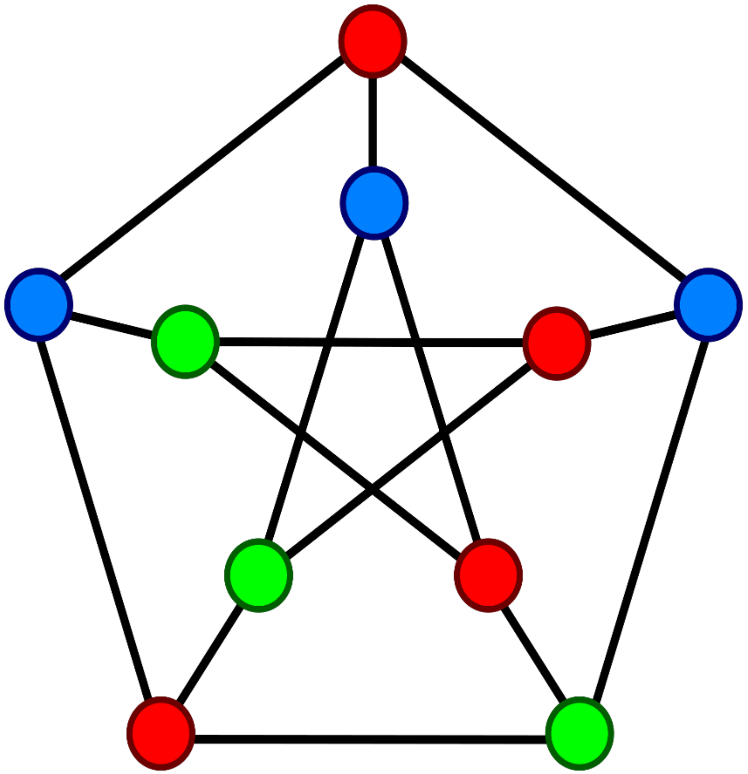
Lecture 13: Summary

William Umboh
School of Computer Science

Note: Slides may not contain all important material. Listen to the lecture recording as well.



Aims of this unit

This unit provides an introduction to the design and analysis of algorithms. We will learn about

- (i) how to reason about algorithms rigorously: Is it correct? Is it fast? Can we do better?
- (ii) how to develop algorithmic solutions to computational problems

Assumes:

- basic knowledge of data structures (stacks, queues, binary trees) and programming at level of COMP2123

- discrete math (graphs, big O notation, proof techniques) at level of MATH1004/MATH1064

How to design algorithms

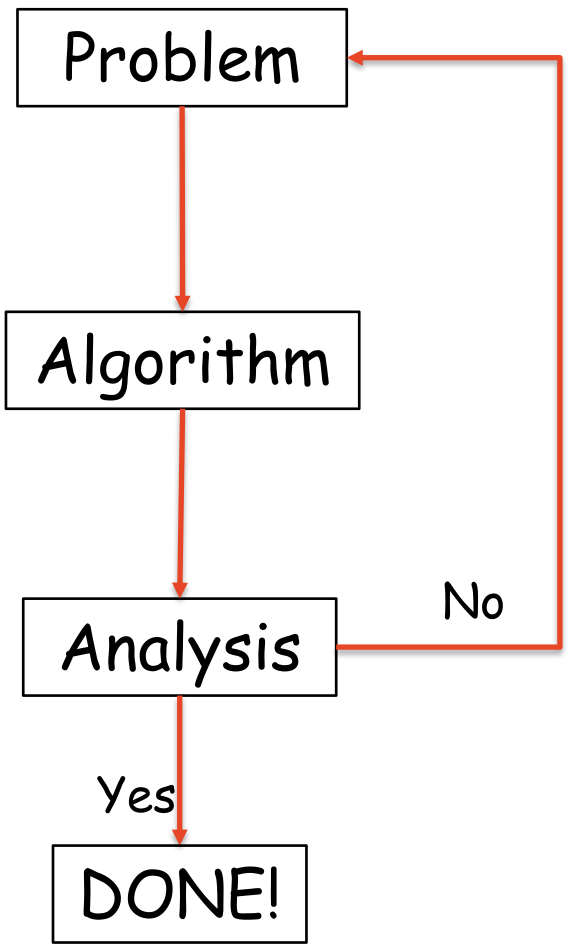
Step 1: Understand problem

Step 4: Better understanding of problem

Step 2: Start with simple alg.

