Combinatorial Optimization in Bioinfo

Lecture 3 – Graph algorithms and assembly

Yann Ponty

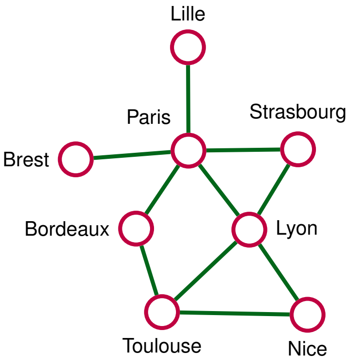
AMIBio Team
CNRS & École Polytechnique

Why graphs?

$$G = (V, E)$$

where $V \rightarrow$ vertices/nodes, and $E \rightarrow$ edges.

What for? Graphs model entities, and the way they are connected

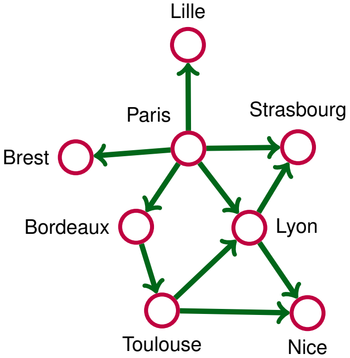


Why graphs?

$$G = (V, E)$$

where $V \rightarrow$ vertices/nodes, and $E \rightarrow$ edges.

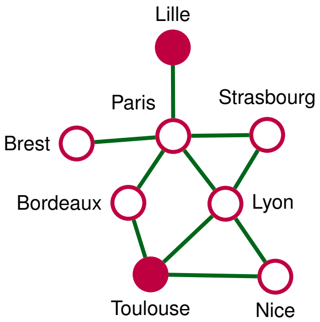
What for? Graphs model entities, and the way they are connected



Some graphs problems

Graphs represent abstractions on which many problems can be formulated[1] providing ready-to-use recipes for algorithm design:

- Shortest path between two nodes (GPS)
- Maximum clique (Community detection; Biclustering)
- Max independent set (Redundancy filtering)
- Max matching (RNA folding with PK)
- Traveling salesperson/SuperString (Assembly)

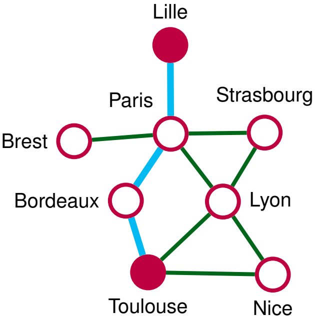


[1] and even sometimes, if you're lucky, solved efficiently

Some graphs problems

Graphs represent abstractions on which many problems can be formulated[1] providing ready-to-use recipes for algorithm design:

- Shortest path between two nodes (GPS)
- Maximum clique (Community detection; Biclustering)
- Max independent set (Redundancy filtering)
- Max matching (RNA folding with PK)
- Traveling salesperson/SuperString (Assembly)

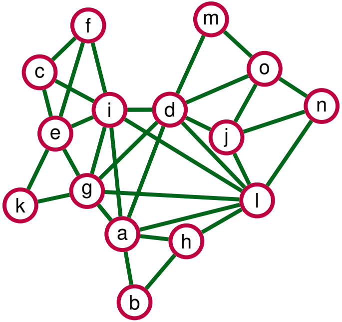


[1] and even sometimes, if you're lucky, solved efficiently

Some graphs problems

Graphs represent abstractions on which many problems can be formulated[1] providing ready-to-use recipes for algorithm design:

- Shortest path between two nodes (GPS)
- Maximum clique (Community detection; Biclustering)
- Max independent set (Redundancy filtering)
- Max matching (RNA folding with PK)
- Traveling salesperson/SuperString (Assembly)

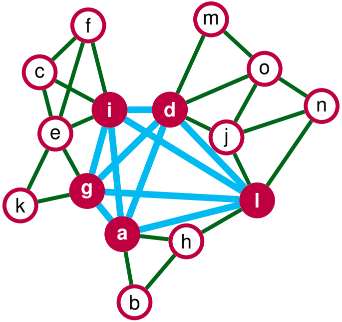


[1] and even sometimes, if you're lucky, solved efficiently

Some graphs problems

Graphs represent abstractions on which many problems can be formulated[1] providing ready-to-use recipes for algorithm design:

- Shortest path between two nodes (GPS)
- Maximum clique (Community detection; Biclustering)
- Max independent set (Redundancy filtering)
- Max matching (RNA folding with PK)
- Traveling salesperson/SuperString (Assembly)



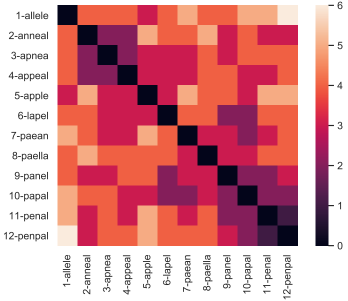
[1] and even sometimes, if you're lucky, solved efficiently

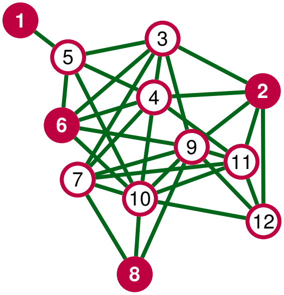
Some graphs problems

Graphs represent abstractions on which many problems can be formulated[1] providing ready-to-use recipes for algorithm design:

- Shortest path between two nodes (GPS)
- Maximum clique (Community detection; Biclustering)
- Max independent set (Redundancy filtering)
- Max matching (RNA folding with PK)
- Traveling salesperson/SuperString (Assembly)

```
1-allele
2-anneal
3-apnea
4-appeal
5-apple
6-lapel
7-paean
8-paella
9-panel
10-papal
11-penal
12-penpal
```





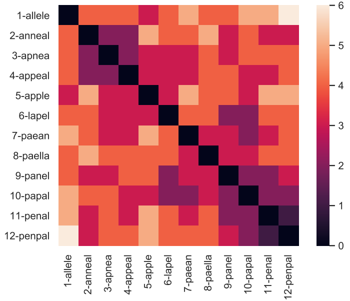
[1] and even sometimes, if you're lucky, solved efficiently

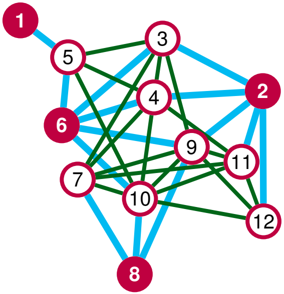
Some graphs problems

Graphs represent abstractions on which many problems can be formulated[1] providing ready-to-use recipes for algorithm design:

- Shortest path between two nodes (GPS)
- Maximum clique (Community detection; Biclustering)
- Max independent set (Redundancy filtering)
- Max matching (RNA folding with PK)
- Traveling salesperson/SuperString (Assembly)

```
1-allele
2-anneal
3-apnea
4-appeal
5-apple
6-lapel
7-paean
8-paella
9-panel
10-papal
11-penal
12-penpal
```



