SE284: Introduction to Graph Algorithms

Katerina Taškova

Email: katerina.taskova@auckland.ac.nz
Room: 303S.483

Outline

Hard (di)graph optimization problems

Lecture Notes 34, Textbook 6.6
(both very short, no examples)

Other (di)graph optimization/decision problems

There are many more graph and network computational problems.

- Many do not have easy or polynomial-time solutions. Best approach: Try all possible solutions.
- However a few of them are in a special category in that their solutions can be verified in polynomial-time (NP).
- In addition, many of these are proven to be harder than anything else in NP (NP-hard).
- Other algorithm design techniques like backtracking, branch-and-bound or approximation algorithms are needed.

Other (di)graph optimization/decision problems

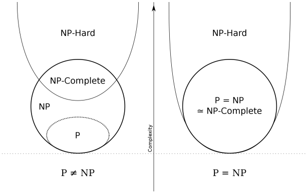
There are many more graph and network computational problems.

- Many do not have easy or polynomial-time solutions. Best approach: Try all possible solutions.
- However a few of them are in a special category in that their solutions can be verified in polynomial-time (NP).
- In addition, many of these are proven to be harder than anything else in NP (NP-hard).
- Other algorithm design techniques like backtracking, branch-and-bound or approximation algorithms are needed.

Other (di)graph optimization/decision problems

There are many more graph and network computational problems.

- Many do not have easy or polynomial-time solutions. Best approach: Try all possible solutions.
- However a few of them are in a special category in that their solutions can be verified in polynomial-time (NP).
- In addition, many of these are proven to be harder than anything else in NP (NP-hard).



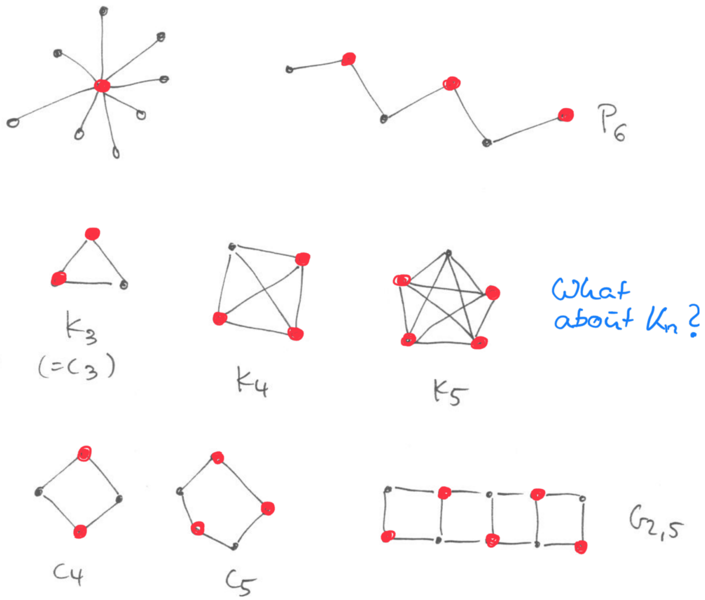
- Other algorithm design techniques like backtracking, branch-and-bound or approximation algorithms are needed.

Examples of NP-complete graph problems

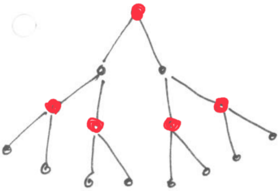
- Vertex Cover Problem [1]: Is there a subset of k vertices such that every edge is covered? (optimization problem: find a minimum-size subset)
- Independent Set Problem: Is there subset of k vertices such that not two are adjacent? (optimization problem: find a maximum-size subset)
- Dominating Set Problem: Is there a subset D of k vertices such that each vertex is in the neighborhood (distance ≤ 1) of D?
- Hamiltonian Cycle Problem: Is there a cycle that uses all vertices? (decision problem)
- Traveling Salesman Problem: Is there a cycle of length $|G|$ in an edge-weighted graph G with cost at most c?
- Is there a k-colouring of a graph, for fixed $k \geq 3$?