Step 5: Improved alg.

Step 3: Does it work? Is it fast?



Main Themes

- Induction:
  - Proof technique
  - Algorithm design method: Greedy, Divide-and-conquer and DP

- Reduction:
  - Algorithm design: Reduction to flows
  - Hardness: Reduction from NP-complete problems

Roadmap [W2 – 10]

Greedy

NP-hardness

Divide & Conquer

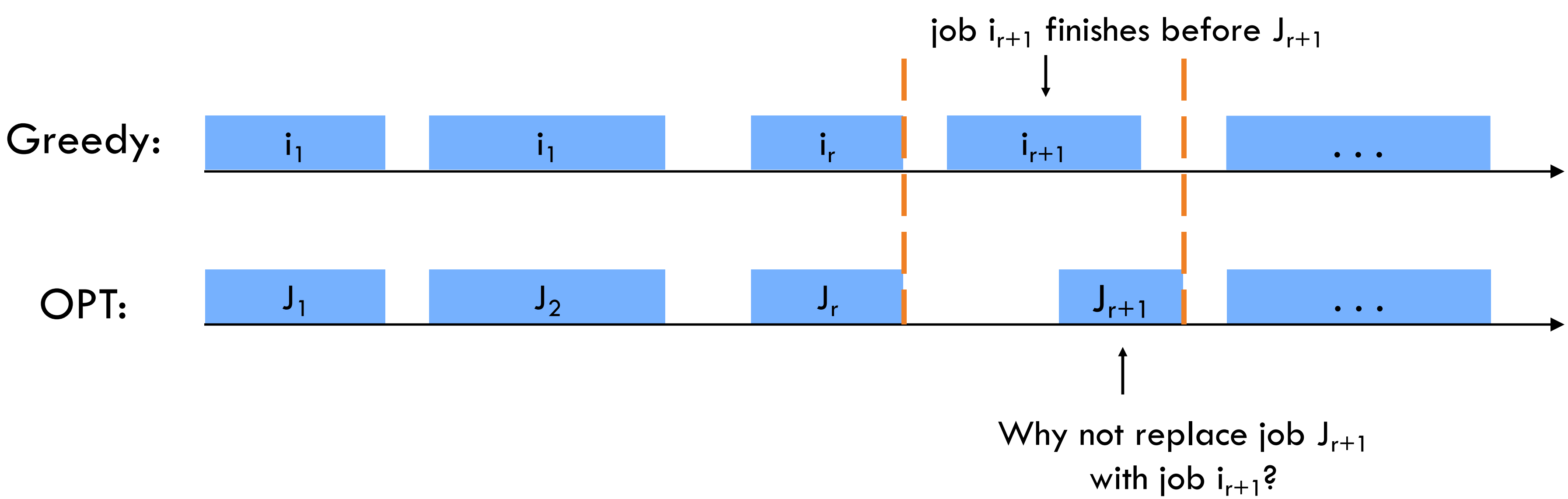
Reductions

Network Flow and Applications

Dynamic Programming

Overview – Greedy Algorithms

- Greedy algorithms
  - Greedy technique
  - Standard correctness proof: exchange argument, lower bound
  - Applications: Scheduling, Caching/paging



