Norwegian University of Science and Technology
Department of Computer and Information Science

Exam authors:
Magnus Lie Hetland
Ole Edsberg
Lars Greger Hagen

Quality control:
Pål Sætrom

Contacts during exam:

Magnus Lie Hetland	(918 51 949)
Ole Edsberg	(952 81 586)

Final Exam in TDT4125 Algorithm Construction, advanced course

Wednesday May 22, 2013
Time: 09:00 – 13:00

Aids: (B) All printed/handwritten; specific, simple calculator

Language: English

Please read the entire exam before you start, plan your time, and prepare any questions for when the teacher comes to the exam room. Make assumptions where necessary. Keep your answers short and concise. Long explanations that do not directly answer the questions are given no weight.

There are 20 subtasks. Each subtask counts for 5 % of the final score.

Problem 1

a) Describe how the Minimum Set Covering problem can be solved with the branch-and-bound method. Describe only the procedures for *branching* and *bounding* (including the pruning of sub-trees).

Solution:

Branching: Assign one as-of-yet unassigned set to be included in the set cover (left-hand branch) or excluded from the set cover (right-hand branch).

Bounding: the pessimistic bound for the cost of the optimal solution is the smallest cost of a solution seen so far (in a leaf node). The optimistic bound for the cost of the best solution in a subtree is the number of sets included already in the path from the root plus the minimum number of the remaining sets needed to produce a sum of cardinalities that equals or exceeds the number of not-yet-covered objects. We prune a subtree if the optimistic bound for the subtree is worse (has greater cost) than the pessimistic bound for the cost of the optimal solution.

Problem 2

a) Define a neighborhood function that will allow the basic local search scheme to be applied to the Minimum Set Covering problem.

Solution: A solution is a bit string giving inclusion/exclusion for each set. The neighbors of a solution x are all solutions that both (i) can be obtained by flipping at most one bit of x and (ii) represent valid set coverings. Students who propose a flip-a-bit neighborhood without the feasibility requirement should get 50% score.

b) The makespan scheduling (MS) problem is a minimization problem where we have to find a *schedule* for n jobs with processing times $p_1, p_2, \ldots, p_n$ on $m \geq 2$ identical processors in such a way that all processors get at least one job, and the time until all jobs are completed is minimized. A schedule is just an assignment of each job to a processor.

For example, an input instance of MS could have $n = 3$ jobs $\{1, 2, 3\}$ with processing times $p_1 = 3$, $p_2 = 2$, $p_3 = 1$ and $m = 2$ processors. An optimal solution for this input instance is the schedule that assigns the job 1 to the first processor and the jobs 2 and 3 to the second processor. We could write this schedule as a set of sets $\{\{1\}, \{2, 3\}\}$. The cost (time to completion) of the optimal solution is 3.

A possible neighborhood for MS can be defined by letting the neighbors of a schedule x be the set of all schedules obtainable from x by choosing one job from a processor with more than one job, and then assigning the chosen job to any processor. The neighbors of the schedule $\{\{1\}, \{2, 3\}\}$ would be the set of three schedules
$\{\{\{1\}, \{2, 3\}\}, \{\{1, 2\}, \{3\}\}, \{\{1, 3\}, \{2\}\}\}$.

Does there exist any input instances for MS where local search with this neighborhood can get stuck in a local, non-global optimum? If you answer no, prove it. If you answer yes, give an example input instance where local search could get stuck, and write the solution at a local, non-global optimum for your example.
Solution: It can get stuck. An input example where it can get stuck is an instance with two processors and jobs $\{1, 2, 3, 4\}$ with sizes $p_1 = 1, p_2 = 2, p_3 = 2, p_4 = 3$. Here, we have a global optimium at $\{\{1, 4\}, \{2, 3\}\}$ and a local, non-global optimum at $\{\{1, 2\}, \{3, 4\}\}$.

c) Your friend claims to have found an exact polynomial-time searchable neighborhood for an NP-hard problem X. Your friend claims that this proves that P=NP because every problem in NP is polynomial-time reducible to X, and X can be solved in polynomial time with ordinary local search using the newly found neighborhood. What is wrong with your friend's reasoning?
Solution: Existence of an exact polynomial-time searchable neighborhood does not imply that the problem is solvable in polynomial time with local search, because local search might still have to perform a superpolynomial number of transitions.

d) Give an example of an optimization problem for which there exists an exact polynomial-time searchable neighborhood. It's OK if the problem is a toy example that you invent just for this exercise.
Solution: Toy example: Maximize the sum of a bit string with neighbors given by flipping arbitrary bits. Non-toy example: Linear programming with the neighborhood used by the SIMPLEX algorithm.