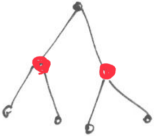
[1] In the textbook, there is a typo in the definition " a special subset of vertices so that each ~~vertex~~ edge of the graph is adjacent to one in that subset"

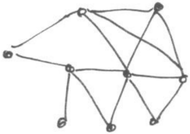
Vertex cover – examples



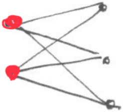
Vertex cover – examples



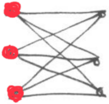




(4)

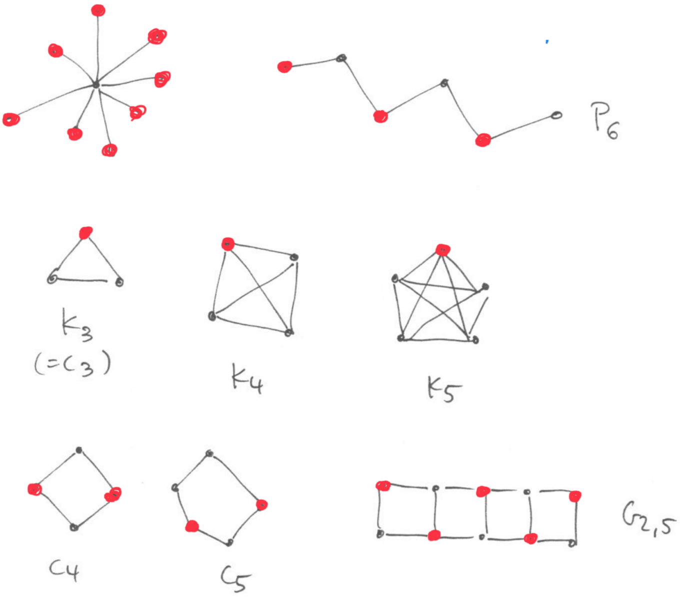


$K_{2,3}$

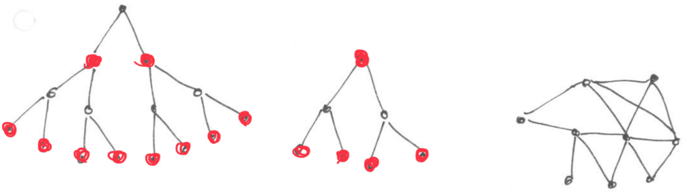


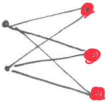
$K_{3,3}$

Independent set – examples

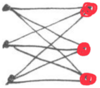


Independent set – examples



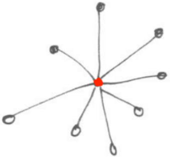


$K_{2,3}$



$K_{3,3}$

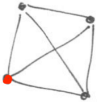
Dominating set – examples



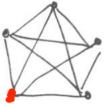
P_6

K_3

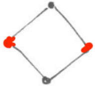
$(=C_3)$



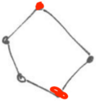
K_4



K_5



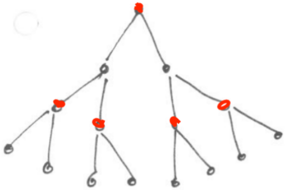
C_4

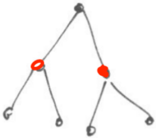


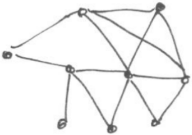
C_5

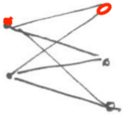
$G_{2,5}$

Dominating set – examples

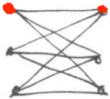








$K_{2,3}$



$K_{3,3}$

Hamiltonian cycle – examples

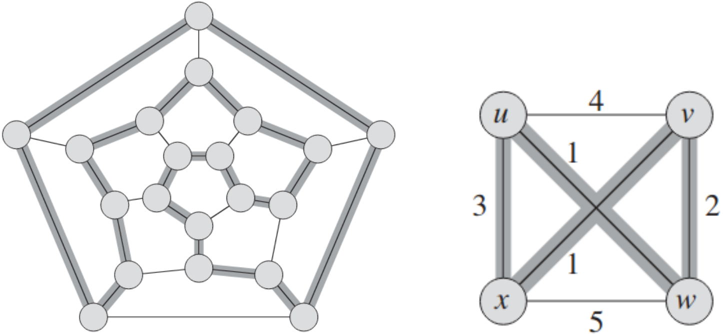


image source: Cormen at al 2011

Revision & exam preparation

Algorithm analysis and sorting algorithms in Week 1-6

Same instruction document as for the mid-term test; check Revision Notes in Module Week 5.

Efficient search (hashing) and graph algorithms in Week 7-12

Here; the same content is also in a separate document on Canvas in Module Exam.

Revision Week 7-9

Hashing hashing function/table, collisions, hashing function choice, collision resolution (chaining, open addressing), hashing analysis, universal hashing

Graph definitions all main graph definitions, reverse digraphs, including adjacency lists and adjacency matrices

General traversal algorithm idea of the algorithm, search forest, tree arcs, forward arcs, back arcs, cross arcs

DFS idea of DFS; pseudocode; search forest,tree, forward, back, and cross arcs in (di)graphs, properties of seen[] and done[] array values

BFS idea of BFS; pseudocode; search forests; tree, forward, back, and cross arcs in graphs and digraphs, properties of d[] array values

PFS pseudocode; how is it different from BSF/DFS

Topological order what is a DAG; sink/source vertices, algorithms to compute topological orders of a DAG

Revision Week 10-12

Cycles in digraphs Finding cycles in directed graphs using DFS

