EITQ

Counting Problems,
Matchgate Quantum Computing

Renaud Vilmart

Homework!

see `https://rvilmart.github.io/ArteQ.html` by the end of the week.

Last Time

- **P**: polynomial-time decision problems

Last Time

- **P**: polynomial-time decision problems
- **NP**: Non-deterministic polynomial-time decision problems

Last Time

- **P**: polynomial-time decision problems
- **NP**: Non-deterministic polynomial-time decision problems
- Polynomial reductions ($\leqslant_P$)

Last Time

- **P**: polynomial-time decision problems
- **NP**: Non-deterministic polynomial-time decision problems
- Polynomial reductions ($\leqslant_P$)
- **NP-hard**: all problems in **NP** reduce to them

Last Time

- **P**: polynomial-time decision problems
- **NP**: Non-deterministic polynomial-time decision problems
- Polynomial reductions ($\leqslant_P$)
- **NP-hard**: all problems in **NP** reduce to them
- **NP-complete**: Both **NP** and **NP-hard**

Last Time

- **P**: polynomial-time decision problems
- **NP**: Non-deterministic polynomial-time decision problems
- Polynomial reductions ($\leqslant_P$)
- **NP-hard**: all problems in **NP** reduce to them
- **NP-complete**: Both **NP** and **NP-hard**
  - SAT, Travelling Salesman, Vertex Cover, 3-Colouring, ...

Last Time

- **P**: polynomial-time decision problems
- **NP**: Non-deterministic polynomial-time decision problems
- Polynomial reductions ($\leqslant_P$)
- **NP-hard**: all problems in **NP** reduce to them
- **NP-complete**: Both **NP** and **NP-hard**
  - SAT, Travelling Salesman, Vertex Cover, 3-Colouring, ...
- $\mathbf{P} \subseteq \begin{array}{c}\mathbf{NP}\\ \mathbf{coNP}\end{array} \subseteq \begin{array}{c}\mathbf{PSPACE}\\ = \\ \mathbf{NPSPACE}\end{array} \subseteq \mathbf{EXPTIME} \subseteq \mathbf{EXPSPACE}$

Last Time

- **P**: polynomial-time decision problems
- **NP**: Non-deterministic polynomial-time decision problems
- Polynomial reductions ($\leqslant_P$)
- **NP-hard**: all problems in **NP** reduce to them
- **NP-complete**: Both **NP** and **NP-hard**
  - SAT, Travelling Salesman, Vertex Cover, 3-Colouring, ...
- $\mathbf{P} \subseteq \begin{matrix}\mathbf{NP} \\ \mathbf{coNP}\end{matrix} \subseteq \begin{matrix}\mathbf{PSPACE} \\ = \\ \mathbf{NPSPACE}\end{matrix} \subseteq \mathbf{EXPTIME} \subseteq \mathbf{EXPSPACE}$
- **PP**: "bad" probabilistic class (contains **NP**)

Last Time

- **P**: polynomial-time decision problems
- **NP**: Non-deterministic polynomial-time decision problems
- Polynomial reductions ($\leqslant_P$)
- **NP-hard**: all problems in **NP** reduce to them
- **NP-complete**: Both **NP** and **NP-hard**
  - SAT, Travelling Salesman, Vertex Cover, 3-Colouring, ...
- $\mathbf{P} \subseteq \begin{array}{c}\mathbf{NP}\\ \mathbf{coNP}\end{array} \subseteq \begin{array}{c}\mathbf{PSPACE}\\ = \\ \mathbf{NPSPACE}\end{array} \subseteq \mathbf{EXPTIME} \subseteq \mathbf{EXPSPACE}$
- **PP**: "bad" probabilistic class (contains **NP**)
- **BPP**: "good" probabilistic class

This Time

- Counting complexity

This Time

- Counting complexity
- **#P**: counting solutions to an **NP** problem

This Time

- Counting complexity
- **#P**: counting solutions to an **NP** problem
- Easy problems with hard counting versions?

This Time

- Counting complexity
- **#P**: counting solutions to an **NP** problem
- Easy problems with hard counting versions?
- (Perfect) matchings on graphs

This Time

- Counting complexity
- **#P**: counting solutions to an **NP** problem
- Easy problems with hard counting versions?
- (Perfect) matchings on graphs
- Planarity

This Time

- Counting complexity
- **#P**: counting solutions to an **NP** problem
- Easy problems with hard counting versions?
- (Perfect) matchings on graphs
- Planarity
- Quantum matchgate computing

This Time

- Counting complexity
- **#P**: counting solutions to an **NP** problem
- Easy problems with hard counting versions?
- (Perfect) matchings on graphs
- Planarity
- Quantum matchgate computing
- Efficiency of planar matchgate simulation

Planar Graphs

- Graph with no edge crossing

Planar Graphs

- Graph with no edge crossing
- Very natural notion

Planar Graphs

- Graph with no edge crossing
- Very natural notion
- Good for visualisation

Planar Graphs

- Graph with no edge crossing
- Very natural notion
- Good for visualisation
- Connection between topology and graph property

Planar Graphs

- Graph with no edge crossing
- Very natural notion
- Good for visualisation
- Connection between topology and graph property
- Additional structure on graph

Planar Graphs

- Graph with no edge crossing
- Very natural notion
- Good for visualisation
- Connection between topology and graph property
- Additional structure on graph
  - Every planar graph is 4-colourable

Planar Graphs

- Graph with no edge crossing
- Very natural notion
- Good for visualisation
- Connection between topology and graph property
- Additional structure on graph
  - Every planar graph is 4-colourable
  - Every clique in a planar graph has size ≤ 4

Planar Graphs

- Graph with no edge crossing
- Very natural notion
- Good for visualisation
- Connection between topology and graph property
- Additional structure on graph
  - Every planar graph is 4-colourable
  - Every clique in a planar graph has size ≤ 4
- Can simplify problems

Planar Graphs

- Graph with no edge crossing
- Very natural notion
- Good for visualisation
- Connection between topology and graph property
- Additional structure on graph
  - Every planar graph is 4-colourable
  - Every clique in a planar graph has size ≤ 4
- Can simplify problems
  - Minimum feedback arc set problem becomes **P**

Planar Graphs

- Graph with no edge crossing
- Very natural notion
- Good for visualisation
- Connection between topology and graph property
- Additional structure on graph
  - Every planar graph is 4-colourable
  - Every clique in a planar graph has size ≤ 4
- Can simplify problems
  - Minimum feedback arc set problem becomes **P**
  - Perfect matchings

Graph Minor

Recall: graph $G = (V, E)$ with $E \subseteq V \times V$

Definition

Graph Minor H minor of G if obtained by:

- deleting vertices
- deleting edges
- contracting edges

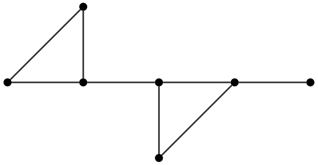
Graph Minor

Recall: graph $G = (V, E)$ with $E \subseteq V \times V$

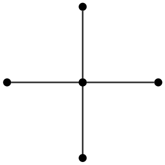
Definition

Graph Minor H minor of G if obtained by:

- deleting vertices
- deleting edges
- contracting edges



G



H

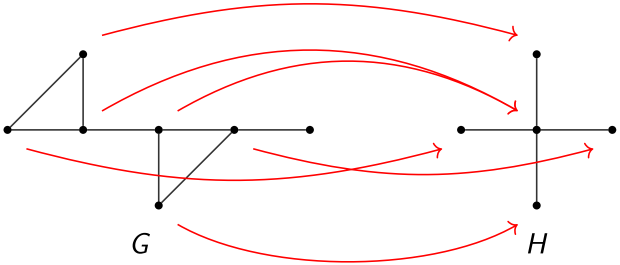
Graph Minor

Recall: graph $G = (V, E)$ with $E \subseteq V \times V$

Definition

Graph Minor H minor of G if obtained by:

- deleting vertices
- deleting edges
- contracting edges



Planarity: K_5 and $K_{3,3}$

Definition (Planar Graph)

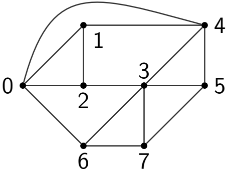
A graph that can be embedded on the plane, without crossings of edges.