Problem 3

a) In the Maximum Clique Problem (MAX-CL) you are given a graph $G = (V, E)$ and have to find a subset $S \subseteq V$ such that every pair of vertices in S has an edge between them and the size of S is maximized. Reduce MAX-CL to IP or 0/1-LP and show how you would relax it to (real-valued) LP.
Solution: Goal: maximum. $|V|$ variables: x_k is 1 if vertex k is part of the clique and 0 othervise. Constraints: For each pair of vertices i and j: $x_i + x_j \leq a_{ij} + 1$, where a_{ij} is 1 if there is an edge between vertices i and j and 0 otherwise. Binarity constraints: For each vertex i, $x_i \in \{0, 1\}$. Relaxation: replace binarity constraints on x_i with $0 \leq x_i \leq 1$.

b) Assume that you have an instance x of some maximization problem. You reduce it to an instance $I(x)$ of IP, relax $I(x)$ to an instance $I_{rel}(x)$ of LP, and find the dual $Dual(I_{rel}(x))$. Let $Opt(x)$ be the cost of an optimal solution to x. Express as precisely as possible what you can infer about the relationship between $Opt(x)$ and the value α in each of the following cases. Use the operators $=$, $\neq$, $>$, $\geq$, $<$ or $\leq$ where applicable.

1. A feasible solution to $I(x)$ has cost α.
   Solution: $Opt(x) \geq \alpha$
2. An optimal solution to $I(x)$ has cost α.
   Solution: $Opt(x) = \alpha$
3. A feasible solution to $I_{rel}(x)$ has cost α.
   Solution: Nothing
4. An optimal solution to $I_{rel}(x)$ has cost α.
   Solution: $Opt(x) \leq \alpha$
5. A feasible solution to $Dual(I_{rel}(x))$ has cost α.
   Solution: $Opt(x) \leq \alpha$
6. An optimal solution to $Dual(I_{rel}(x))$ has cost α.
   Solution: $Opt(x) \leq \alpha$
7. A feasible solution to $I_{rel}(x)$ has cost α, and a feasible solution to $Dual(I_{rel}(x))$ also has cost α.
   Solution: $Opt(x) \leq \alpha$
8. An optimal solution to $I_{rel}(x)$ has cost α, and an optimal solution to $Dual(I_{rel}(x))$ also has cost α.
   Solution: $Opt(x) \leq \alpha$
9. A feasible solution to $I_{rel}(x)$ has cost α, a feasible solution to $Dual(I_{rel}(x))$ also has cost α, and a feasible solution to $I(x)$ also has cost α.
   Solution: $Opt(x) = \alpha$
10. An optimal solution to $I_{rel}(x)$ has cost α, and an optimal solution to $Dual(I_{rel}(x))$ also has cost α, and a feasible solution to $I(x)$ also has cost α.
    Solution: $Opt(x) = \alpha$

c) Assume that you have an optimization problem which you reduce to IP and then relax to LP. If you have a proof that the optimal solutions to the relaxed problem always have integer values, what does this imply for the complexity class of the original problem? Give a reason for your answer.
Solution: The original problem is in PO, because it can be reduced to linear programming, which is in PO. You will also get full score if you answer P.