Girth of a graph definition; how can BFS be used to find the girth of a graph

Connectivity of graphs what does it mean for a graph to be connected; what are the connected components of a graph and how to compute them

Connectivity of digraphs when is a digraph strongly connected; what is a strongly connected component of a digraph; can DFS/BFS be used to find the strongly connected components of a digraph; what are the steps in the algorithm for finding strongly connected components of a digraph

Bipartite graphs what is a bipartite graph; properties of bipartite graphs; what algorithm can be used to decide if a graph is bipartite, colorings of graphs

Revision Week 10-12

Weighted (di)graphs definition, adjacency matrix/list of a weighted (di)graph, distance matrix, diameter/radius/eccentricity of (di)graph

SSSP Dijkstra's algorithm, Bellman-Ford algorithm, pseudocode for both; what are the underlying ideas of both algorithms; why do they work; what are the differences/commonalities between them

APSP Floyd's algorithm, pseudocode, idea of Floyd's algorithm, why/when does it work

MST definition of a minimum spanning tree; Prim's and Kruskal's algorithm; pseudocode of both algorithms, differences and commonalities between the two algorithms

Hard problems definitions of the Hamiltonian cycle, traveling salesman, independent/dominating set, and vertex cover problem

Exam preparation

- In short, everything we covered in the lectures is examinable.
- The exam is 2 hours long and approximately half of the marks will come from multiple choice questions (MCQ) and the other half from short answer questions (SAQ).
- The marks will split as follows, 1/3 of the marks for Week 1-6 material, 1/3 of the marks for Week 7-9 material, 1/3 of the marks for Week 10-12 material.

Exam preparation (cont.)

- It is important to understand the proofs and to be able to follow each step in the proofs.
- However, given the length and the structure of the exam (MCQ & SAQ) you will not be required to write any long proofs.
- The main emphasis is on understanding how and why algorithms work, including what are their running times (and why they have such running times).
- You should be able to apply the discussed algorithms to any given example, and know the differences as well as limitations of the algorithms.
- Please read the textbook (provides also additional exercises/examples).

Thank you!

Best of luck with your exams!

Note: I will keep my online office hours on 26 Oct, 1-2pm. If you cannot make it let me know by email and we can schedule another time.