Overview – Divide and Conquer

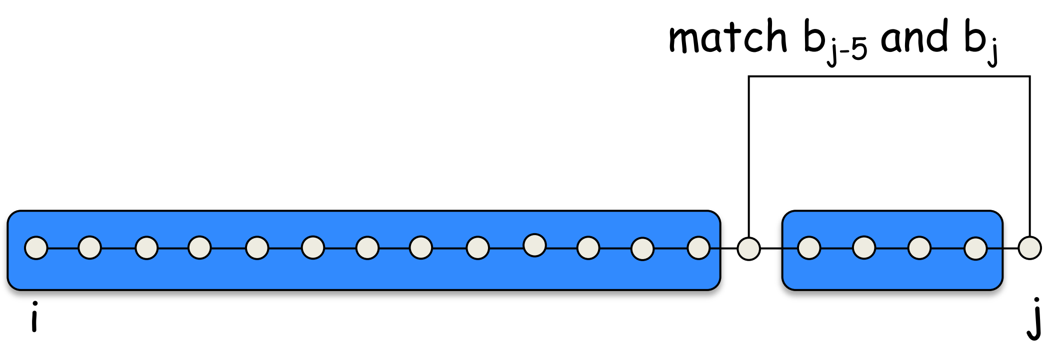
- Divide-and-Conquer algorithms
  - General technique: divide, solve and combine
  - Recursion: How to state and solve a recursion (unrolling, Master method)
  - Standard correctness proof: Induction
  - Applications: Mergesort, Inversions, Closest Pairs of Points

A	L	G	O	R	I	T	H	M	S			
A	L	G	O	R		I	T	H	M	S	divide	$O(1)$
A	G	L	O	R		H	I	M	S	T	sort	$2T(n/2)$
A	G	H	I	L	M	O	R	S	T		merge	$O(n)$