Problem 4 Consider the problem of choosing which people to invite for a party. Let *Peop* be the set of people you could invite, and let $f(x, y)$ be a function that for each pair of people

x and y gives a non-negative integer that measures how familiar x and y are with each other. We always have $f(x, y) = f(y, x)$. You want to invite a subset $\mathit{Inv} \subseteq \mathit{Peop}$ of people such that:

1. Each pair of different people a and b you invite have at least some familiarity with each other. In other words,
$$\forall(a, b \in \mathit{Inv}, a \neq b) f(a, b) > 0.$$
2. The sum of familiarity over all pairs of people you invite is maximized. In other words,
$$\sum_{a,b \in \mathit{Inv}, a \neq b} f(a, b)$$
is maximized.

a) Is this problem strongly NP-hard? Give a reason for your answer.
Solution: FIXME

Problem 5

a) Let's pretend that you have to solve the Minimum Vertex Cover problem with a Genetic Algorithm (GA). Specify the following components of a GA for this problem: (i) representation of solutions as individuals, ii) the crossover operator, (iii) the mutation operator, and (iv) the fitness function. Don't explain GAs in general. Make it simple.
Solution: Representation: A bit string with one bit for each vertex, indicating whether it's in the cover or not Crossover oprator: First child gets first half of first parent and second half of second parent. Second child gets what's left. Mutation operator: Randomly flip a bit. Fitness function: Fitness 0 if the candidate is not a valid vertex cover. Otherwise, fitness $\frac{1}{N}$, where N is the sum of the bit string. The student should have a reasonable plan for handling individuals that are not valid vertex covers, either in the fitness function or in the crossover and mutation operators.

b) The schema theorem is based on the assumption of a 100 % crossover rate. How would the schema theorem change if the crossover rate was 0 %? Give a reason for your answer.
Solution: The factor $\frac{n-\text{length}(s)}{n}$ would be dropped, because there is no longer any chance to destroy a schema through crossover.

Problem 6

a) Approximation algorithms give guarantees for how inaccurate they are, even though the optimum is unknown. How is this possible? Explain very briefly, in your own words.
Solution: The error is measured relative to an optimistic bound for the optimum.

A riddle: I can't necessarily give you the exact answer to your problem, but I can get as close as you ask me to, in polynomial time (though that time may grow exponentially as a function of my accuracy). I can even do this for any problems within any given distance from yours, although the further away they are, the longer it will take.

b) Who am I?
Solution: A superstable PTAS.

For a given family of edge-weight functions in $\text{TSP}_\triangle$, you have found a way of constructing a minimal/minimum spanning tree (MST) so that all the internal nodes have even degree, and all the leaves can be matched at a cost of $\frac{1}{\sqrt{5}} \cdot \text{OPT}$, where OPT is the cost of the optimum.

c) How would you use your MST and matching to create an approximation algorithm for $\text{TSP}_\triangle$, and what would ϵ be?
Solution: Use them in Christofides' algorithm, yielding $\epsilon = \frac{1}{\sqrt{5}}$.

A *pangram* or *holoalphabetic sentence* for a given alphabet is a sentence where every character of the alphabet occurs at least once. English examples include "The quick brown fox jumps over the lazy dog," and "Quick hijinx swiftly revamped gazebo." Consider the following generalized, simplified problem.

Simplified Pangram Problem (SPP)

Input: An alphabet Σ and a language (i.e., set of words) $L \subset \Sigma^*$, where $|L| = n$ and $|\Sigma| = m$.

Constraints: $\mathcal{M}(\Sigma, L) = \{S \subseteq L \mid \Sigma = \bigcup_{w \in S} w\}$.

Costs: For every $S \in \mathcal{M}(\Sigma, L)$, $cost(S, \Sigma, L) = |S|$.

Goal: *minimum*.

d) Describe an efficient approximation algorithm for SPP. What is the approximation ratio (δ) for this algorithm?
Solution: This is simply the Set Cover Problem (SCP), which can be approximated with the greedy algorithm, yielding $\delta = \ln \Sigma + 1$.