Planarity: K_5 and $K_{3,3}$

Definition (Planar Graph)

A graph that can be embedded on the plane, without crossings of edges.

Can be given as a set of faces (ordered list of vertices).

E.g. 

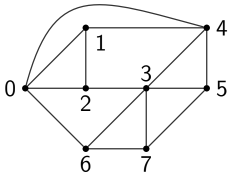
$$\{(0,1,2),(1,4,3,2),(3,4,5),(3,5,7),\\(3,7,6),(0,2,3,6),(0,4,1)\}$$

Planarity: K_5 and $K_{3,3}$

Definition (Planar Graph)

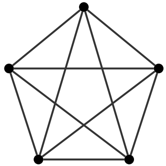
A graph that can be embedded on the plane, without crossings of edges.

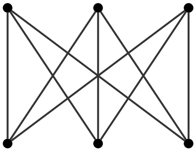
Can be given as a set of faces (ordered list of vertices).

E.g. 

$$\{(0,1,2),(1,4,3,2),(3,4,5),(3,5,7),(3,7,6),(0,2,3,6),(0,4,1)\}$$

Two very important non-planar graphs:

$K_5 :=$ 

$K_{3,3} :=$ 

Deciding Planarity

Theorem (Planarity)

A graph G is planar iff none of its minors is K_5 or $K_{3,3}$.

Deciding Planarity

Theorem (Planarity)

A graph G is planar iff none of its minors is K_5 or $K_{3,3}$.

- Checking if $H = (V_H, E_H)$ is a minor of $G = (V_G, E_G)$ is NP-complete

Deciding Planarity

Theorem (Planarity)

A graph G is planar iff none of its minors is K_5 or $K_{3,3}$.

- Checking if $H = (V_H, E_H)$ is a minor of $G = (V_G, E_G)$ is NP-complete
- If H is fixed, $\mathcal{O}(|V|^2)$

Deciding Planarity

Theorem (Planarity)

A graph G is planar iff none of its minors is K_5 or $K_{3,3}$.

- Checking if $H = (V_H, E_H)$ is a minor of $G = (V_G, E_G)$ is NP-complete
- If H is fixed, $\mathcal{O}(|V|^2)$
  - however $2 \uparrow\uparrow (2 \uparrow\uparrow (2 \uparrow\uparrow (|V_H|/2)))$ constant

Deciding Planarity

Theorem (Planarity)

A graph G is planar iff none of its minors is K_5 or $K_{3,3}$.

- Checking if $H = (V_H, E_H)$ is a minor of $G = (V_G, E_G)$ is NP-complete
- If H is fixed, $\mathcal{O}(|V|^2)$
  - however $2 \uparrow\uparrow (2 \uparrow\uparrow (2 \uparrow\uparrow (|V_H|/2)))$ constant
- Linear-time algorithms exist for planarity!

Deciding Planarity

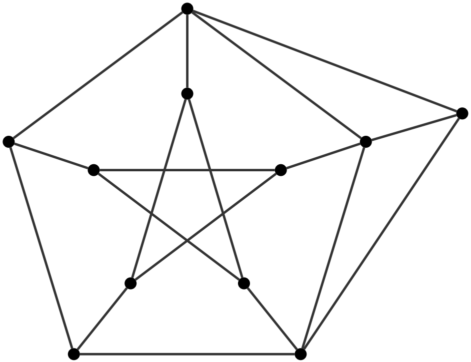
Theorem (Planarity)

A graph G is planar iff none of its minors is K_5 or $K_{3,3}$.

- Checking if $H = (V_H, E_H)$ is a minor of $G = (V_G, E_G)$ is NP-complete
- If H is fixed, $\mathcal{O}(|V|^2)$
  - however $2 \uparrow\uparrow (2 \uparrow\uparrow (2 \uparrow\uparrow (|V_H|/2)))$ constant
- Linear-time algorithms exist for planarity!
  - Planarity is **P**.

Example: Planarity?

Is the following graph planar?

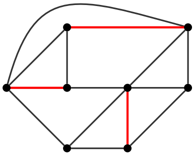


Matchings and Perfect Matchings

Definition

With $G = (V, E)$ a graph:

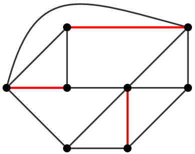
- Matching: $M \subseteq E$, s.t. $\forall e_i, e_j \in M,\ i \neq j \implies e_i \cap e_j = \emptyset$.



Matchings and Perfect Matchings

Definition

With $G = (V, E)$ a graph:

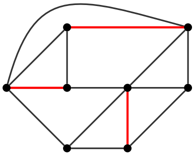


- Matching: $M \subseteq E$, s.t. $\forall e_i, e_j \in M,\ i \neq j \implies e_i \cap e_j = \emptyset$.
- Maximum cardinality matching: Matching that maximises the number of covered vertices.

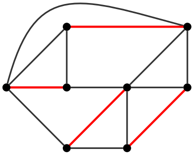
Matchings and Perfect Matchings

Definition

With $G = (V, E)$ a graph:

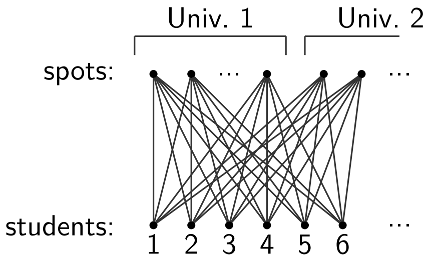


- Matching: $M \subseteq E$, s.t. $\forall e_i, e_j \in M,\ i \neq j \implies e_i \cap e_j = \emptyset$.
- Maximum cardinality matching: Matching that maximises the number of covered vertices.



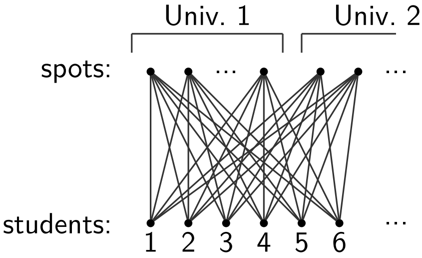
- Perfect Matching: A matching M that contains all vertices, i.e. $\forall v \in V, \exists e \in M, v \in e$.

Example: Parcoursup



Aparté:

Example: Parcoursup



Aparté:

Particular case of a bipartite graph.

Suppose $|students| < |spots|$. A maximum cardinality matching covers all students iff $\forall W \subseteq \text{students},\ |W| \leq |N(W)|$.

Proof of above gives an algorithm.