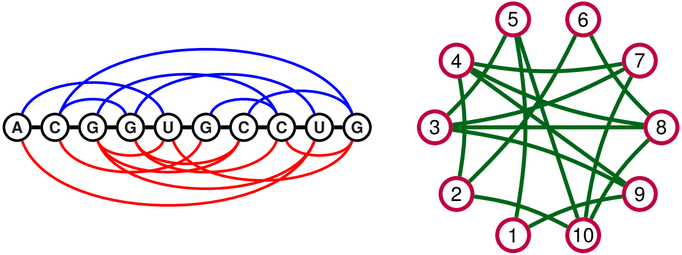


[1] and even sometimes, if you're lucky, solved efficiently

Some graphs problems

Graphs represent abstractions on which many problems can be formulated[1] providing ready-to-use recipes for algorithm design:

- Shortest path between two nodes (GPS)
- Maximum clique (Community detection; Biclustering)
- Max independent set (Redundancy filtering)
- Max matching (RNA folding with PK)
- Traveling salesperson/SuperString (Assembly)

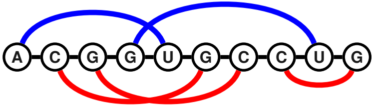


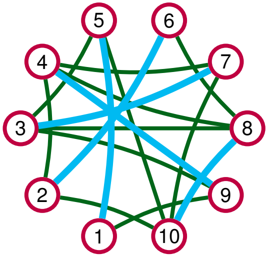
[1] and even sometimes, if you're lucky, solved efficiently

Some graphs problems

Graphs represent abstractions on which many problems can be formulated[1] providing ready-to-use recipes for algorithm design:

- Shortest path between two nodes (GPS)
- Maximum clique (Community detection; Biclustering)
- Max independent set (Redundancy filtering)
- Max matching (RNA folding with PK)
- Traveling salesperson/SuperString (Assembly)





[1] and even sometimes, if you're lucky, solved efficiently

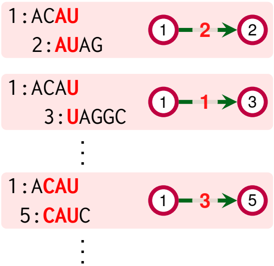
Some graphs problems

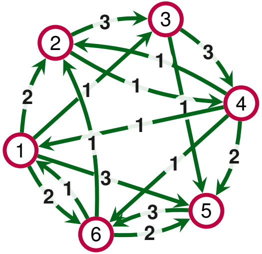
Graphs represent abstractions on which many problems can be formulated[1] providing ready-to-use recipes for algorithm design:

- Shortest path between two nodes (GPS)
- Maximum clique (Community detection; Biclustering)
- Max independent set (Redundancy filtering)
- Max matching (RNA folding with PK)
- Traveling salesperson/SuperString (Assembly)

NGS Reads

1: ACAU
2: AUAG
3: UAGGC
4: GGCA
5: CAUC
6: AUCA





[1] and even sometimes, if you're lucky, solved efficiently

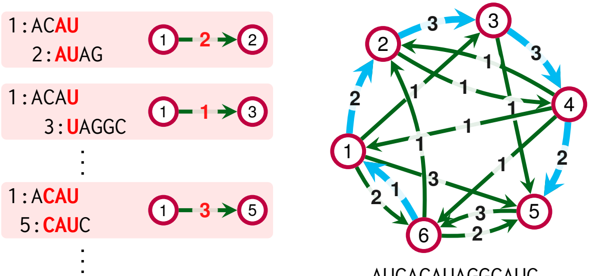
Some graphs problems

Graphs represent abstractions on which many problems can be formulated[1] providing ready-to-use recipes for algorithm design:

- Shortest path between two nodes (GPS)
- Maximum clique (Community detection; Biclustering)
- Max independent set (Redundancy filtering)
- Max matching (RNA folding with PK)
- Traveling salesperson/SuperString (Assembly)

NGS Reads

1: ACAU
2: AUAG
3: UAGGC
4: GGCA
5: CAUC
6: AUCA



[1] and even sometimes, if you're lucky, solved efficiently

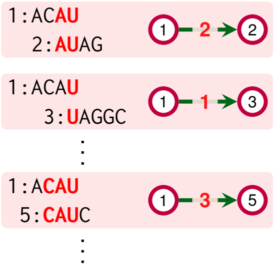
Some graphs problems

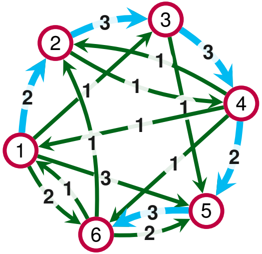
Graphs represent abstractions on which many problems can be formulated[1] providing ready-to-use recipes for algorithm design:

- Shortest path between two nodes (GPS)
- Maximum clique (Community detection; Biclustering)
- Max independent set (Redundancy filtering)
- Max matching (RNA folding with PK)
- Traveling salesperson/SuperString (Assembly)

NGS Reads

1: ACAU
2: AUAG
3: UAGGC
4: GGCA
5: CAUC
6: AUCA





ACAUAGGCAUCA

[1] and even sometimes, if you're lucky, solved efficiently

Shortest path

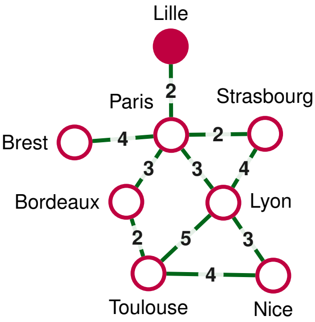
SHORTEST PATH problem

Input: (Di)graph $G = (V, E)$; Origin $s \in E$ and Destination $t \in E$; Distance function $\delta : E \to \mathbb{R}^+$

Output: Minimal distance path from s to t, *i.e.* $p^\star$ such that

$$p = \underset{\substack{p=(u_1,u_2\ldots u_k)\\ u_1=s, u_k=t}}{\operatorname{argmin}} \sum_{i=1}^{k-1} \delta(u_i, u_{i+1})$$

- Easy problem $\Leftrightarrow$ Poly-time algo.
- Idea (Dijkstra):
  - Flood from s
  - At each step, extend closest non-visited vertex (priority queue)
  - Stop when t is reached
- Complexity: $\Theta(|V| + |E|)$
- Powerful but beware of $|V|$



Shortest path

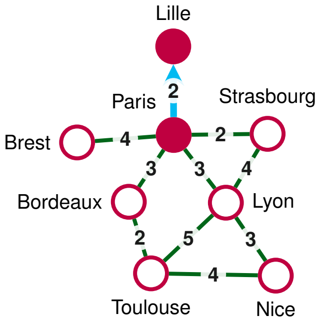
Shortest Path problem

Input: (Di)graph $G = (V, E)$; Origin $s \in E$ and Destination $t \in E$; Distance function $\delta : E \to \mathbb{R}^+$

