The University of Sydney **School of Information Technologies**

Unit of Study: COMP3310/3610 Theory of Computation

Assignment/Project Title Page

Individual Assessment

Assignment name: ___ Assignment - Problem set 2 _____

Tutorial time: __ Tuesday 12noon _____

Tutor name: __ Florian Verhein _____

Declaration

I declare that I have read and understood the *University of Sydney Student Plagiarism: Coursework Policy and Procedure,* and except where specifically acknowledged, the work contained in this assignment/project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.

I understand that understand that failure to comply with the *Student Plagiarism: Coursework Policy and Procedure* can lead to severe penalties as outlined under Chapter 8 of the *University of Sydney By-Law 1999* (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

Student Id: _____

Student name: _____

Signed: _____

Date: _____

COMP3310/3610: Theory of Computation

Problem set 2

due 12noon Tuesday October 17, 2006 (week 12)

Total marks: 100. This problem set is 10% of the final raw mark.
This problem set has 5 questions. There is different set of questions for COMP3310 and COMP3610-Advanced. Those enrolled in COMP3310 should answer questions labelled N1-N5, and those enrolled in COMP3610 advanced should answer the questions labelled A1-A5. The questions are listed separately for the two streams. There are common questions that appear in both problem sets.

Assume that the alphabet is $\Sigma = \{0, 1\}$ unless stated otherwise. Please submit the University "Cover sheet for an individual project/assignment" with your answers.
This is an individual assessment; submit your own write-up of your solutions. **Collaboration** and discussion of the problems in this assignment is **encouraged**, but please write your solutions on your own and list the names of the people you collaborated with for this problem set.
Electronic submissions are much prefered. Please send a pdf of your solutions to `tasos@it.usyd.edu.au`. Hard copies (**including a signed University "Cover sheet for an individual project/assignment"**) can be submitted at the School of IT building, 2nd floor, reception area, at the unit coordinator's mailbox or at the COMP3310 submission box.

Problem set 2 for COMP3310

N1 **(Polynomial time algorithms under assumptions, 15 marks)** (Problem 7.36p298)
Show that if $P = NP$, a polynomial time algorithm exists that produces a **satisfying assignment** when given a satisfiable Boolean formula.
Note: The algorithm you are asked to provide computes a function, but NP contains languages/decision problems, not function problems. The $P = NP$ assumption implies that SAT is in P, so testing satisfiability is solvable in polynomial time. But the assumption doesnt say how this test is done, and the test may not reveal satisfying assignments. You must show that you can find them anyway.

N2 **(Proving NP-completeness, 20 marks)** (Problem 7.26p297 from Sipser)
You are given a box and a collection of cards as indicated in the picture p297 sipser textbook. Because of the pegs in the box, each card will fit into the box in exactly two ways. Each card contains two columns of squares; these are arranged symmetrically on the card, so that if the card is flipped over, the squares appear in the same places. Now, for a given card, some of these squares may or may not be punched out. A solution is a placement of cards into the box so that the bottom of the box is completely covered. Show that the language

> $PUZZLE = \{< c_1, \ldots, c_k > \mid$ each c_i represents a card and this collection of cards has a solution$\}$

Show that $PUZZLE$ is NP-complete.

N3 **(Aproximation algorithms, 15 marks)**
Show how to improve the approximation ratio of the 2-approximation **makespan** algorithm to a $(2 - \frac{1}{m})$-approximation polynomial time approximation algorithm, where m is the number of the machines. *(10 marks)*

N4 **(Approximation algorithms, 25 marks)**
A thief breaks into someones apartment. He knows that he has limited time and can only take with him one load of loot. He has a bag of capacity b with him which we will call a knapsack. The set of valuable items in the apartment is denoted by X, and for each $x_i \in X$ there is an associated value (or profit) p_i and a size a_i. The thief would like to fill his knapsack up to its capacity with items that will maximize his profit. The problem is to help out the thief in deciding which subset of items he should choose to steal.
The maximum **knapsack** problem is defined as follows.
Instance: Finite set X of items, and for each $x_i \in X$, a corresponding value $p_i \in \mathbb{Z}^+$(positive integer) and a size $a_i \in \mathbb{Z}^+$, and a positive integer b.
Desired solution: A set of items $Y \subseteq X$ such that $\sum_{x_i \in Y} a_i \leq b$, that maximizes the total value of the chosen items, $\sum_{x_i \in Y} p_i$.
Convince yourself that the knapsack problem is NP-complete (no need to include this in your answer).
First, consider the following greedy algorithm. First sort the items according to their value. Then just add in your knapsack items starting from the most valuable ones until the knapsack is full. This algorithm will **not** give you a good approximation algorithm. Give an example (or examples) of a knapsack instance (input items their values and sizes) for which the given greedy algorithm is more than a 2-factor away from the optimal (for example, you could come up with an instance that you could fit items in the knapsack of total value say x but the given algorithm would choose items whose total value is less than $x/4$).
Find a polynomial time approximation algorithm for the maximum knapsack problem that achieves a solution that is at most a factor 2 away from the optimal (a 2-approximation).
Hint: A different greedy algorithm will do. Perhaps you can sort the items according to something more useful that also considers their size.