Augmenting Matchings

Given a matching M of graph $G = (V, E)$:

- Alternating path: Path that alternates between edges in M and edges not in M

Augmenting Matchings

Given a matching M of graph $G = (V, E)$:

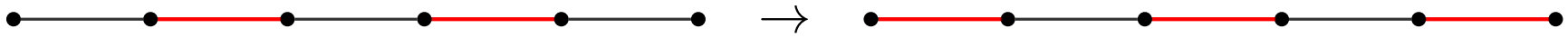
- Alternating path: Path that alternates between edges in M and edges not in M
- Augmenting path: Alternating path p with both endpoints not covered by M

Augmenting Matchings

Given a matching M of graph $G = (V, E)$:

- Alternating path: Path that alternates between edges in M and edges not in M
- Augmenting path: Alternating path p with both endpoints not covered by M

- Augmenting a matching: $M \oplus p$

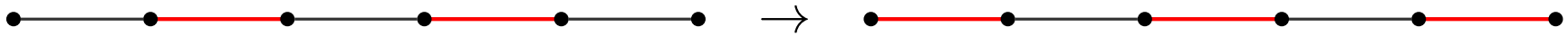


Augmenting Matchings

Given a matching M of graph $G = (V, E)$:

- Alternating path: Path that alternates between edges in M and edges not in M
- Augmenting path: Alternating path p with both endpoints not covered by M

- Augmenting a matching: $M \oplus p$



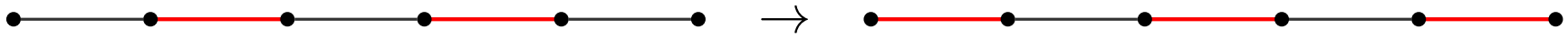
- Result 1: M is maximum iff there is no augmenting path

Augmenting Matchings

Given a matching M of graph $G = (V, E)$:

- Alternating path: Path that alternates between edges in M and edges not in M
- Augmenting path: Alternating path p with both endpoints not covered by M

- Augmenting a matching: $M \oplus p$



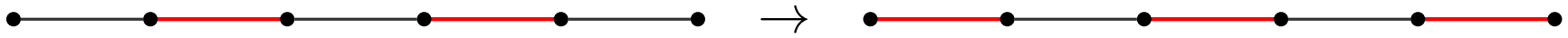
- Result 1: M is maximum iff there is no augmenting path
- Result 2: we can find augmenting paths and augment M efficiently

Augmenting Matchings

Given a matching M of graph $G = (V, E)$:

- Alternating path: Path that alternates between edges in M and edges not in M
- Augmenting path: Alternating path p with both endpoints not covered by M

- Augmenting a matching: $M \oplus p$



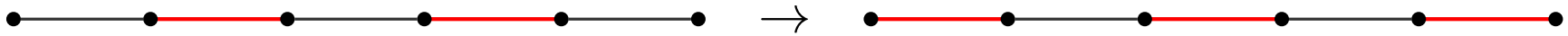
- Result 1: M is maximum iff there is no augmenting path
- Result 2: we can find augmenting paths and augment M efficiently
  - Finding a maximum matching is **P**

Augmenting Matchings

Given a matching M of graph $G = (V, E)$:

- Alternating path: Path that alternates between edges in M and edges not in M
- Augmenting path: Alternating path p with both endpoints not covered by M

- Augmenting a matching: $M \oplus p$



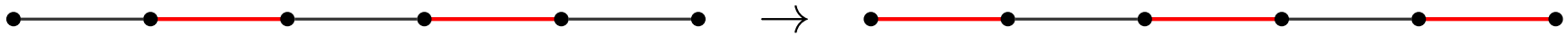
- Result 1: M is maximum iff there is no augmenting path
- Result 2: we can find augmenting paths and augment M efficiently
  - Finding a maximum matching is **P**
- Result 3: A maximum matching is a perfect matching iff G has perfect matchings

Augmenting Matchings

Given a matching M of graph $G = (V, E)$:

- Alternating path: Path that alternates between edges in M and edges not in M
- Augmenting path: Alternating path p with both endpoints not covered by M

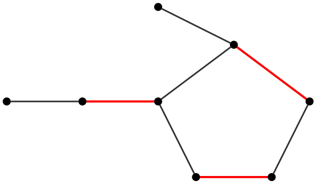
- Augmenting a matching: $M \oplus p$



- Result 1: M is maximum iff there is no augmenting path
- Result 2: we can find augmenting paths and augment M efficiently
  - Finding a maximum matching is **P**
- Result 3: A maximum matching is a perfect matching iff G has perfect matchings
  - Deciding the existence of a perfect matching is **P**

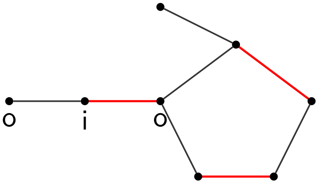
Finding Augmenting Path

Problem with naive depth-first search:



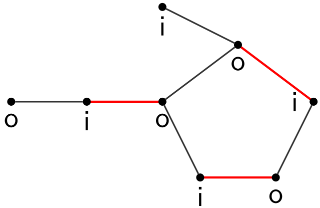
Finding Augmenting Path

Problem with naive depth-first search:



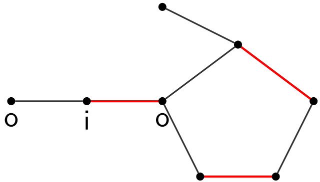
Finding Augmenting Path

Problem with naive depth-first search:



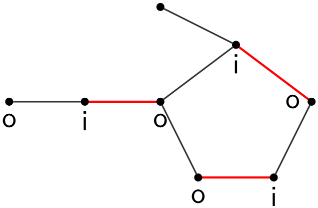
Finding Augmenting Path

Problem with naive depth-first search:



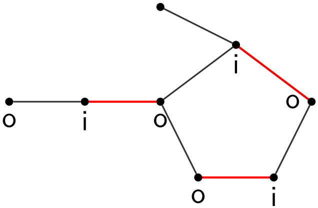
Finding Augmenting Path

Problem with naive depth-first search:



Finding Augmenting Path

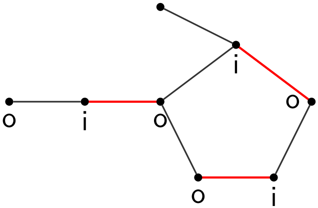
Problem with naive depth-first search:



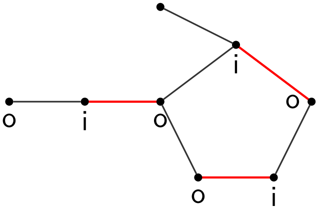
Solution: When marking vertex with “o”, check if some neighbours were already marked as “o”. If yes, contract whole part of path between the two vertices:

Finding Augmenting Path

Problem with naive depth-first search:

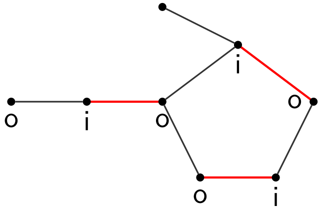


Solution: When marking vertex with “o”, check if some neighbours were already marked as “o”. If yes, contract whole part of path between the two vertices:

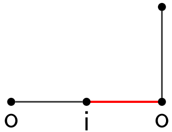


Finding Augmenting Path

Problem with naive depth-first search:

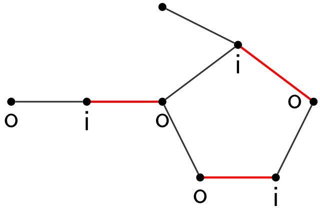


Solution: When marking vertex with "o", check if some neighbours were already marked as "o". If yes, contract whole part of path between the two vertices:

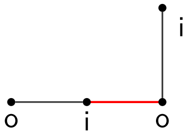


Finding Augmenting Path

Problem with naive depth-first search:



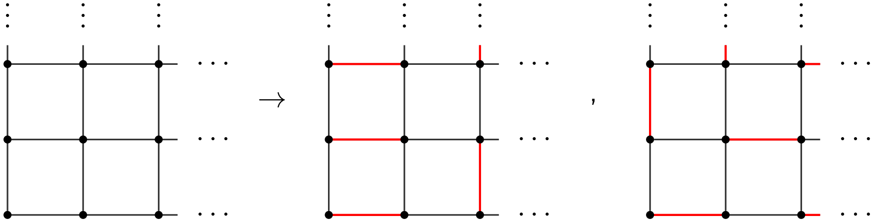
Solution: When marking vertex with "o", check if some neighbours were already marked as "o". If yes, contract whole part of path between the two vertices:



Combinatorics of Perfect Matchings

Original problem:

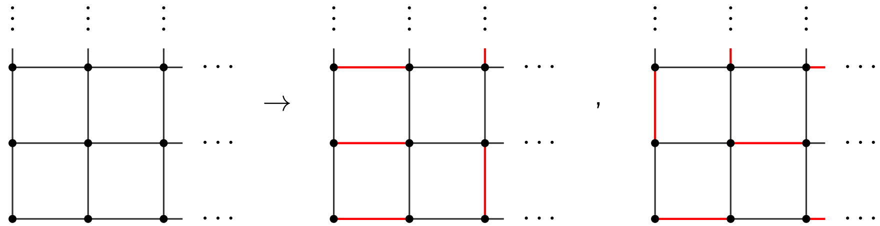
- Molecules laid out on a lattice. How many ways to "dimerise" them?