Output: Minimal distance path from s to t, *i.e.* $p^\star$ such that

$$p = \underset{\substack{p=(u_1,u_2\ldots u_k)\\ u_1=s, u_k=t}}{\text{argmin}} \sum_{i=1}^{k-1} \delta(u_i, u_{i+1})$$

- Easy problem $\Leftrightarrow$ Poly-time algo.
- Idea (Dijkstra):
  - Flood from s
  - At each step, extend closest non-visited vertex (priority queue)
  - Stop when t is reached
- Complexity: $\Theta(|V| + |E|)$
- Powerful but beware of $|V|$



Shortest path

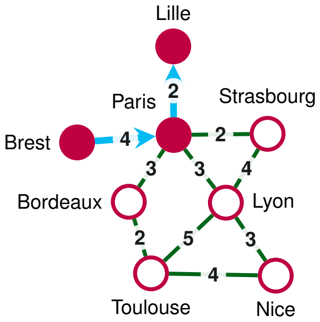
Shortest Path problem

Input: (Di)graph $G = (V, E)$; Origin $s \in E$ and Destination $t \in E$; Distance function $\delta : E \to \mathbb{R}^+$

Output: Minimal distance path from s to t, *i.e.* $p^\star$ such that

$$p = \underset{\substack{p=(u_1,u_2\ldots u_k)\\ u_1=s, u_k=t}}{\operatorname{argmin}} \sum_{i=1}^{k-1} \delta(u_i, u_{i+1})$$

- Easy problem $\Leftrightarrow$ Poly-time algo.
- Idea (Dijkstra):
  - Flood from s
  - At each step, extend closest non-visited vertex (priority queue)
  - Stop when t is reached
- Complexity: $\Theta(|V| + |E|)$
- Powerful but beware of $|V|$



Shortest path

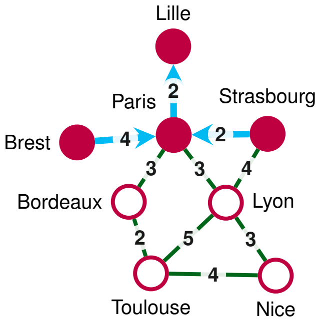
SHORTEST PATH problem

Input: (Di)graph $G = (V, E)$; Origin $s \in E$ and Destination $t \in E$; Distance function $\delta : E \rightarrow \mathbb{R}^+$

Output: Minimal distance path from s to t, *i.e.* $p^\star$ such that

$$p = \underset{\substack{p=(u_1,u_2\ldots u_k)\\ u_1=s,u_k=t}}{\operatorname{argmin}} \sum_{i=1}^{k-1} \delta(u_i, u_{i+1})$$

- Easy problem $\Leftrightarrow$ Poly-time algo.
- Idea (Dijkstra):
  - Flood from s
  - At each step, extend closest non-visited vertex (priority queue)
  - Stop when t is reached
- Complexity: $\Theta(|V| + |E|)$
- Powerful but beware of $|V|$



Shortest path

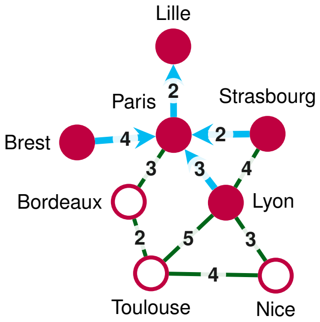
SHORTEST PATH problem

Input: (Di)graph $G = (V, E)$; Origin $s \in E$ and Destination $t \in E$; Distance function $\delta : E \to \mathbb{R}^+$

Output: Minimal distance path from s to t, *i.e.* $p^\star$ such that

$$p = \underset{\substack{p=(u_1,u_2\ldots u_k)\\ u_1=s, u_k=t}}{\operatorname{argmin}} \sum_{i=1}^{k-1} \delta(u_i, u_{i+1})$$

- Easy problem $\Leftrightarrow$ Poly-time algo.
- Idea (Dijkstra):
  - Flood from s
  - At each step, extend closest non-visited vertex (priority queue)
  - Stop when t is reached
- Complexity: $\Theta(|V| + |E|)$
- Powerful but beware of $|V|$



Shortest path

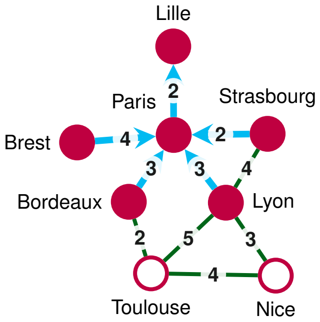
Shortest Path problem

Input: (Di)graph $G = (V, E)$; Origin $s \in E$ and Destination $t \in E$; Distance function $\delta : E \to \mathbb{R}^+$

Output: Minimal distance path from s to t, *i.e.* $p^\star$ such that

$$p = \underset{\substack{p=(u_1, u_2 \ldots u_k)\\ u_1=s, u_k=t}}{\operatorname{argmin}} \sum_{i=1}^{k-1} \delta(u_i, u_{i+1})$$

- Easy problem $\Leftrightarrow$ Poly-time algo.
- Idea (Dijkstra):
  - Flood from s
  - At each step, extend closest non-visited vertex (priority queue)
  - Stop when t is reached
- Complexity: $\Theta(|V| + |E|)$
- Powerful but beware of $|V|$



Shortest path

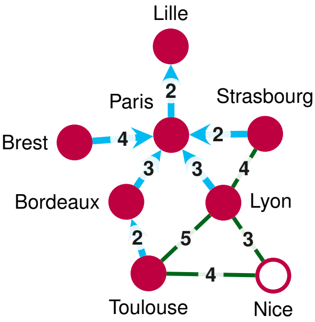
Shortest Path problem

Input: (Di)graph $G = (V, E)$; Origin $s \in E$ and Destination $t \in E$; Distance function $\delta : E \to \mathbb{R}^+$

Output: Minimal distance path from s to t, *i.e.* $p^\star$ such that

$$p = \underset{\substack{p=(u_1,u_2\ldots u_k)\\ u_1=s,u_k=t}}{\operatorname{argmin}} \sum_{i=1}^{k-1} \delta(u_i, u_{i+1})$$

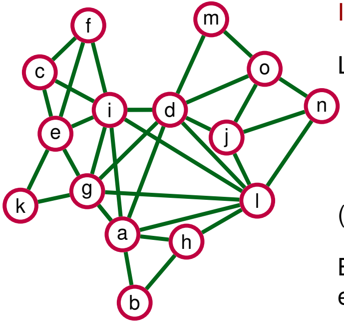
- Easy problem $\Leftrightarrow$ Poly-time algo.
- Idea (Dijkstra):
  - Flood from s
  - At each step, extend closest non-visited vertex (priority queue)
  - Stop when t is reached
- Complexity: $\Theta(|V| + |E|)$
- Powerful but beware of $|V|$



Community detection and Max clique

MAX CLIQUE problem
Input: Graph $G = (V, E)$
Output: Largest set of pairwise connected vertices.