$\Rightarrow \quad T(n) = O(n) + 2T(n/2) = O(n \log n)$

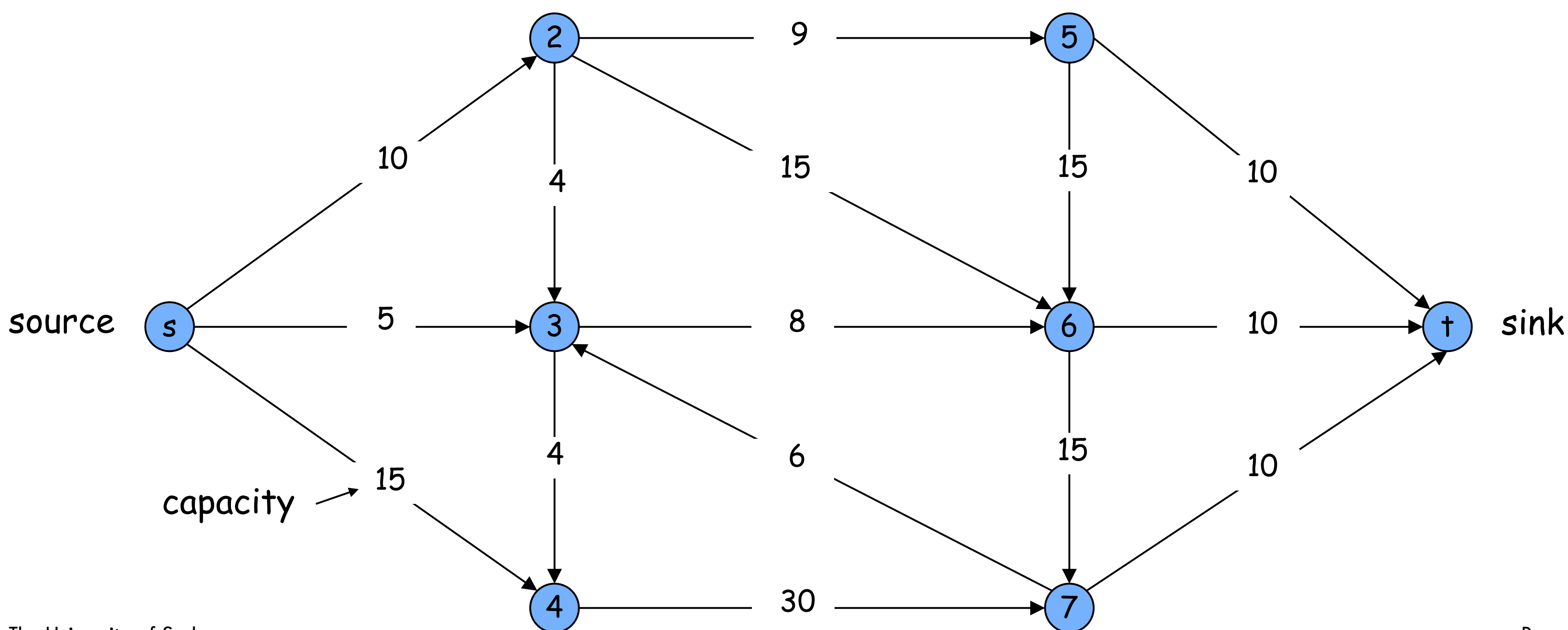
Overview – Dynamic programming

- Dynamic programming
  - General technique: Subproblems and recurrence
    - Define subproblems
    - Define recurrence linking subproblems
  - Correctness proof: Justify recurrence, base cases
  - Applications: Knapsack, weighted scheduling, RNA, Bellman-Ford,…