Combinatorics of Perfect Matchings

Original problem:

- Molecules laid out on a lattice. How many ways to "dimerise" them?

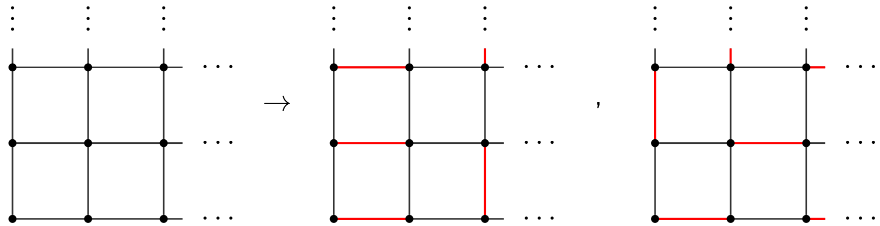


- $\equiv$ counting perfect matchings in the lattice.

Combinatorics of Perfect Matchings

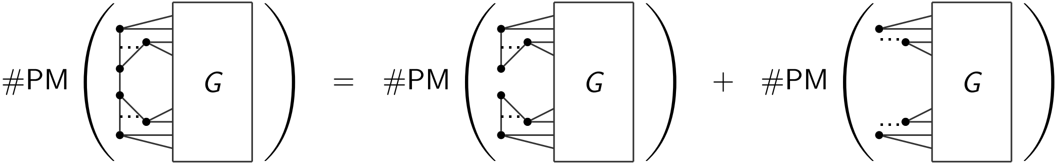
Original problem:

- Molecules laid out on a lattice. How many ways to "dimerise" them?

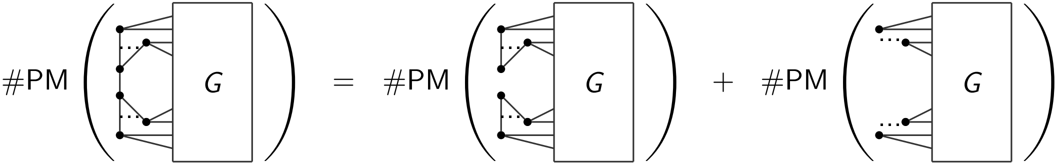


- $\equiv$ counting perfect matchings in the lattice.
- Requires studying complexity of <u>counting problems</u>

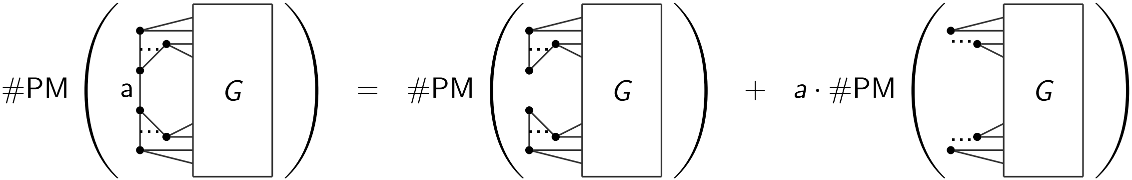
A Natural Way to Count Perfect Matchings



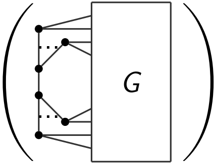
A Natural Way to Count Perfect Matchings

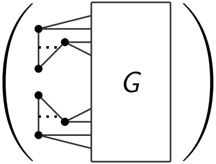


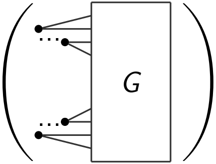
With weights on edges ($w : E \to R$):



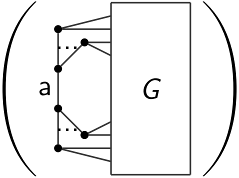
A Natural Way to Count Perfect Matchings

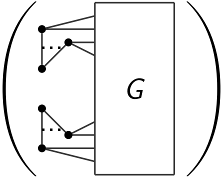
#PM 

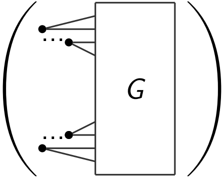
= #PM 

+ #PM 

With weights on edges ($w : E \to R$):

#PM 

= #PM 

+ $a \cdot$ #PM 

$\#\mathrm{PM}_R$: counting perfect matchings with weights in R.

Complexity of Counting Problems

- **#P** problems: functions $f : \{0,1\}^* \rightarrow \mathbb{N}$ such that there exists a polynomial-time non-deterministic Turing machine M such that $f(x)$ is the number of accepting paths in M's computation of x.

Complexity of Counting Problems

- **#P** problems: functions $f : \{0,1\}^* \to \mathbb{N}$ such that there exists a polynomial-time non-deterministic Turing machine M such that $f(x)$ is the number of accepting paths in M's computation of x.
  - number of occurrences of element x in a list l
  - #SAT: number of solutions to a SAT problem.
  - #TS: number of itineraries shorter than t
  - #PM: number of perfect matchings
  - PERM: computing the permanent of a $\{0,1\}$-matrix

Complexity of Counting Problems

- **#P** problems: functions $f : \{0, 1\}^* \rightarrow \mathbb{N}$ such that there exists a polynomial-time non-deterministic Turing machine M such that $f(x)$ is the number of accepting paths in M's computation of x.
  - number of occurrences of element x in a list l
  - #SAT: number of solutions to a SAT problem.
  - #TS: number of itineraries shorter than t
  - #PM: number of perfect matchings
  - PERM: computing the permanent of a $\{0, 1\}$-matrix

  At least as hard as **NP**.

Complexity of Counting Problems

- **#P** problems: functions $f : \{0,1\}^* \to \mathbb{N}$ such that there exists a polynomial-time non-deterministic Turing machine M such that $f(x)$ is the number of accepting paths in M's computation of x.
  - number of occurrences of element x in a list l
  - $\#$SAT: number of solutions to a SAT problem.
  - $\#$TS: number of itineraries shorter than t
  - $\#$PM: number of perfect matchings
  - PERM: computing the permanent of a $\{0,1\}$-matrix

  At least as hard as **NP**.

- **FP** problems: function problems solvable by a polynomial-time deterministic Turing machine
  - number of occurrences of element x in a list l

Complexity of Counting Problems

- **#P** problems: functions $f : \{0,1\}^* \to \mathbb{N}$ such that there exists a polynomial-time non-deterministic Turing machine M such that $f(x)$ is the number of accepting paths in M's computation of x.
  - number of occurrences of element x in a list l
  - #SAT: number of solutions to a SAT problem.
  - #TS: number of itineraries shorter than t
  - #PM: number of perfect matchings
  - PERM: computing the permanent of a $\{0,1\}$-matrix

  At least as hard as **NP**.

- **FP** problems: function problems solvable by a polynomial-time deterministic Turing machine
  - number of occurrences of element x in a list l

- **#P-complete**: **#P** problems such that for all other **#P** problems f, there is a polynomial-time counting reduction ($\preccurlyeq_{\#}$) from f to that problem

Polynomial-Time Counting Reduction

Given two problems X and Y, we say that X reduces to Y ($X \leqslant_{\#} Y$) if there exist polynomial-time computable functions f and g such that the following diagram commutes:

$$\begin{tikzcd} {} \arrow[r, "X"] \arrow[d, "f"'] & {} \\ {} \arrow[r, "Y"] & {} \arrow[u, "g"'] \end{tikzcd}$$

Polynomial-Time Counting Reduction

Given two problems X and Y, we say that X reduces to Y ($X \preccurlyeq_{\#} Y$) if there exist polynomial-time computable functions f and g such that the following diagram commutes:

$$\begin{tikzcd} \cdot \arrow[r, "X"] \arrow[d, "f"'] & \cdot \\ \cdot \arrow[r, "Y"] & \cdot \arrow[u, "g"'] \end{tikzcd}$$

When X and Y are functions, this means $X = g \circ Y \circ f$.
(In general X and Y need not be functions.)