Useful when trying to detect a group of highly interconnected elements (*e.g.* molecules)



In a nutshell: A hard problem!

Let $n = |V| + |E|$ & k max size of clique:

- NP-hard: No $\mathcal{O}(P(n))$ algo
- Not FPT (W[1]): No $\mathcal{O}(\alpha^k.P(n))$ algo
- XP: Trivial algo in $\mathcal{O}(n^k)$

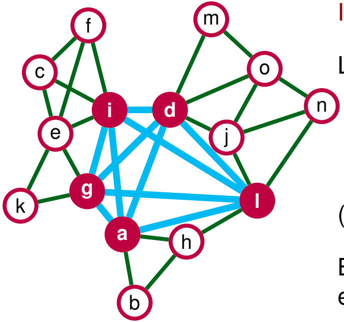
(unless P=NP)

But many heuristic solutions, so still worth considering if natural (last resort)

Community detection and Max clique

MAX CLIQUE problem
Input: Graph $G = (V, E)$
Output: Largest set of pairwise connected vertices.

Useful when trying to detect a group of highly interconnected elements (*e.g.* molecules)



In a nutshell: A hard problem!

Let $n = |V| + |E|$ & k max size of clique:

- NP-hard: No $\mathcal{O}(P(n))$ algo
- Not FPT (W[1]): No $\mathcal{O}(\alpha^k.P(n))$ algo
- XP: Trivial algo in $\mathcal{O}(n^k)$

(unless P=NP)

But many heuristic solutions, so still worth considering if natural (last resort)

Max Independent Sets

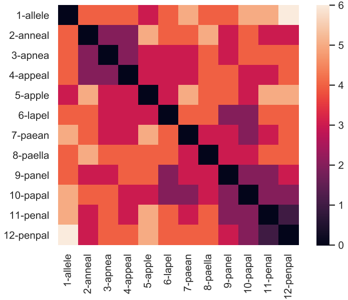
MAX INDEPENDENT SETS problem

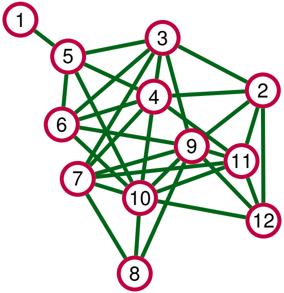
Input: Graph $G = (V, E)$

Output: Largest set of pairwise disconnected vertices.

Very natural while producing conflict-free collections of objects.

```
1-allele
2-anneal
3-apnea
4-appeal
5-apple
6-lapel
7-paean
8-paella
9-panel
10-papal
11-penal
12-penpal
```





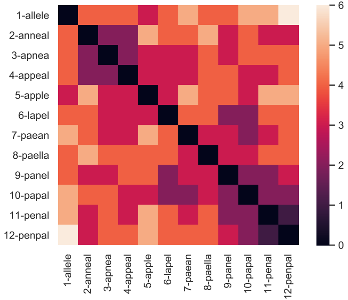
Unfortunately, NP-hard (again), but can be solved in $\mathcal{O}(2^t.|V| + |E|)$ time, *i.e.* efficiently for input graphs of low tree-width t.

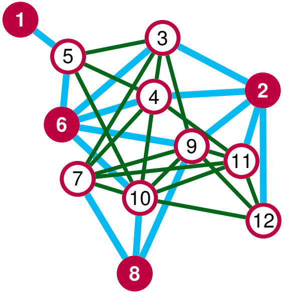
Max Independent Sets

MAX INDEPENDENT SETS problem
Input: Graph $G = (V, E)$
Output: Largest set of pairwise disconnected vertices.

Very natural while producing conflict-free collections of objects.

```
1-allele
2-anneal
3-apnea
4-appeal
5-apple
6-lapel
7-paean
8-paella
9-panel
10-papal
11-penal
12-penpal
```





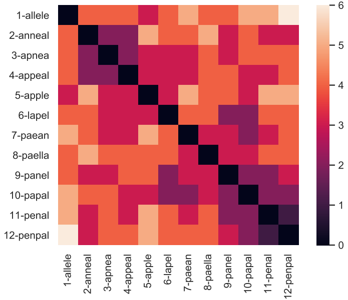
Unfortunately, NP-hard (again), but can be solved in $\mathcal{O}(2^t.|V| + |E|)$ time, *i.e.* efficiently for input graphs of low tree-width t.

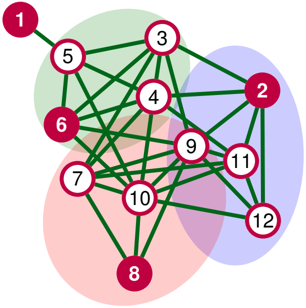
Max Independent Sets

MAX INDEPENDENT SETS problem
Input: Graph $G = (V, E)$
Output: Largest set of pairwise disconnected vertices.

Very natural while producing conflict-free collections of objects.

```
1-allele
2-anneal
3-apnea
4-appeal
5-apple
6-lapel
7-paean
8-paella
9-panel
10-papal
11-penal
12-penpal
```





Unfortunately, NP-hard (again), but can be solved in $\mathcal{O}(2^t.|V| + |E|)$ time, *i.e.* efficiently for input graphs of low tree-width t.

Max (weighted) matching

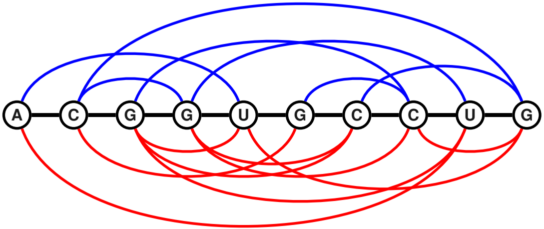
MAX MATCHING problem

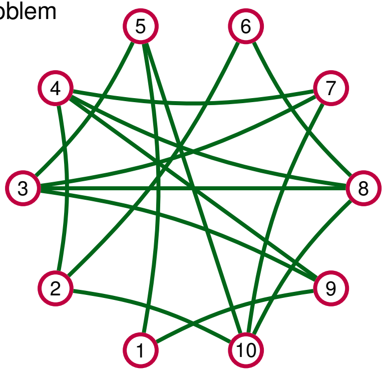
Input: Graph $G = (V, E)$

Output: Largest set of edges such that each vertex is represented in ≤ 1 edge.