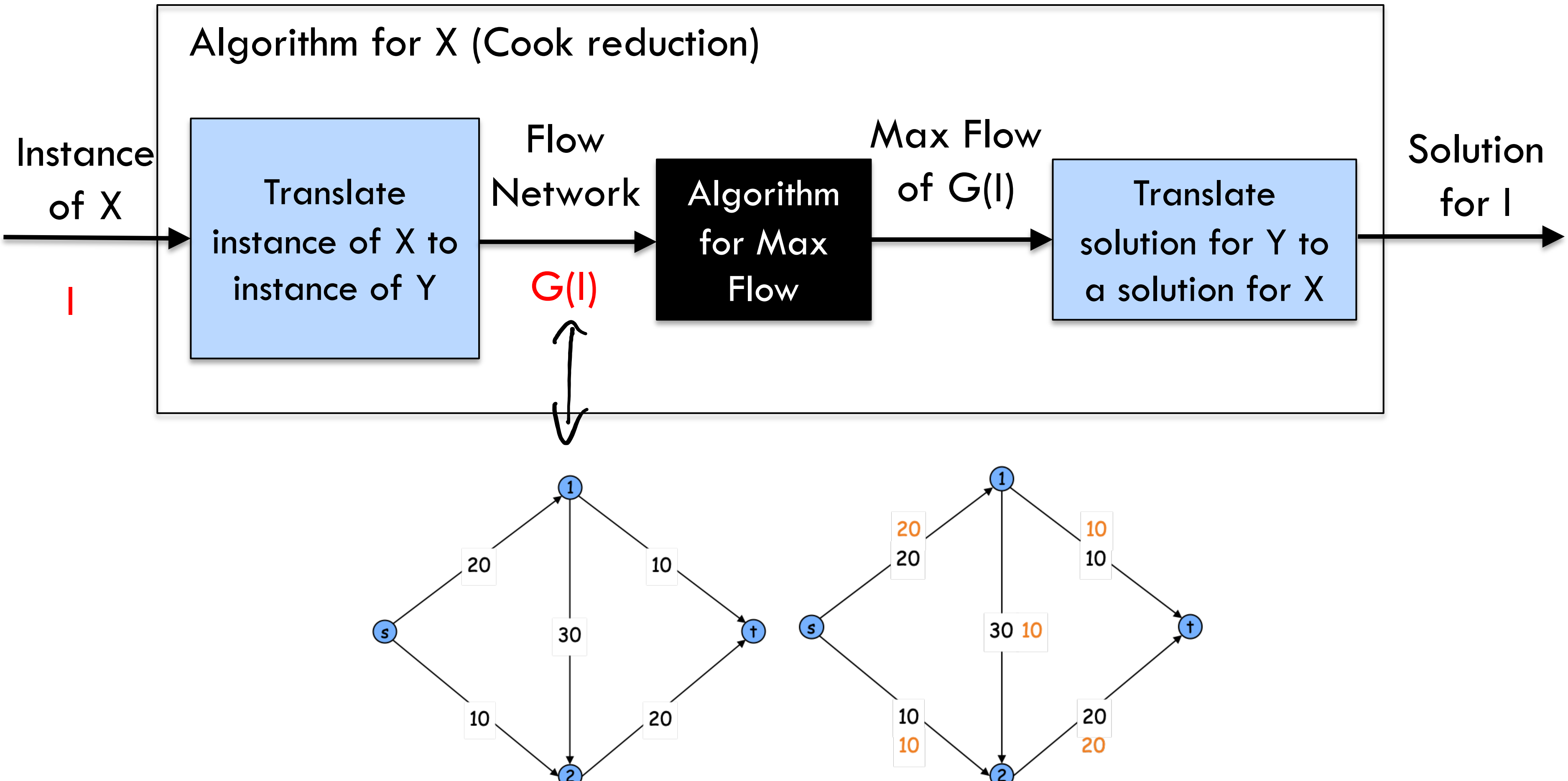
Overview – Flow Networks

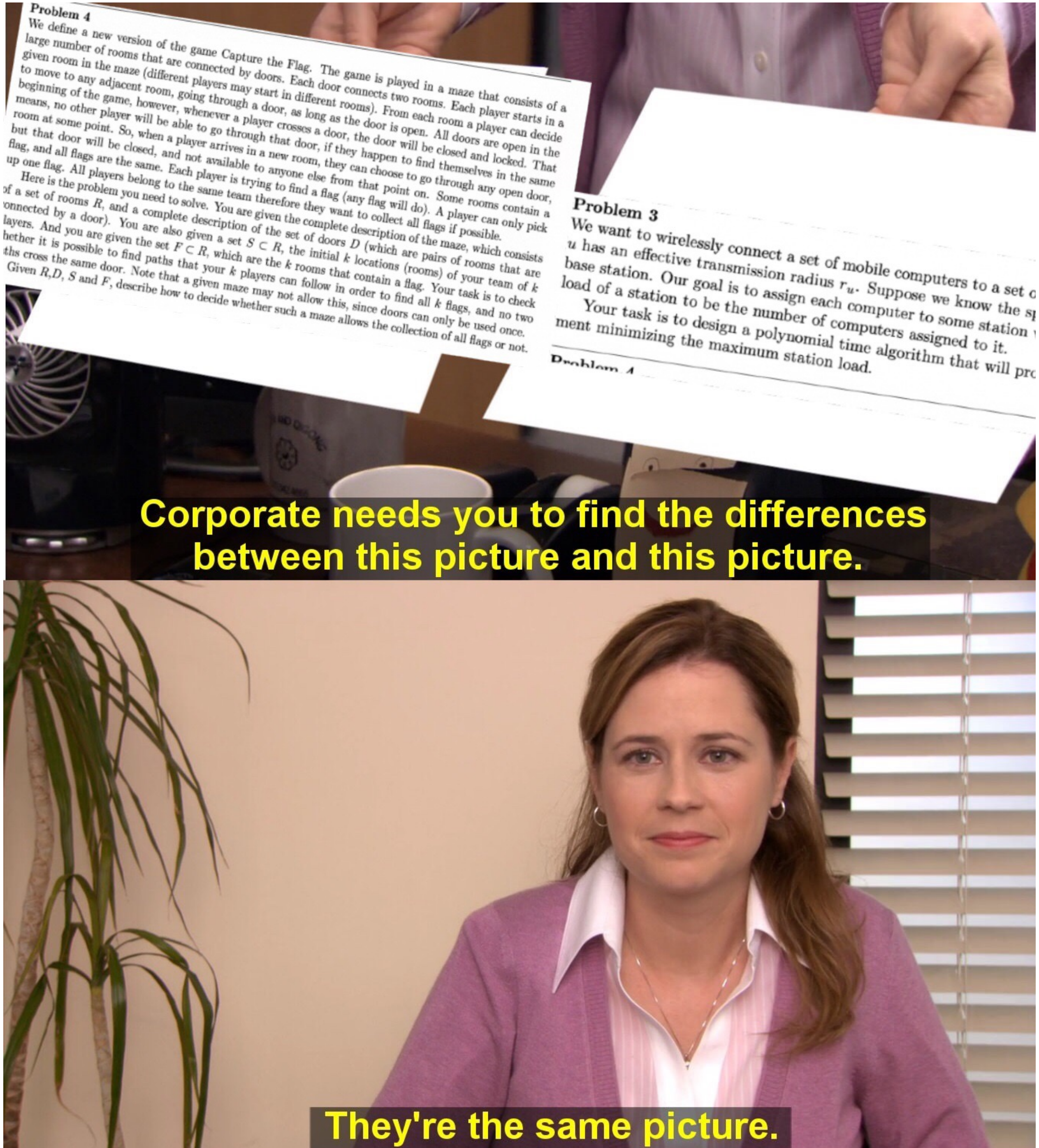
- Flow networks
  - Properties of flow network: max flow, min cut, integer lemma,…
  - General technique: reduce to a flow network
  - Correctness proof: Solution for X ⇔ Solution for FN
  - Applications: matching, edge-disjoint paths, circulation,…