Polynomial-Time Counting Reduction

Given two problems X and Y, we say that X reduces to Y ($X \preccurlyeq_{\#} Y$) if there exist polynomial-time computable functions f and g such that the following diagram commutes:

$$\begin{tikzcd} \cdot \arrow[r, "X"] \arrow[d, "f"'] & \cdot \\ \cdot \arrow[r, "Y"] & \cdot \arrow[u, "g"'] \end{tikzcd}$$

When X and Y are functions, this means $X = g \circ Y \circ f$.
(In general X and Y need not be functions.)

$\#$3-SAT is **#P-complete**: the Cook-Levin construction preserves the number of solutions (i.e. $g = id$)

Proving #PM is **#P-complete**

- #SAT $\preccurlyeq_{\#}$ PERM$_{\{-1,0,1,2,3\}}$
  - link between cycle covers and permanent

Proving #PM is **#P-complete**

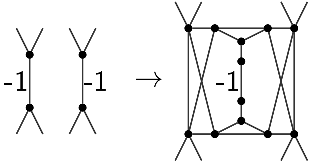
- $\#\text{SAT} \leqslant_{\#} \text{PERM}_{\{-1,0,1,2,3\}}$
  - link between cycle covers and permanent
- $\text{PERM}_{\{-1,0,1,2,3\}} \leqslant_{\#} \#\text{PM}_{\{-1,1,2,3\}}$
  - link between perfect matchings and permanent

Proving #PM is **#P-complete**

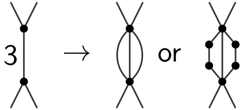
- #SAT $\preccurlyeq_{\#}$ $\mathrm{PERM}_{\{-1,0,1,2,3\}}$
  - link between cycle covers and permanent
- $\mathrm{PERM}_{\{-1,0,1,2,3\}} \preccurlyeq_{\#} \#\mathrm{PM}_{\{-1,1,2,3\}}$
  - link between perfect matchings and permanent
- $\#\mathrm{PM}_{\{-1,1,2,3\}} \preccurlyeq_{\#} \#\mathrm{PM}_{\{1,2,3\}}$

Proving #PM is **#P-complete**

- $\#\mathsf{SAT} \leqslant_{\#} \mathsf{PERM}_{\{-1,0,1,2,3\}}$
  - link between cycle covers and permanent
- $\mathsf{PERM}_{\{-1,0,1,2,3\}} \leqslant_{\#} \#\mathsf{PM}_{\{-1,1,2,3\}}$
  - link between perfect matchings and permanent
- $\#\mathsf{PM}_{\{-1,1,2,3\}} \leqslant_{\#} \#\mathsf{PM}_{\{1,2,3\}}$

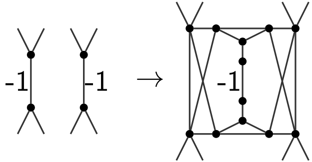


- $\#\mathsf{PM}_{\{1,2,3\}} \leqslant_{\#} \#\mathsf{PM}$

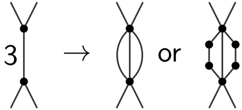


Proving #PM is **#P-complete**

- $\#\mathrm{SAT} \leqslant_{\#} \mathrm{PERM}_{\{-1,0,1,2,3\}}$
  - link between cycle covers and permanent
- $\mathrm{PERM}_{\{-1,0,1,2,3\}} \leqslant_{\#} \#\mathrm{PM}_{\{-1,1,2,3\}}$
  - link between perfect matchings and permanent
- $\#\mathrm{PM}_{\{-1,1,2,3\}} \leqslant_{\#} \#\mathrm{PM}_{\{1,2,3\}}$



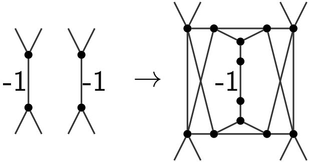
- $\#\mathrm{PM}_{\{1,2,3\}} \leqslant_{\#} \#\mathrm{PM}$



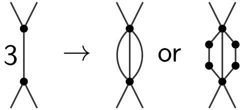
More details in the TD sheet.

Proving #PM is **#P-complete**

- $\#\text{SAT} \leqslant_{\#} \text{PERM}_{\{-1,0,1,2,3\}}$
  - link between cycle covers and permanent
- $\text{PERM}_{\{-1,0,1,2,3\}} \leqslant_{\#} \#\text{PM}_{\{-1,1,2,3\}}$
  - link between perfect matchings and permanent
- $\#\text{PM}_{\{-1,1,2,3\}} \leqslant_{\#} \#\text{PM}_{\{1,2,3\}}$



- $\#\text{PM}_{\{1,2,3\}} \leqslant_{\#} \#\text{PM}$



More details in the TD sheet.

While PM $\in$ **P**, #PM $\in$ **#P-complete**! (other example: #2SAT)

Planarity in Perfect Matchings

- Recall #PM $\in$ **#P-complete**

Planarity in Perfect Matchings

- Recall $\#\mathrm{PM} \in$ **#P-complete**
- However $\#\mathrm{PM}_{\mathsf{planar}} \in$ **FP**!

Planarity in Perfect Matchings

- Recall $\#\text{PM} \in$ **#P-complete**
- However $\#\text{PM}_{\text{planar}} \in$ **FP**!
  - FKT algorithm

Planarity in Perfect Matchings

- Recall $\#\mathrm{PM} \in$ **#P-complete**
- However $\#\mathrm{PM}_{\mathrm{planar}} \in$ **FP**!
  - FKT algorithm
    - Proof of validity and algo itself are non-trivial

Planarity in Perfect Matchings

- Recall $\#\mathrm{PM} \in$ **#P-complete**
- However $\#\mathrm{PM}_{\mathsf{planar}} \in \mathbf{FP}$!
  - FKT algorithm
    - Proof of validity and algo itself are non-trivial
  - Rewriting of tensors

Planarity in Perfect Matchings

- Recall $\#\text{PM} \in$ **#P-complete**
- However $\#\text{PM}_{\text{planar}} \in$ **FP**!
  - FKT algorithm
    - Proof of validity and algo itself are non-trivial
  - Rewriting of tensors
- (Planar) matchgates: family of quantum circuits efficiently simulable

Tensors in Matchgates

Everything with 2-dimensional indices:

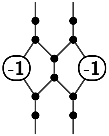
$$\left(\textcircled{r} \right)^i_j = r^i \delta^i_j = \begin{cases} 1 & \text{if } i = j = 0 \\ r & \text{if } i = j = 1 \\ 0 & \text{if } i \neq j \end{cases}$$

$$\left(\bullet \right)^{i_1, \dots, i_n}_{j_1, \dots, j_m} = \begin{cases} 1 & \text{if } \sum_k i_k + \sum_k j_k = 1 \\ 0 & \text{otherwise} \end{cases}$$

Tensors in Matchgates

Everything with 2-dimensional indices:

$$\left(\; (r) \; \right)^i_j = r^i \delta^i_j = \begin{cases} 1 & \text{if } i = j = 0 \\ r & \text{if } i = j = 1 \\ 0 & \text{if } i \neq j \end{cases} \qquad \left(\; \bullet \; \right)^{i_1,\ldots,i_n}_{j_1,\ldots,j_m} = \begin{cases} 1 & \text{if } \sum_k i_k + \sum_k j_k = 1 \\ 0 & \text{otherwise} \end{cases}$$