N5 **(More NP-completeness, 25 marks)** (Problem 7.34p298 from Sipser)
A subset of the nodes of a graph G is a **dominating set** if every other ndoe G is adjacent to some node in the subset. Let $DOMSET = \{< G, k > | G$ has a dominating set with k nodes$\}$. Show that it is NP-complete by giving a reduction from vertex cover.

Problem set 2 for COMP3610 - Advanced

A1 **(Polynomial time algorithms under assumptions, 15 marks)**
Prove that if $P = NP$ then we can factor integers in polynomial time.
Note: The algorithm you are asked to provide computes a function, but NP contains languages/decision problems, not function problems. The $P = NP$ assumption implies that the decision problem associated with factoring is in P, but the assumption doesnt say how this problem is done in polynomial, and the algorithm may not reveal factors. You must show that you can find a factoring by using only the fact that decision problems in NP can be done in poly time.

A2 **(Proving NP-completeness, 20 marks)** (Problem 7.26p297 from Sipser)
You are given a box and a collection of cards as indicated in the picture p297 sipser textbook. Because of the pegs in the box, each card will fit into the box in exactly two ways. Each card contains two columns of squares; these are arranged symmetrically on the card, so that if the card is flipped over, the squares appear in the same places. Now, for a given card, some of these squares may or may not be punched out. A solution is a placement of cards into the box so that the bottom of the box is completely covered. Show that the language

> $PUZZLE = \{< c_1, \ldots, c_k > \mid$ each c_i represents a card and this collection of cards has a solution$\}$

Show that $PUZZLE$ is NP-complete.

A3 **(Approximation algorithms, 15 marks)**
Show how to improve the approximation ratio of the 2-approximation **makespan** algorithm to a $(2 - \frac{1}{m})$-approximation polynomial time approximation algorithm, where m is the number of the machines.

A4 **(Approximation algorithms, 25 marks)**
A thief breaks into someones apartment. He knows that he has limited time and can only take with him one load of loot. He has a bag of capacity b with him which we will call a knapsack. The set of valuable items in the apartment is denoted by X, and for each $x_i \in X$ there is an associated value (or profit) p_i and a size a_i. The thief would like to fill his knapsack up to its capacity with items that will maximize his profit. The problem is to help out the thief in deciding which subset of items he should choose to steal.
The maximum **knapsack** problem is defined as follows.
Instance: Finite set X of items, and for each $x_i \in X$, a corresponding value $p_i \in \mathbb{Z}^+$(positive integer) and a size $a_i \in \mathbb{Z}^+$, and a positive integer b.
Desired solution: A set of items $Y \subseteq X$ such that $\sum_{x_i \in Y} a_i \leq b$, that maximizes the total value of the chosen items, $\sum_{x_i \in Y} p_i$.
Convince yourself that the knapsack problem is NP-complete (no need to include this in your answer).
First, consider the following greedy algorithm. First sort the items according to their value. Then just add in your knapsack items starting from the most valuable ones until the knapsack is full. This algorithm will **not** give you a good approximation algorithm. Give an example (or examples) of a knapsack instance (input items their values and sizes) for which the given greedy algorithm is more than a 2-factor away from the optimal (for example, you could come up with an instance that you could fit items in the knapsack of total value say x but the given algorithm would choose items whose total value is less than $x/4$).
Find a polynomial time approximation algorithm for the maximum knapsack problem that achieves a solution that is at most a factor 2 away from the optimal (a 2-approximation).
Hint: A different greedy algorithm will do. Perhaps you can sort the items according to something more useful that also considers their size.

A5 **(Approximation algorithms, 25 marks)**
Given a large collection of **protein** molecules whose properties we do not quite understand we would like to build a much smaller set of representatives that we will call **representative set**, that summurizes the properties of the entire collection. The properties we would like the representative set to have, are the following two:

- The set should relatively small to make it easier to study it
- Every protein in the full collection should be similar to some protein in the representative set

To make things more precise, let P be a large set of proteins. We define similarity on proteins by a *distance function* d: Given two proteins p and q their distance is given by a number $d(p,q) \geq 0$. In practice the sequence alignment measure is used typically as a distnce metric. There a predifined distance cut-off Δ that is specified as part of the input to the problem: Two proteins a re considered similar to one another if and only if $d(p,q) \leq \Delta$. We say that a subset of $R \subseteq P$ is a representative set if for every protein $p \in P$ there exists a representative protein $q \in R$ such that $d(p,q) \leq \Delta$. Our goal is to find the smallest representative possible.

Give a polynomial time algorithm that approximates the minimum representative set to within a factor of $O(\log n)$ where n is the size of the full collection P. Specifically, your algorithm should have the following property: If the minimum possible size of a representative set is s^* then your algorithm should return a set of size at most $O(s^* \log n)$. (Hint: The centre selection problem might be a good start)