Reduction to Max Flow

A reduction from Problem X to Max Flow is an algorithm of the following form:

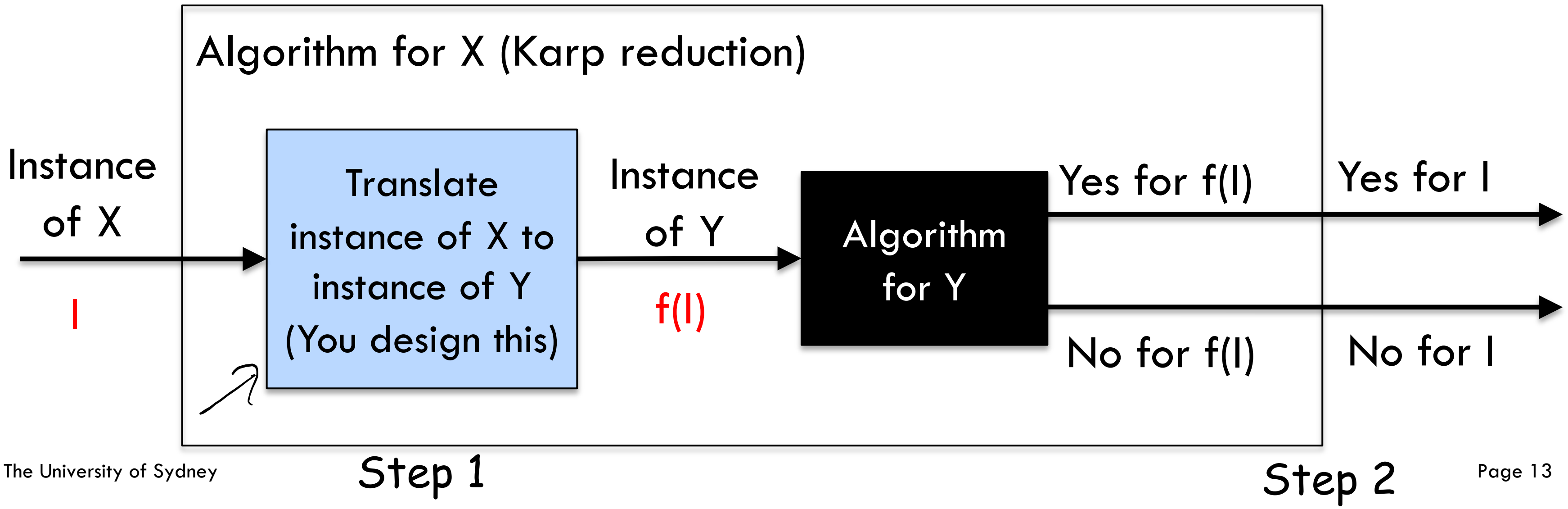



Problem 4
Problem 3
We want to wirelessly connect a set of mobile computers to a set o
u has an effective transmission radius r_u. Suppose we know the sp
base station. Our goal is to assign each computer to some station
load of a station to be the number of computers assigned to it.
Your task is to design a polynomial time algorithm that will pro
ment minimizing the maximum station load.
Corporate needs you to find the differences
between this picture and this picture.
They're the same picture.