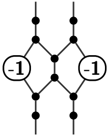
 $=$  $: |i,j\rangle \mapsto (-1)^{ij} |j,i\rangle$ i.e. $\Big($  $\Big)^{a,b}_{c,d} = (-1)^{ab} \delta^a_d \delta^b_c$

Tensors in Matchgates

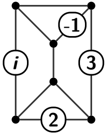
Everything with 2-dimensional indices:

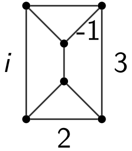
$$\left(\ \textcircled{r}\ \right)^i_j = r^i\delta^i_j = \begin{cases}1 & \text{if } i=j=0\\ r & \text{if } i=j=1\\ 0 & \text{if } i\neq j\end{cases}$$

$$\left(\ \bullet\ \right)^{i_1,\dots,i_n}_{j_1,\dots,j_m} = \begin{cases}1 & \text{if } \sum_k i_k + \sum_k j_k = 1\\ 0 & \text{otherwise}\end{cases}$$

 = 

$: |i,j\rangle \mapsto (-1)^{ij}\,|j,i\rangle$ i.e. $\Big($  $\Big)^{a,b}_{c,d} = (-1)^{ab}\delta^a_d\delta^b_c$



computes the number of perfect matchings in 

.

Matchgate Computations

Let $\quad A = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix}, \qquad B = \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix}$

- "Matchgate:" $G(A, B) = \begin{pmatrix} a_{00} & 0 & 0 & a_{01} \\ 0 & b_{00} & b_{01} & 0 \\ 0 & b_{10} & b_{11} & 0 \\ a_{10} & 0 & 0 & a_{11} \end{pmatrix}$

Matchgate Computations

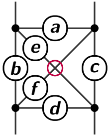
Let $$A = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix}, \qquad B = \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix}$$

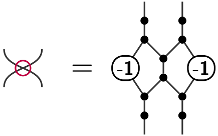
- "Matchgate:" $G(A, B) = \begin{pmatrix} a_{00} & 0 & 0 & a_{01} \\ 0 & b_{00} & b_{01} & 0 \\ 0 & b_{10} & b_{11} & 0 \\ a_{10} & 0 & 0 & a_{11} \end{pmatrix}$
- If $\det(A) = \det(B)$ then $G(A, B)$ is a **"planar"** matchgate

Matchgate Computations

Let $\quad A = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix}, \qquad B = \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix}$

- "Matchgate:" $G(A, B) = \begin{pmatrix} a_{00} & 0 & 0 & a_{01} \\ 0 & b_{00} & b_{01} & 0 \\ 0 & b_{10} & b_{11} & 0 \\ a_{10} & 0 & 0 & a_{11} \end{pmatrix}$
- If $\det(A) = \det(B)$ then $G(A, B)$ is a **"planar"** matchgate

planar matchgate: 

with 

Simulating Matchgates

Statement of the problem:

- Parameters:
  - description of a (matchgate) circuit $\mathcal{C}$
  - number i

Simulating Matchgates

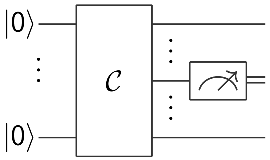
Statement of the problem:

- Parameters:
  - description of a (matchgate) circuit $\mathcal{C}$
  - number i
- Output:
  - the probability of measuring 0 on qubit i after applying circuit $\mathcal{C}$ on state $|0...0\rangle$

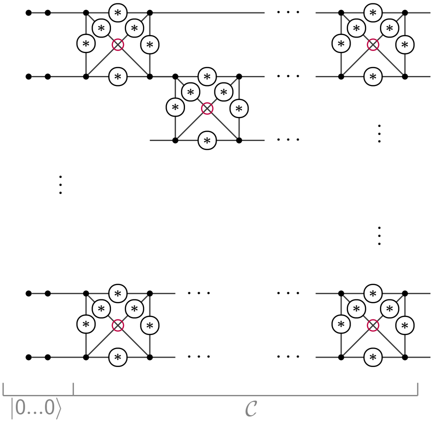
Simulating Matchgates

Statement of the problem:

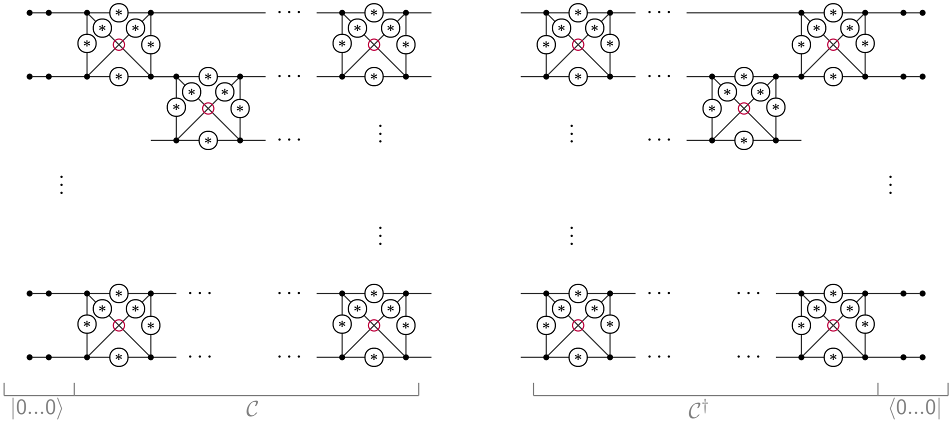
- Parameters:
  - description of a (matchgate) circuit $\mathcal{C}$
  - number i
- Output:
  - the probability of measuring 0 on qubit i after applying circuit $\mathcal{C}$ on state $|0...0\rangle$



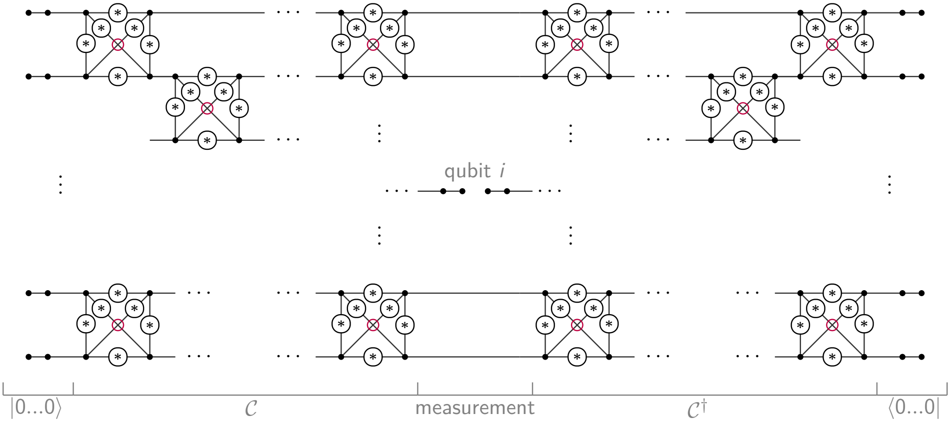
Simulating Matchgates with Tensors



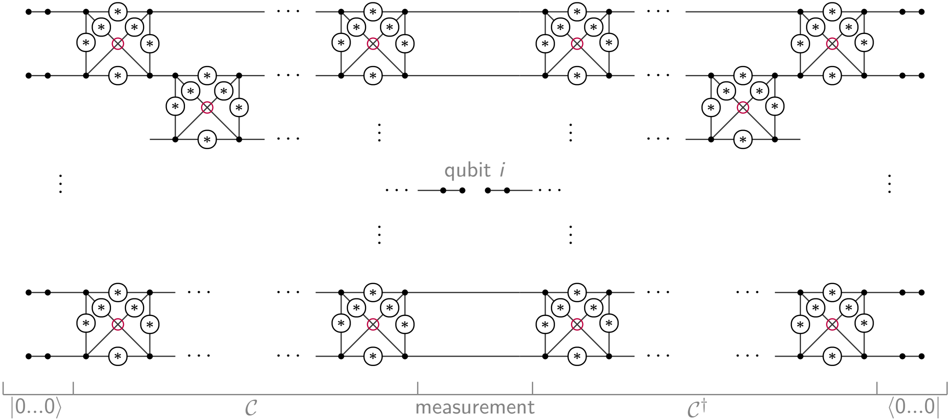
Simulating Matchgates with Tensors



Simulating Matchgates with Tensors



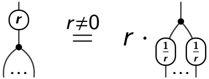
Simulating Matchgates with Tensors

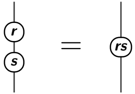


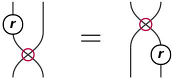
Interpretation in $\mathcal{M}_{1\times1}(\mathbb{C}) \cong \mathbb{C}$: a scalar, the probability we are looking for!
Alternatively: graph with complex edge weights.

