientation of the special edge?

Theorem (Hearn & Demaine, 2005)

- NCL-CONFIGURATION-TO-EDGE is **PSPACE-complete**

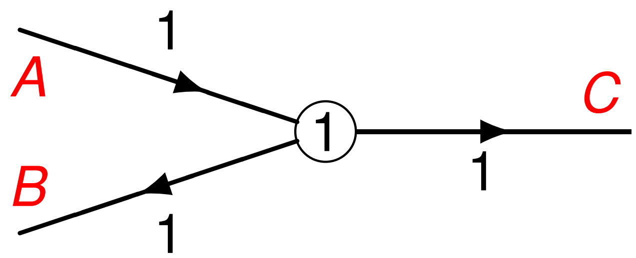
NCL machines

<u>**Theorem**</u> (Hearn & Demaine, 2005)

- NCL-CONFIGURATION-TO-EDGE is **PSPACE-complete**
- even when restricted to NCL machines in which:
  - the underlying graph is planar,
  - all vertices have degree three,
  - all vertices have weight 1 or 2,
  - all edges have weight 1

Restricted NCL machines

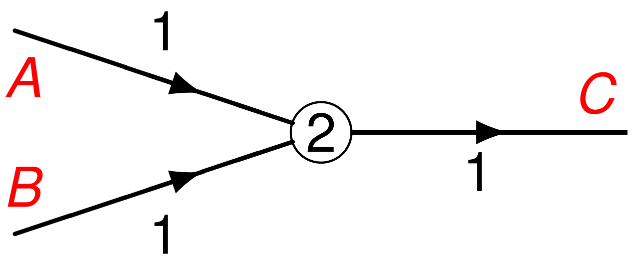
- - all vertices have weight 1 or 2,
  - all edges have weight 1
- vertices with weight 1 give an OR-like behaviour:



edge C can only go outwards, if at least one of A, B goes inwards

Restricted NCL machines

- - all vertices have weight 1 or 2,
  - all edges have weight 1
- vertices with weight 1 give an OR-like behaviour
- vertices with weight 2 give an AND-like behaviour:



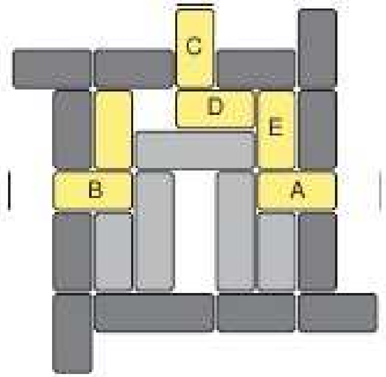
edge C can only go outwards, if both A, B go inwards

Restricted NCL machines

- - all vertices have weight 1 or 2,
  - all edges have weight 1
- vertices with weight 1 give an OR-like behaviour
- vertices with weight 2 give an AND-like behaviour:
- with some care,
  with these elements we can build any logical circuit
  - and that way prove that the restricted NCL-CONFIGURATION-TO-EDGE is **PSPACE-complete**

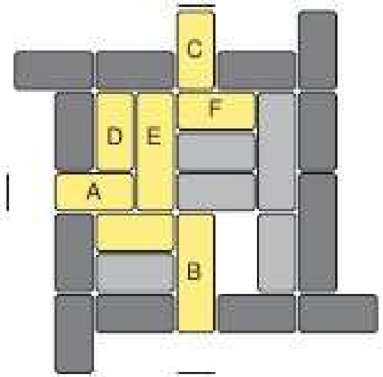
Back to Rush Hour

- an OR-like collection of cars:



C can only move in, if at least one of **A**, **B** moves out

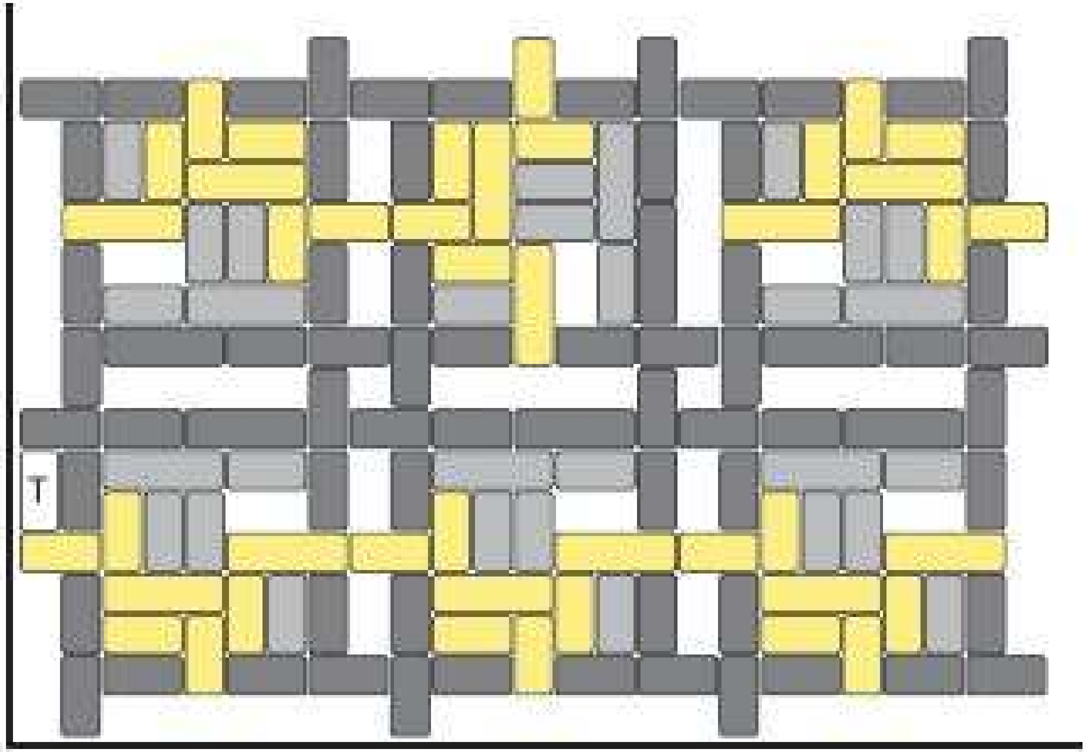
- an AND-like collection of cars:



C can only move in, if both **A** and **B** move out

Back to Rush Hour

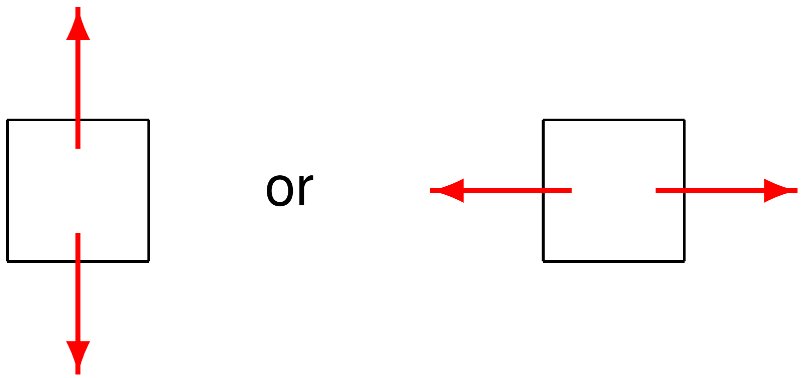
- and then combine it all in big tableaus:



A final open problem

- RUSH-HOUR is **PSPACE-complete**,
  even if all cars have length two

- what is the complexity if all cars have length one?
  - i.e., each car is a 1×1 block,
    but can move in only one direction



Can you move your city car out of the garage?