Typical use-case: Simplify a n-body problem into a 2-body problem

$\rightarrow$ Useful for approximate solutions (guaranteed approx ratio)





At last, an *easy* problem! ($\exists$ poly-time algo 😎)

Idea (Edmunds): Greedy optim. by swapping augmenting paths

Yields global max. in $\mathcal{O}(|E|\sqrt{|V|})$ from Micali and Vazirani

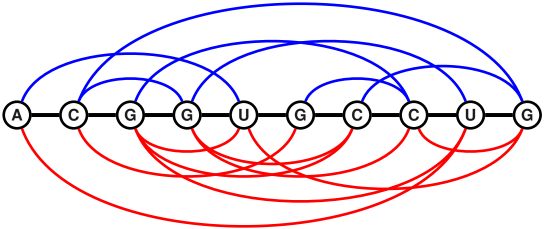
Max (weighted) matching

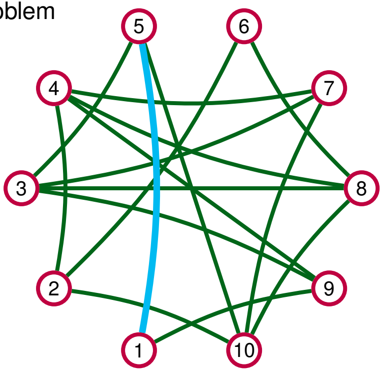
MAX MATCHING problem

Input: Graph $G = (V, E)$

Output: Largest set of edges such that each vertex is represented in ≤ 1 edge.

Typical use-case: Simplify a *n*-body problem into a 2-body problem
$\rightarrow$ Useful for approximate solutions (guaranteed approx ratio)



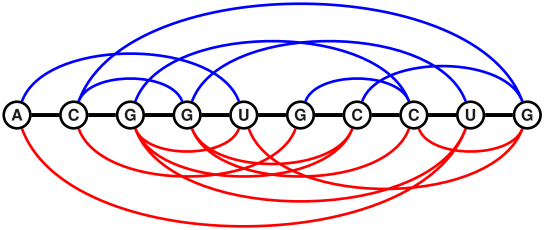


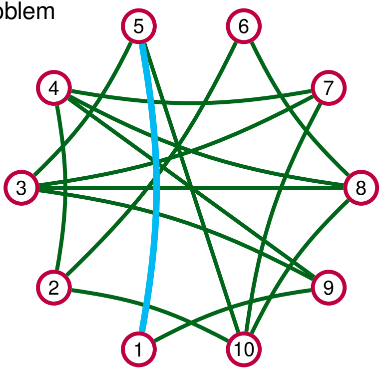
At last, an *easy* problem! ($\exists$ poly-time algo 😎)
Idea (Edmunds): Greedy optim. by swapping augmenting paths
Yields global max. in $\mathcal{O}(|E|\sqrt{|V|})$ from Micali and Vazirani

Max (weighted) matching

MAX MATCHING problem
Input: Graph $G = (V, E)$
Output: Largest set of edges such that each vertex is represented in ≤ 1 edge.

Typical use-case: Simplify a *n*-body problem into a 2-body problem
$\rightarrow$ Useful for approximate solutions (guaranteed approx ratio)





At last, an *easy* problem! ($\exists$ poly-time algo 😎)
Idea (Edmunds): Greedy optim. by swapping augmenting paths
Yields global max. in $\mathcal{O}(|E|\sqrt{|V|})$ from Micali and Vazirani

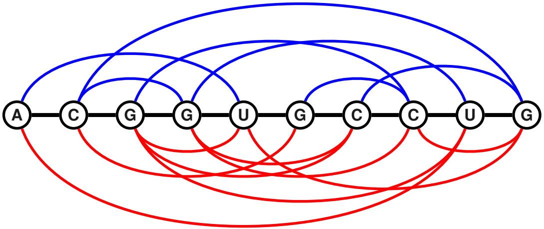
Max (weighted) matching

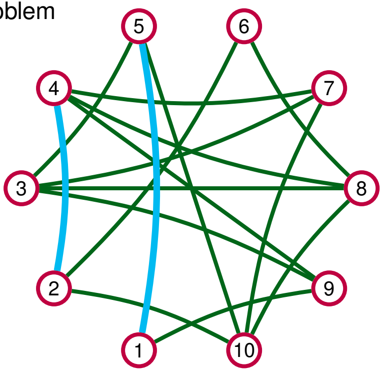
MAX MATCHING problem

Input: Graph $G = (V, E)$

Output: Largest set of edges such that each vertex is represented in ≤ 1 edge.

Typical use-case: Simplify a n-body problem into a 2-body problem
$\rightarrow$ Useful for approximate solutions (guaranteed approx ratio)





At last, an *easy* problem! ($\exists$ poly-time algo 😎)
Idea (Edmunds): Greedy optim. by swapping augmenting paths
Yields global max. in $\mathcal{O}(|E|\sqrt{|V|})$ from Micali and Vazirani

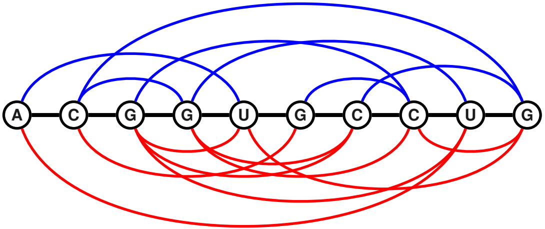
Max (weighted) matching

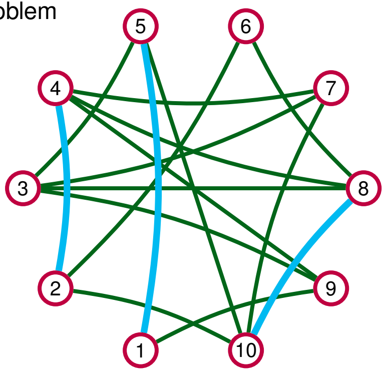
Max matching problem

Input: Graph $G = (V, E)$

Output: Largest set of edges such that each vertex is represented in ≤ 1 edge.

Typical use-case: Simplify a n-body problem into a 2-body problem
$\rightarrow$ Useful for approximate solutions (guaranteed approx ratio)





At last, an *easy* problem! ($\exists$ poly-time algo 😎)
Idea (Edmunds): Greedy optim. by swapping augmenting paths
Yields global max. in $\mathcal{O}(|E|\sqrt{|V|})$ from Micali and Vazirani

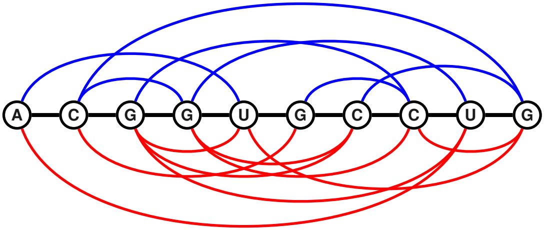
Max (weighted) matching

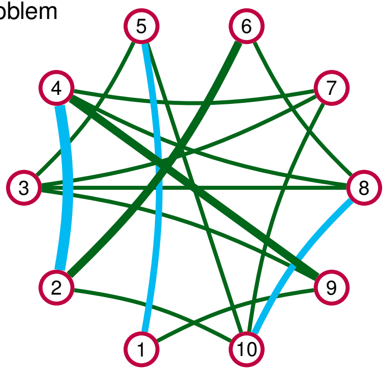
MAX MATCHING problem

Input: Graph $G = (V, E)$

Output: Largest set of edges such that each vertex is represented in ≤ 1 edge.