Simulating Planar Matchgates via Rewriting

- Weights can be moved around

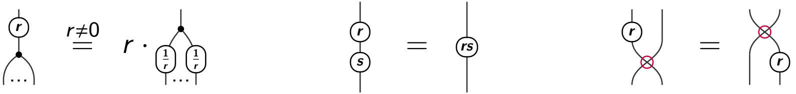






Simulating Planar Matchgates via Rewriting

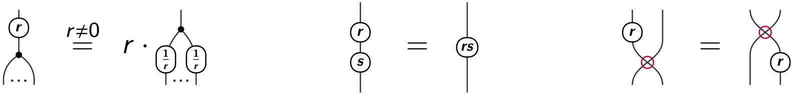
- Weights can be moved around



- Fermionic swaps  are dealt with like swaps

Simulating Planar Matchgates via Rewriting

- Weights can be moved around

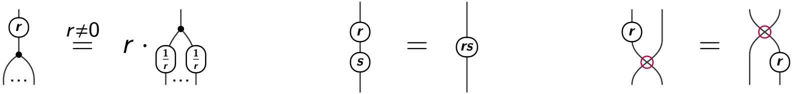


- Fermionic swaps  are dealt with like swaps

- Few rewrites

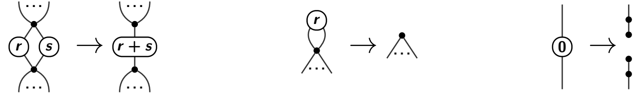
Simulating Planar Matchgates via Rewriting

- Weights can be moved around



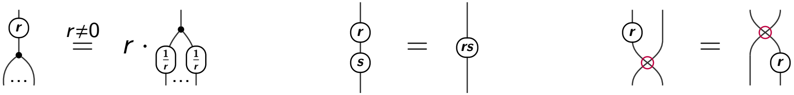
- Fermionic swaps  are dealt with like swaps

- Few rewrites



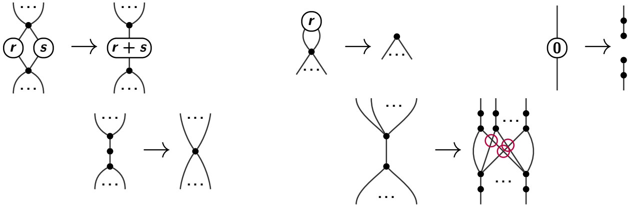
Simulating Planar Matchgates via Rewriting

