#5 Assignment - CMPT 405

Luiz Fernando Peres de Oliveira - 301288301 - lperesde@sfu.ca

November 30, 2018

#1

Because an edge covering of G is a set $A \subset E$ such that for every node $v \in V$, v is one of the vertices in at least one of the edges in A, we know that there is a polynomial algorithm that describes finding an edge cover of G, for example by simply running *BFS*. However, because finding the smallest possible set of edges $A^* \subset E$ that satisfy an edge covering is an optimization problem, we need to prove that finding such set can be done in polynomial time.

The intuition of our algorithm to minimize the number of edges will be that an edge covering cannot have subpaths of more than two edges, where the subpaths are all subsets of connected egdes in A. A simple proof would be: let A' be an edge covering of G and let p be a subpath formed by some of the edges of A', where $|p| > 2$. If p is $a \to b \to c \to d$, then $|A'| > |A^*|$, as there is an extra edge, (b, c) in this case, that could be removed, p can be split into two different subpaths $p_1 = a \to b$ and $p_2 = c \to d$, minimizing the edge covering and therefore $|A'|$ is not optimal and cannot equal $|A^*|$.

Because the property above holds true, where for every subpath $p \subset A^*$, $|p| = 1$ or $|p| = 2$, we can solve this problem by first computing a maximum matching on G (therefore covering every p, where $|p| = 1$), and then, for each vertex v left uncovered, covering v by adding an edge connecting to one of the already covered vertices (therefore augmenting the size of some p). Because the second part of the algorithm is trivial ($O(m + n), n = |V|, m = |E|$), the total running time of the algorithm is the same as Edmonds algorithm ($O(n^2m)$ or $O(\sqrt{m}n$ with improvements), thus, it can be implemented in polynomial time.

#2

The intuition here is that the *half-3-CNF-SAT* is NP-Complete, as the problem is a subset of the *3-SAT* problem. We then want to prove that it is possible to reduce the *3-SAT* problem to the *half-3-CNF-SAT* in polynomial time and finally, show that finding a truth assignment ϕ that satisfies exactly $k/2$ of the k clauses is NP-hard.

Proof. **Reducing 3-SAT to half-3CNF-SAT:** Because half-3-CNF-SAT problem is the problem of checking if exactly half of the clauses evaluate to true, then half of the clauses must evaluate to false. Imagine ϕ contains m clauses, then, in our reduction we will add three variables x, y and z and create a ϕ' with $4m$ clauses, such that ϕ' is an extension of ϕ, meaning that ϕ' contain all m clauses of ϕ and:

- m clauses of the form: $(x \vee \neg x \vee y)$ *// always true*
- $2m$ clauses of the form: $(x \vee y \vee z)$ *// either all true or all false*

Reduction in polynomial time: The reduction can be done in polynomial time because we only added three new variables, copied the m clauses from ϕ and created $3m$ clauses.

3-SAT has a solution iff half-3-CNF-SAT has a solution:

$\Longrightarrow$ Assume that there is a truth assignment ϕ that satisfies the m clauses. Then there are $2m$ clauses in ϕ' that are satisfied (the clauses in $phi+$ the m clauses that are always true and we let x, y and z be false so that $2m$ clauses are false. Therefore, half of the clauses are true and half o the clauses are false.

$\Longleftarrow$ Assume that there is a truth assignment that satisfies *half-3-CNF-SAT* or a truth assignment that makes half the clauses in ϕ' true and half false. As m clauses are always true in ϕ', $2m$ clauses cannot be true if *half-3-CNF-SAT* is satisfied because that would comprise $3m$ clauses, therefore breaking it. Thus, the $2m$ clauses must be false so that ϕ is true and meaning that *3-SAT* is also satisfied.

NP-hard problem: Because *half-3-CNF-SAT* is NP-complete, its optimization version (finding a set of exactly $k/2$ clauses that are satisfied by ϕ) is NP-hard.