Problem 7 Let G be a connected graph, with $|V(G)| \geq 2$. Consider the following approximation algorithm for the minimum vertex cover problem (MIN-VCP). Find a depth-first tree T over G, and output the set S of all internal (non-leaf) nodes in T.

a) Show that S is a vertex cover for G, but that it might not be, had we used breadth-first search instead.
Solution: The only edges that could potentially not be covered by S would be edges between two leaves u and v in T. With BFS, such could exist, but not with DFS: Once DFS reaches u, it would immediately progress to v, making u an internal node.

b) Show that there is a matching $M \subseteq E(G)$ such that $|M| \geq |S|/2$.
Solution: Color each internal level of T red and black, alternatingly, and let the red and black levels contain n_r and n_b nodes, respectively. Red and black matchings of size n_r and n_b can be constructed from the internal nodes by connecting each to one arbitrarily chosen child. We have $|S| = n_r + n_b$, so the result follows.

c) If OPT is the minimum size of a vertex cover for G, show that $|S| \leq 2 \cdot \mathrm{OPT}$.
Solution: Every edge in the matching M must be covered, so $\mathrm{OPT} \geq |M| \geq |S|/2$.

Problem 8 The so-called *restricted shortest path* problem (RSP) is defined as follows: You are given a graph G with n nodes and m edges. Each edge $\{i, j\} \in E(G)$ has a positive integer-valued cost $c(i, j)$, and a positive integer-valued delay $d(i, j)$. The cost (delay) of a path is the sum of the costs (delays) along all of its edges. We wish to find the minimum cost over s-t-paths (for two given nodes $s, t \in V(G)$) whose total delay does not exceed a given threshold D. RSP is NP-hard, but there is an FPTAS for the problem, with a running time of $\mathcal{O}(mn/\epsilon)$.

Now consider the problem RSP_k, where the maximum cost over all the edges is less than n^k, for some constant k.

a) Show how the FPTAS for RSP can be used to solve RSP_k in polynomial time. What is the running time? (Explain your reasoning.)
Solution: The idea is to set ϵ to a value where the approximation must be identical to the (integer-valued) optimum, OPT, that is, $(1 + \epsilon) \cdot \mathrm{OPT} < \mathrm{OPT} + 1$. We know that $\mathrm{OPT} < (n - 1)n^k$, so we can simply let $\epsilon = 1/((n - 1)n^k)$. The running time is then $\mathcal{O}(mn/\epsilon) = \mathcal{O}(mn(n - 1)n^k) = \mathcal{O}(mn^{k+2})$.

Problem 9 A *hitting set* is defined as follows: Let U be the universe of objects and let $\mathcal{S}$ be a set of sets $\{S_1, S_2, \ldots, S_n\}$ where $S_i \subseteq U$ for all $S_i \in \mathcal{S}$. A hitting set for $\mathcal{S}$ is a set $H \subseteq U$ that intersects with all sets in $\mathcal{S}$. In other words, for all $S_i \in \mathcal{S}$, we have $H \cap S_i \neq \emptyset$. We call $|H|$ the size of the hitting set.

For example, if we have $U = \{a, b, c, d\}$, and $\mathcal{S} = \{\{a, b\}, \{b, c, d\}, \{c, d\}\}$, then $H = \{b, c\}$ is a hitting set for $\mathcal{S}$, and its size is 2.

The *hitting set problem* (HSP) is a decision problem defined as follows: You are given a universe of objects U, a set of sets $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ where $S_i \subseteq U$ for all $S_i \in \mathcal{S}$, and a positive integer k. The task is to answer yes if there exists a hitting set for $\mathcal{S}$ of size k, and no if not.

For example, consider the problem instance $U = \{a, b, c, d\}$, $\mathcal{S} = \{\{a, b, c\}, \{a, c\}, \{d\}\}$, $k = 2$. The answer for this instance is yes. The possible hitting sets are $\{a, d\}$ and $\{c, d\}$. If we change k to 1, the answer is no.