- Weights can be moved around

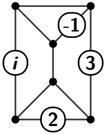


- Fermionic swaps  are dealt with like swaps

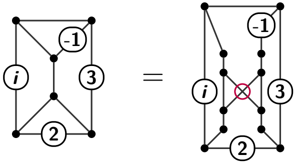
- Few rewrites



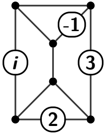
Example



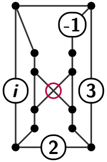
Example



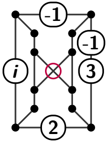
Example



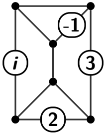
$$=$$



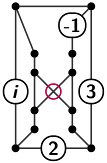
$$= -1 \cdot$$



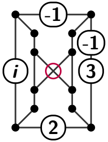
Example



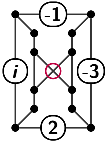
$=$



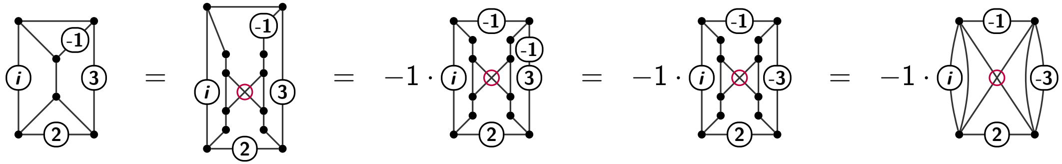
$= \; -1 \cdot$



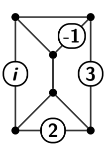
$= \; -1 \cdot$



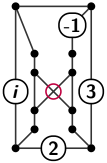
Example



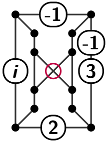
Example



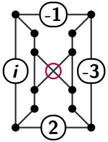
$=$



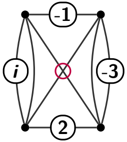
$= -1 \cdot$



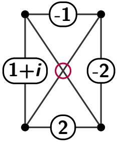
$= -1 \cdot$



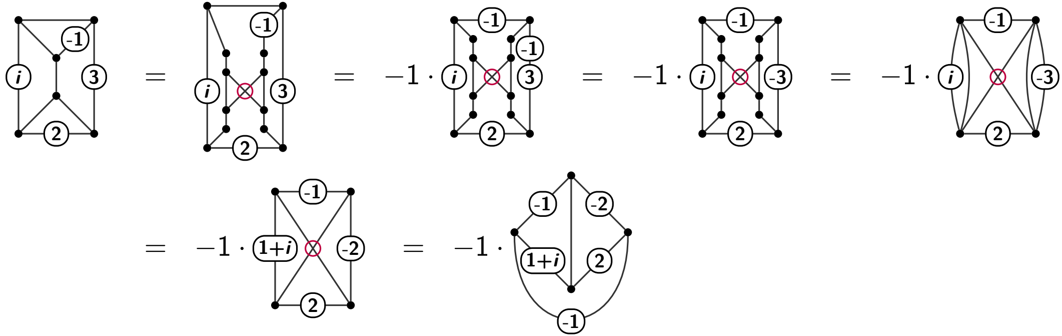
$= -1 \cdot$



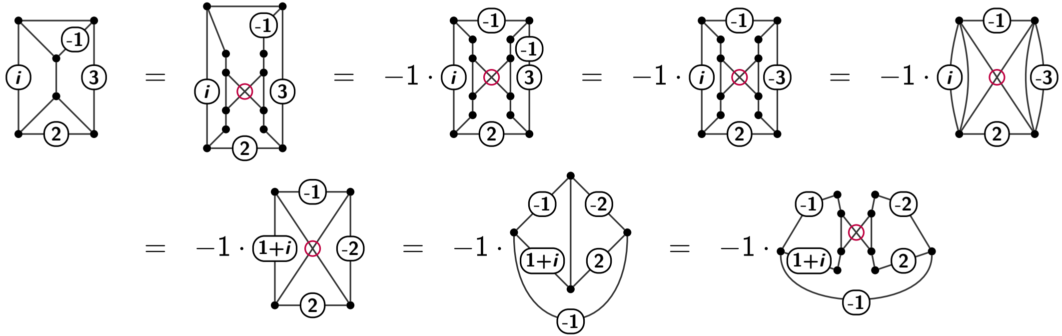
$= -1 \cdot$



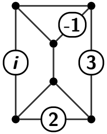
Example



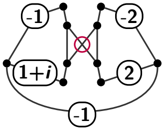
Example



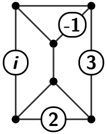
Example



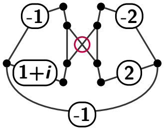
$= \; -1 \cdot$



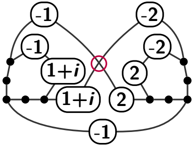
Example



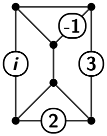
$= -1 \cdot$



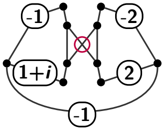
$= -1 \cdot$



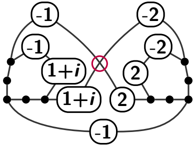
Example



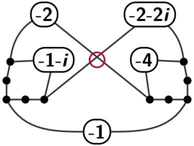
$= -1 \cdot$



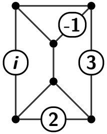
$= -1 \cdot$



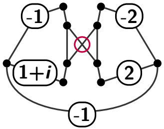
$= -1 \cdot$



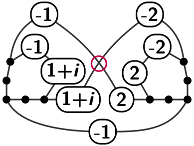
Example



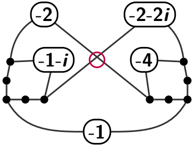
$= -1 \cdot$



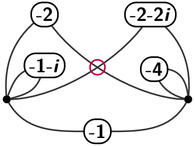
$= -1 \cdot$



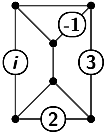
$= -1 \cdot$



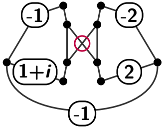
$= -1 \cdot$



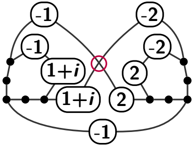
Example



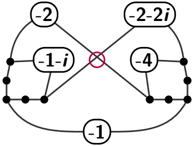
$= -1 \cdot$



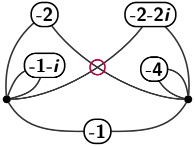
$= -1 \cdot$



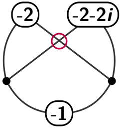
$= -1 \cdot$



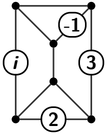
$= -1 \cdot$



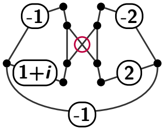
$= -1 \cdot$



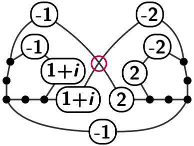
Example



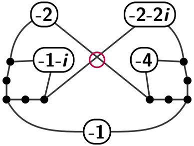
$= -1 \cdot$



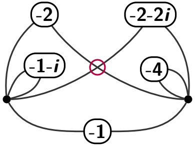
$= -1 \cdot$



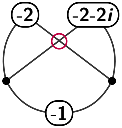
$= -1 \cdot$



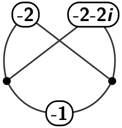
$= -1 \cdot$



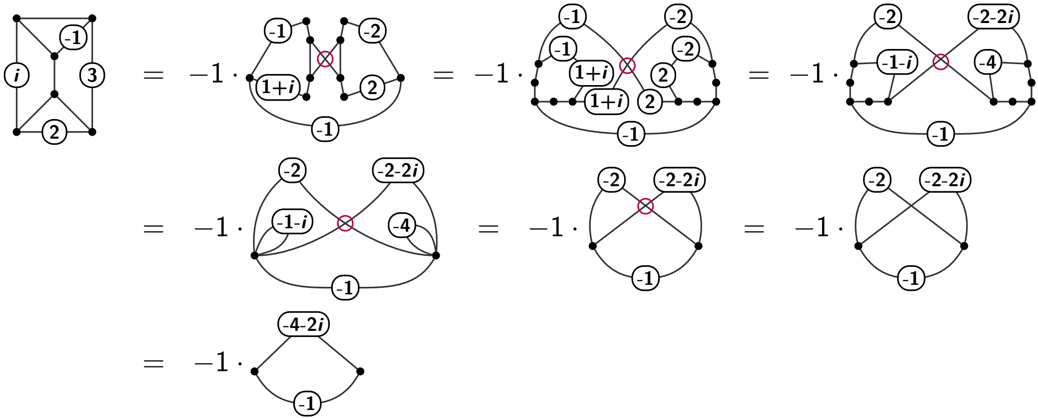
$= -1 \cdot$



$= -1 \cdot$

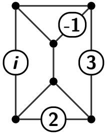


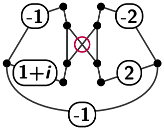
Example

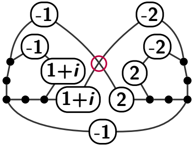


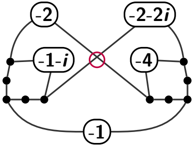
i
-1
3
2
= −1 ·
-1
-2
1+i
2
-1
= −1 ·
-1
-2
-1
-2
1+i
2
1+i
2
-1
= −1 ·
-2
-2-2i
-1-i
-4
-1
= −1 ·
-2
-2-2i
-1-i
-4
-1
= −1 ·
-2
-2-2i
-1
= −1 ·
-2
-2-2i
-1
= −1 ·
-4-2i
-1

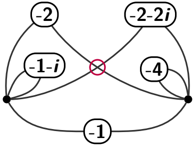
Example

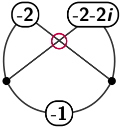


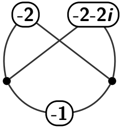
$= -1 \cdot$ 

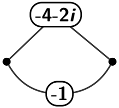
$= -1 \cdot$ 

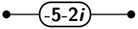
$= -1 \cdot$ 

$= -1 \cdot$ 

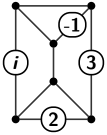
$= -1 \cdot$ 

$= -1 \cdot$ 

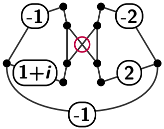
$= -1 \cdot$ 

$= -1 \cdot$ 

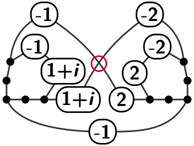
Example



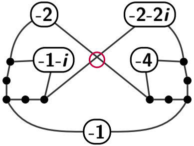
$= \; -1 \cdot$



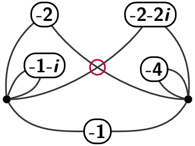
$= \; -1 \cdot$



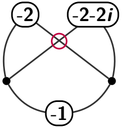
$= \; -1 \cdot$



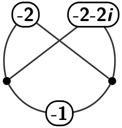
$= \; -1 \cdot$



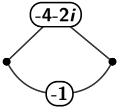
$= \; -1 \cdot$



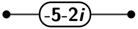
$= \; -1 \cdot$



$= \; -1 \cdot$



$= \; -1 \cdot$



$= \; 5 + 2i$

Recap

- Deciding existence of solution to SAT problem: **NP-complete**

Recap

- Deciding existence of solution to SAT problem: **NP-complete**
- Counting solutions to SAT problem: **#P-complete**

Recap

- Deciding existence of solution to SAT problem: **NP-complete**
- Counting solutions to SAT problem: **#P-complete**

- Deciding existence of perfect matching: **P**

Recap

- Deciding existence of solution to SAT problem: **NP-complete**
- Counting solutions to SAT problem: **#P-complete**

- Deciding existence of perfect matching: **P**
- Counting perfect matchings: **#P-complete**

Recap

- Deciding existence of solution to SAT problem: **NP-complete**
- Counting solutions to SAT problem: **#P-complete**

- Deciding existence of perfect matching: **P**
- Counting perfect matchings: **#P-complete**

- Counting perfect matchings in planar graph: **FP**

Recap

- Deciding existence of solution to SAT problem: **NP-complete**
- Counting solutions to SAT problem: **#P-complete**

- Deciding existence of perfect matching: **P**
- Counting perfect matchings: **#P-complete**

- Counting perfect matchings in planar graph: **FP**
- Deciding planarity: **P**