□

#3

We know that if we have an algorithm A deciding whether or not a CNF formula is satisfiable (SAT problem) in polynomial time, then it is the case that we can find a satisfying assignment for such formula in polynomial time. Let Γ be the CNF formula with n literals $(l_1, ..., l_n)$ that can take the values of either $\{T, F\}$ and finally, let τ be the satisfying assignment output for each literal in Γ. We can then find a satisfying assignment by:

Input: A, Γ
Apply $A(\Gamma)$ and return "unsatisfiable" if UNSAT
$\Gamma' \leftarrow \emptyset$ // $\Gamma' \subseteq \Gamma$
$\tau \leftarrow \emptyset$
for i **from** 1 **to** n **do**
    $\Gamma' \leftarrow \Gamma' \cup \{l_i\}$
    $l_i = F$
    if not $A(\Gamma')$ **then**
        $l_i = T$
    end if
    $\tau \leftarrow \tau \cup \{l_i\}$
end for
return τ

Since A runs in polynomial time, our algorithm runs in $O(A \cdot n)$ and therefore is polynomial.

#4

- The number of vertices is even by the **handshake theorem**: every finite undirected graph has an even number of vertices with odd degree.
- Every vertex has even degree because the previous step take all vertices of odd degree and find a minimum-weight perfect matching in the induced subgraph given by such vertices.
- It is possible make a Hamiltonian circuit by skipping repeated vertices as all vertices have even degree.
- Let TSP^* be the optimal traveling salesman tour. Removing an arbitrary edge from TSP^* produces a spanning tree T with total weight $w(T)$ less or equal to the total weight TSP^*. If you then number the vertices of odd degree around TSP^* and partition TSP^* into two sets of paths of odd and even numbers, then each set of paths corresponds to a perfect matching of the vertices of odd degree that matches the endpoints of each path, and the weight is at most equal to the weight of the paths. Since these two sets partitions the edges of TSP^*, one of the two has at most half of the weight of TSP^* and because of the triangle inequality, the same holds true for its corresponding matching. Adding the weights of the minimum spanning tree and minimum weight perfect matching we have an Euler tour at most $3(TSP^*)/2$.

#5

There is no k-approximation algorithm with $k < \frac{3}{2}$, unless $P = NP$.

Proof. We must reduce the *PARTITION* problem to *BIN PACKING*. For an instance $x_1, ..., x_n$ of *PARTITION*, we let $X = \sum x_i$ and consider an instance of *BIN PACKING* $p_i = 2x_i/X$. We can see that the minimum number of bins for *BIN PACKING* instance is 2 if and only if the answer to the *PARTITION* problem is *YES*. For means of contradiction, unless $P = NP$, assume that there was a k-approximation for *BIN PACKING* with $k < \frac{3}{2}$, then, whenever the exact minimum is 2, the algorithm would always find it, since 3 would mean an approximation factor $\geq 3/2$, thus, the reduction would be solved in polynomial time (which is not the case). The optimization of this problem is then NP-hard. □

Bin Packing First Fit: The first fit algorithm processes the items in arbitrary order, for each item, it tries to place it in the first bin that fits the item. If no bin is found, it opens a new bin, fit the item in it and repeat the whole process greedily.

Let BP^* be the optimal solution for the *BIN PACKING* problem. The first fit algorithm is within a 2-approximation factor of BP^* because the number of bins used by the algorithm does not exceed twice the optimal number of bins. It is not possible for 2 bins to be half full because the algorithm would have included the second item in the first bin and will not open another bin of half size. Because we may say that every new bin will only be opened after the previous bins are at least half full (remembering that smaller items might fit in larger bins). Because of this property, we have that at most twice of the space of all bins was not used and therefore the first fit FF solution is within twice the optimal number of bins (meaning that we might open two times more bins than we need and $FF \leq 2(BP^*)$).