Typical use-case: Simplify a n-body problem into a 2-body problem
$\rightarrow$ Useful for approximate solutions (guaranteed approx ratio)



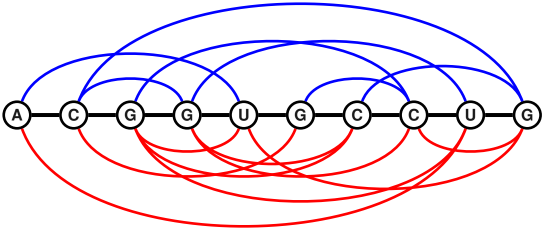


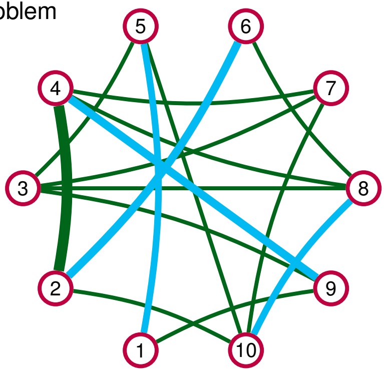
At last, an *easy* problem! ($\exists$ poly-time algo 😎)
Idea (Edmunds): Greedy optim. by swapping augmenting paths
Yields global max. in $\mathcal{O}(|E|\sqrt{|V|})$ from Micali and Vazirani

Max (weighted) matching

MAX MATCHING problem
Input: Graph $G = (V, E)$
Output: Largest set of edges such that each vertex is represented in $\leq$ 1 edge.

Typical use-case: Simplify a n-body problem into a 2-body problem
$\rightarrow$ Useful for approximate solutions (guaranteed approx ratio)





At last, an *easy* problem! (∃ poly-time algo 😎)
Idea (Edmunds): Greedy optim. by swapping augmenting paths
Yields global max. in $\mathcal{O}(|E|\sqrt{|V|})$ from Micali and Vazirani

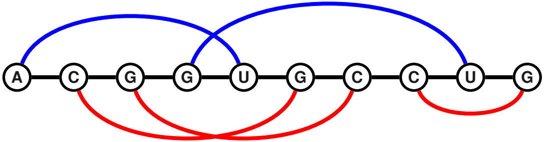
Max (weighted) matching

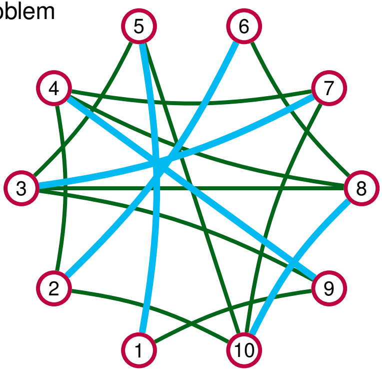
MAX MATCHING problem

Input: Graph $G = (V, E)$

Output: Largest set of edges such that each vertex is represented in ≤ 1 edge.

Typical use-case: Simplify a n-body problem into a 2-body problem
$\rightarrow$ Useful for approximate solutions (guaranteed approx ratio)





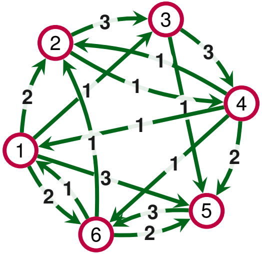
At last, an *easy* problem! ($\exists$ poly-time algo 😎)
Idea (Edmunds): Greedy optim. by swapping augmenting paths
Yields global max. in $\mathcal{O}(|E|\sqrt{|V|})$ from Micali and Vazirani

Hamiltonian paths

HAMILTONIAN PATH problem
Input: (Di)graph $G = (V, E)$
Output: Path of G passing through vertex of V exactly once

NP-hard along with its optimization version

TRAVELING SALESPERSON (TSP) problem
Input: (Di)graph $G = (V, E)$; Reward function $\rho : E \to \mathbb{R}$
Output: Hamiltonian path p maximizing reward $\sum_{e \in p} \rho(e)$

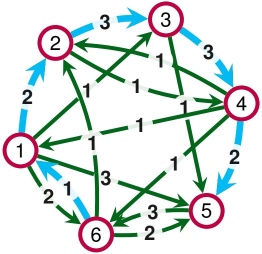


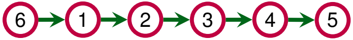
Hamiltonian paths

HAMILTONIAN PATH problem
Input: (Di)graph $G = (V, E)$
Output: Path of G passing through vertex of V exactly once

NP-hard along with its optimization version

TRAVELING SALESPERSON (TSP) problem
Input: (Di)graph $G = (V, E)$; Reward function $\rho : E \to \mathbb{R}$
Output: Hamiltonian path p maximizing reward $\sum_{e \in p} \rho(e)$





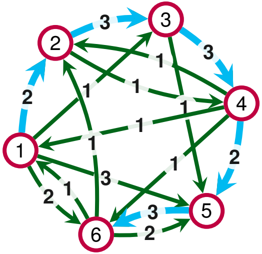
Reward: 11

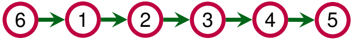
Hamiltonian paths

HAMILTONIAN PATH problem
Input: (Di)graph $G = (V, E)$
Output: Path of G passing through vertex of V exactly once

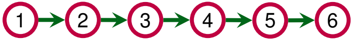
NP-hard along with its optimization version

TRAVELING SALESPERSON (TSP) problem
Input: (Di)graph $G = (V, E)$; Reward function $\rho : E \to \mathbb{R}$
Output: Hamiltonian path p maximizing reward $\sum_{e \in p} \rho(e)$





Reward: 11



Reward: 13

TSP and Assembly

Assembly: Given set of NGS reads, find smallest (parsimonious) genome/transcript that explains presence of each read in dataset

> SUPERSTRING **problem**
>
> **Input:** Strings $w_1, w_2, \ldots w_k$
>
> **Output:** String w of min. length, where each w_i occurs as motif

Again a hard problem, but highly similar to TSP.

NGS Reads

1 : ACAU
2 : AUAG
3 : UAGGC
4 : GGCA
5 : CAUC
6 : AUCA

Concatenation always possible but costly

ACAU AUAG UAGGC GGCA CAUC AUCA (len=25)

Compacting overlaps may be beneficial

ACAUAGGCAUCA (len=12)

but savings depends on order.