Overview – NP-completeness

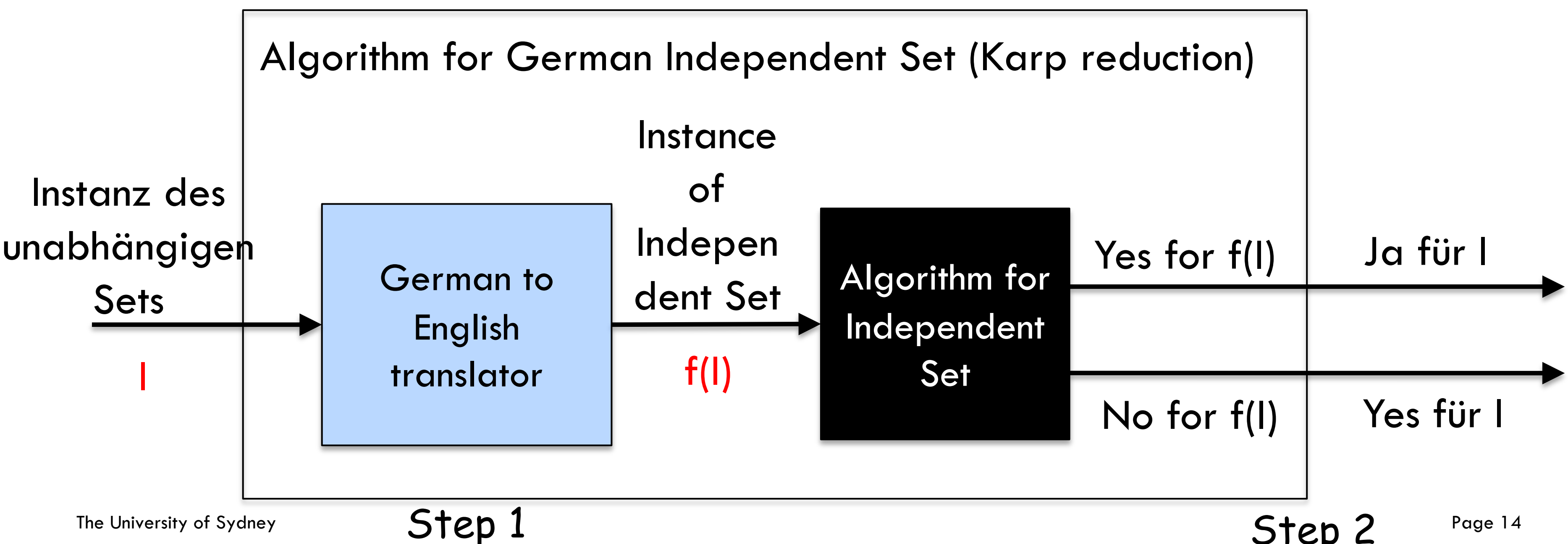
Complexity

- Polynomial-time reductions
  - Gadget reductions
- Classes: P, NP, NP-complete, NP-hard
- How to prove that a problem belongs to P/NP/NP-complete
- Understand the NP-complete problems in lectures.
- When proving NP-completeness, must use Karp reduction, not Cook.