a) Consider the parameterization

$$Pat(U, \mathcal{S}, k) = \max\{k, d(\mathcal{S})\}$$

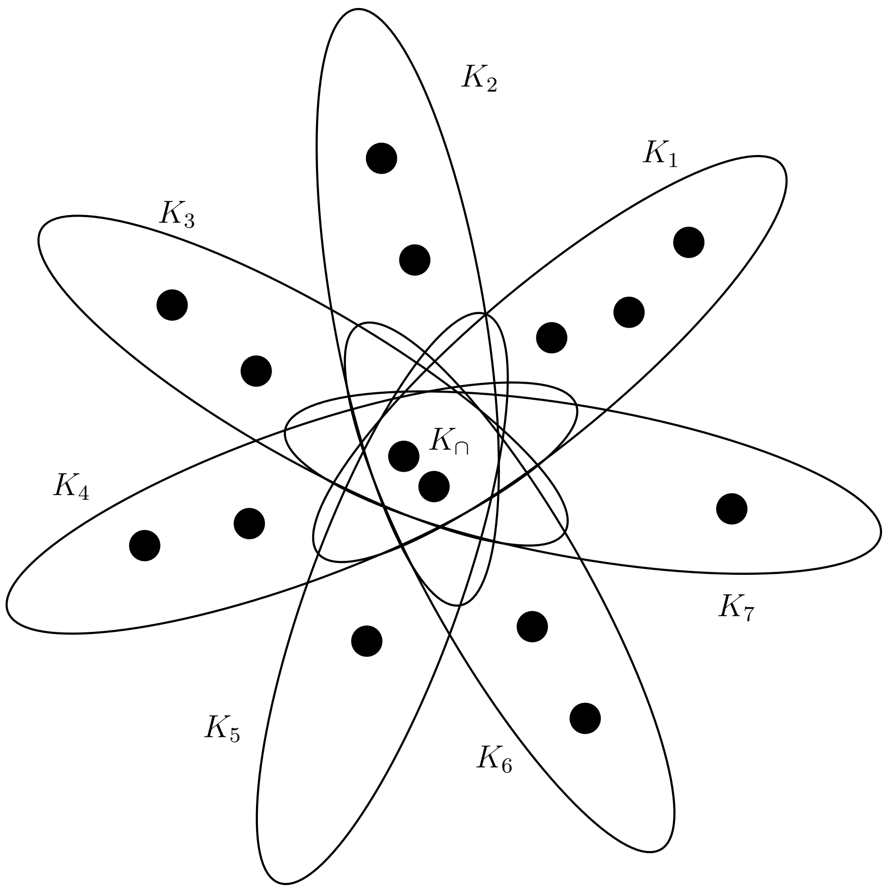
of HSP, where $d(\mathcal{S})$ is the size of the largest set in $\mathcal{S}$. Give a Pat-parameterized polynomial time algorithm for HSP.

Hint: See Appendix

Solution: While there are more than $k^{d(\mathcal{S})+1}d(\mathcal{S})!$ sets in $\mathcal{S}$, there is a sunflower S of size $k+1$ with intersection $S_\cap$, which can be found in polynomial time. If $S_\cap = \emptyset$, there is no solution. If $S_\cap \neq \emptyset$ the problem can be reduced to $((\mathcal{S} - S) \cup \{S_\cap\}, k)$.

After a polynomial amount of such reductions, the number of sets in $\mathcal{S}$ is bounded by $k^{d(\mathcal{S})+1}d(\mathcal{S})!$. Since the size of each set is bounded by $d(\mathcal{S})$, the input size is bounded by $d(\mathcal{S}) \cdot k^{d(\mathcal{S})+1}d(\mathcal{S})!$, which is bounded by a function of $Pat(U, \mathcal{S}, k)$.

Appendix



Sunflower: Sets $\mathcal{K} = \{K_1, K_2, \ldots, K_k\}$ form a *sunflower* if there exists a set $K_\cap$ such that for all $K_i, K_j \in \mathcal{K}$, $i \neq j$, the intersection $K_i \cap K_j$ is equal to $K_\cap$.

The sunflower lemma: If a collection of sets $\mathcal{K} = \{K_1, K_2, \ldots, K_m\}$ consists of $p^{d+1} \cdot d!$ sets, and all sets in $\mathcal{K}$ has size less than or equal to d, then $\mathcal{K}$ contains a sunflower of size $p + 1$. Furthermore, this sunflower can be found in polynomial time.

Observation: Assume that an instance $(\mathcal{S}, k)$ of the hitting set problem contains a sunflower $\mathcal{K}$ of size $k + 1$ with intersection set $K_\cap$. If $K_\cap = \emptyset$, there are $k + 1$ disjoint sets, so there is no solution. If $K_\cap \neq \emptyset$, the problem is equivalent to removing the sunflower sets from $\mathcal{S}$, and adding $S_\cap$ instead.