Idea: Find order that maximizes overlaps

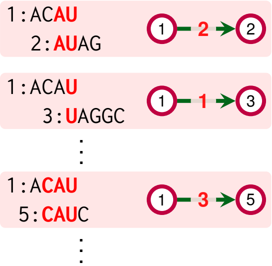
$$\text{len}(w_1, w_2 \ldots) = \sum_i \text{len}(w_i) - \sum_i \text{ov}(w_i, w_{i+1}))$$

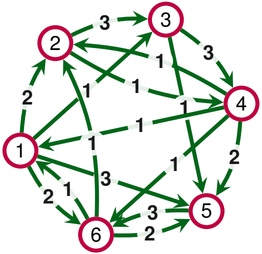
Rem: Assumes no read contained into another

Detailed constructs

NGS Reads

1: ACAU
2: AUAG
3: UAGGC
4: GGCA
5: CAUC
6: AUCA

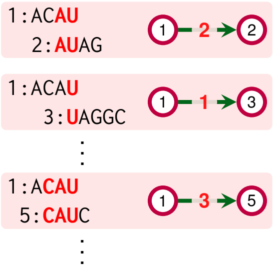


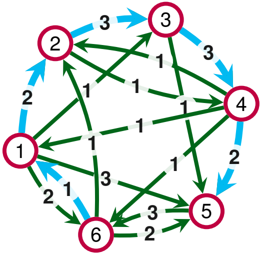


Detailed constructs

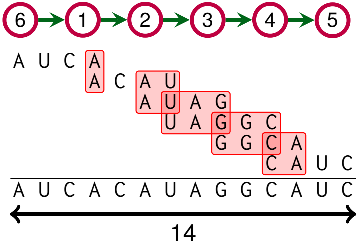
NGS Reads

1: ACAU
2: AUAG
3: UAGGC
4: GGCA
5: CAUC
6: AUCA





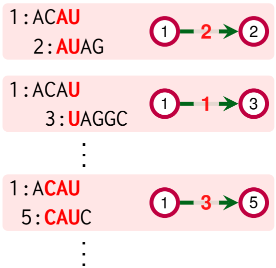
AUCACAUAGGCAUC

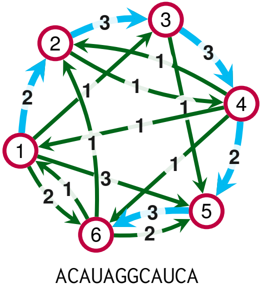


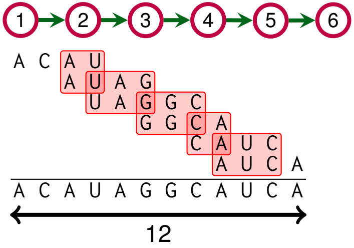
Detailed constructs

NGS Reads

1: ACAU
2: AUAG
3: UAGGC
4: GGCA
5: CAUC
6: AUCA



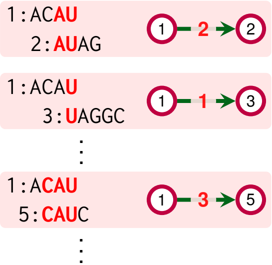


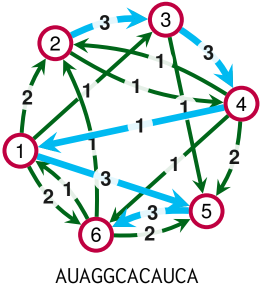


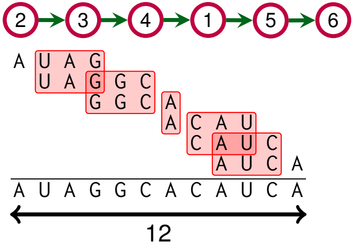
Detailed constructs

NGS Reads

1: ACAU
2: AUAG
3: UAGGC
4: GGCA
5: CAUC
6: AUCA





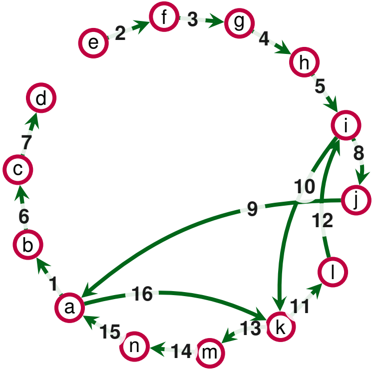


Eulerian paths

EULERIAN PATH problem

Input: Graph $G = (V, E)$

Output: Path p traversing every edge of E exactly once



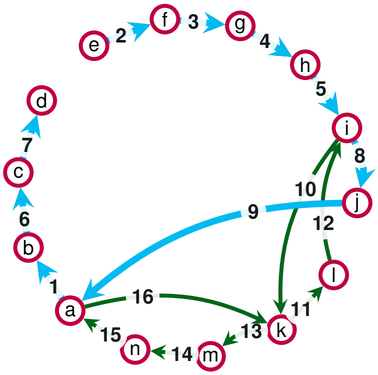
Possible only if vertices balanced
(except possibly start & end)

Algo: Build start→end greedy path, extending until deadend
While edge(s) remain, iterate: build cycle and insert it

Build edge sequence from cycles
(any order→ Possibly many!)

Eulerian paths

EULERIAN PATH problem
Input: Graph $G = (V, E)$
Output: Path p traversing every edge of E exactly once



Possible only if vertices balanced
(except possibly start & end)

Algo: Build start→end greedy path, extending until deadend
While edge(s) remain, iterate: build cycle and insert it

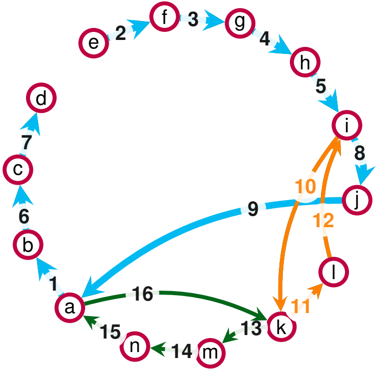
Build edge sequence from cycles
(any order→ Possibly many!)

Eulerian paths

EULERIAN PATH problem

Input: Graph $G = (V, E)$

Output: Path p traversing every edge of E exactly once



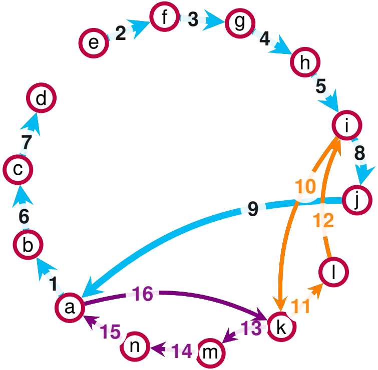
Possible only if vertices balanced
(except possibly start & end)

Algo: Build start→end greedy path, extending until deadend
While edge(s) remain, iterate: build cycle and insert it

Build edge sequence from cycles
(any order→ Possibly many!)