Overview – Karp Reductions

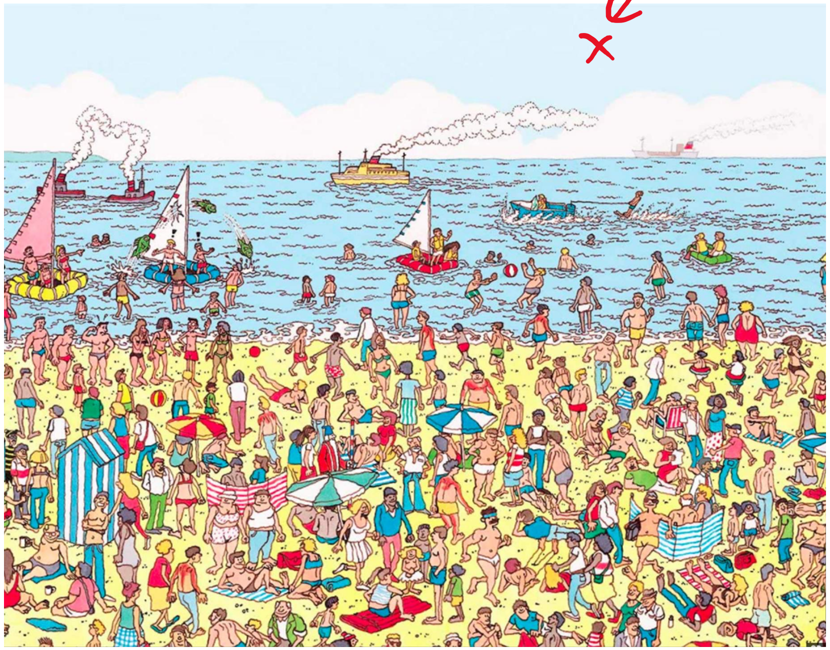
– I like to think of a Karp reduction as a one-way translator from one problem to another similar to language translations



Real-World Example of Verifier: Where’s Waldo

this location is not a valid certificate

- Decision problem: Is Waldo in the picture?
- Search problem: Where’s Waldo?
- Certificate: A location in the picture
- Certificate is correct if Waldo is at that location and incorrect otherwise
- Certifier checks if Waldo is indeed at the given location



Overview – Coping with hardness

- Coping with hardness
  - Understand the basic concepts:
    - Dynamic programming and greedy on trees
    - Restricted instances
    - Approximation algorithms
- ~~Coping with uncertainty~~
  - ~~Online algorithms~~

More algorithms?

- COMP3530: Discrete Optimization (S2)
  Lecturer: Julian Mestre

- Other opportunities:
  - Project units such as SCDL3991, etc
  - Vacation projects
  - Honours (many of our students have won the University Medal; 4/4 of my students have won it)
- Algorithms group
  - Joachim Gudmundsson, Andre van Renssen (Geometry)
  - Julian Mestre, me (Optimization)
  - Clement Canonne (Randomized algorithms, distribution testing)
  - Sasha Rubin (Logic)
  - Lijun Chang (Graph algorithms)
  - Overview of research in advanced lecture

Please get in touch with us for more information

Exam for COMP3027

Time: 10 minutes reading time

2 hours

Number of problems: 4 (ordered from easiest to hardest…imo)

2 short-answer questions assessing surface-level knowledge and ability to analyse correctness and running time of a given algorithm [10 marks each]

2 design questions: greedy, divide-and-conquer, dynamic programming, flows, NP-completeness, reductions [20 marks each]

Separate Canvas site Final_Exam for: COMP3027 and Final_Exam for: COMP3927

See Practice Final on Ed. No solutions provided. Discuss your solutions on Ed.

Exam Conditions

Open-book: You must write everything in your own words. No copying from any source.

You must do exam on your own. No communication with anyone else.

Submissions must be type-written using LaTeX or word-processing software. Hand-written solutions not accepted. Exceptions allowed for illustrations.

Submit only your answers. Do **not** copy the questions.

Exam Tips

Practice on problems under exam conditions

Memorise key definitions, proof techniques, and make your own cheat-sheet. Don't waste time looking these up during the exam

Have a template ready to go, especially if you are using LaTeX. Don't waste time with LaTeX compilation errors during the exam.

Resist the temptation to write a perfect solution on the first go. Have something written for all the questions and then return to edit.

Caution: Do not share cheat-sheets and templates.

Please remember to fill in the unit of study evaluation

- https://student-surveys.sydney.edu.au/students/

What was good? What was bad?

- Tutor and tutorials
- Were the extra resources, e.g. exemplars, useful?
- Assignments, feedback

Example of changes based on previous years' feedback:

- More programming exercises
- Faster marking of assignments
- Exemplars for assignments
- Assignment

Thanks for taking the class!

Good luck on the exam!