Eulerian paths

> **EULERIAN PATH problem**
> **Input:** Graph $G = (V, E)$
> **Output:** Path p traversing every edge of E exactly once



Possible only if vertices balanced
(except possibly start & end)

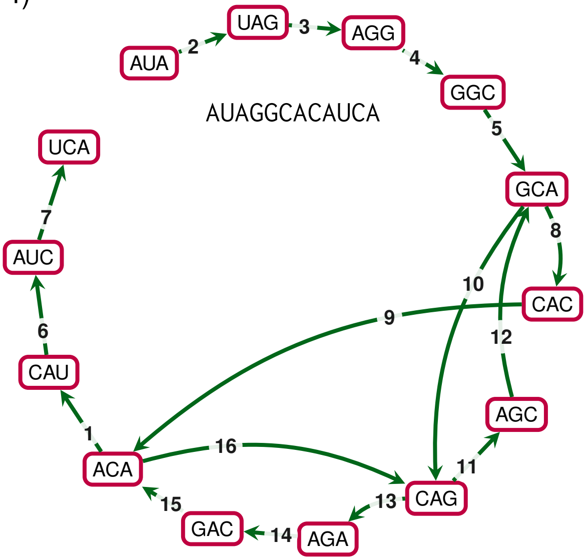
Algo: Build start→end greedy path, extending until deadend
While edge(s) remain, iterate: build cycle and insert it

Build edge sequence from cycles
(any order→ Possibly many!)

k-mers based assembly

NGS k-mers (k=4)

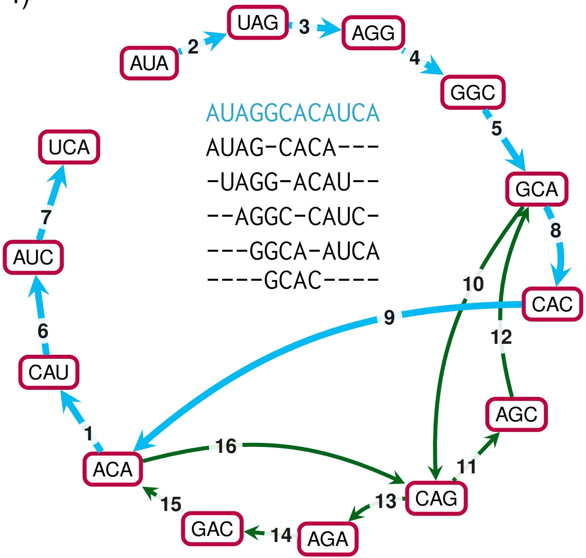
1: ACAU
2: AUAG
3: UAGG
4: AGGC
5: GGCA
6: CAUC
7: AUCA
8: GCAC
9: CACA
10: GCAG
11: CAGC
12: AGCA
13: CAGA
14: AGAC
15: GACA
16: ACAG



k-mers based assembly

NGS k-mers (k=4)

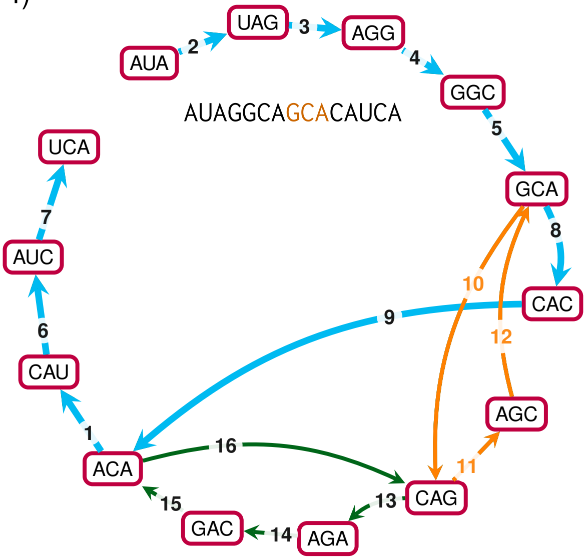
1: ACAU
2: AUAG
3: UAGG
4: AGGC
5: GGCA
6: CAUC
7: AUCA
8: GCAC
9: CACA
10: GCAG
11: CAGC
12: AGCA
13: CAGA
14: AGAC
15: GACA
16: ACAG



k-mers based assembly

NGS k-mers (k=4)

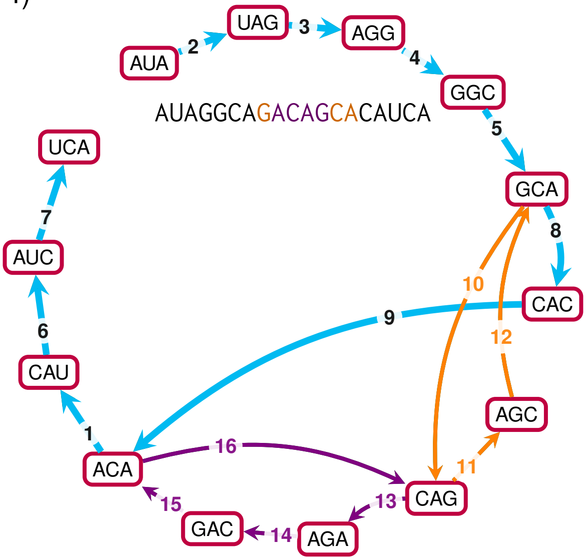
1: ACAU
2: AUAG
3: UAGG
4: AGGC
5: GGCA
6: CAUC
7: AUCA
8: GCAC
9: CACA
10: GCAG
11: CAGC
12: AGCA
13: CAGA
14: AGAC
15: GACA
16: ACAG



k-mers based assembly

NGS k-mers (k=4)

1: ACAU
2: AUAG
3: UAGG
4: AGGC
5: GGCA
6: CAUC
7: AUCA
8: GCAC
9: CACA
10: GCAG
11: CAGC
12: AGCA
13: CAGA
14: AGAC
15: GACA
16: ACAG



Conclusions

- Graphs are awesome, and ubiquitous in Bioinformatics
- Faced with a new problem, finding formulation as graph problem allows to tap into centuries of algorithmic design. . .
- . . . to find efficient (poly time) algorithms . . .
- . . . or identify workarounds for hardness results (FPT, approx, heuristics)
- It also enables efficient laziness through reuse of standard implementations for algorithms and data structure → Lab work
- Knowing graphs algorithms informs choice of model/objective during method development, to achieve good tradeoff between expressivity and tractability

But beware of using graphs just for the sake of using them[2].

[2] From a Hammer's perspective, everything looks like a nail. . . Don't be a Hammer!