COMP5711-HW2

WONG Yuk Chun (20419764) ycwongal@connect.ust.hk

Fixed Parameter Algorithms

Problem 1

First we define reduction of the problem by $x \in A$ as, remove x from A and remove all B_i which contains x, report whether there exists $k-1$ size of H that hits all remaining sets B_j.

Then we proof hitting set of size k exists if and only if for any B_i, there exists $x \in B_i$, such that the reduced problem by x has a $k-1$ size hitting set.

$\Rightarrow$: Suppose there is a hitting set $H \subset A$ and $|H| \leq k$, consider one of the set B_i, at least one of the elements $x \in B_i$ is in H. Then we fix one x remove all B_j that contains x, delete x from A, a hitting set of size $k-1$ for the leftover B_s still exists.

$\Leftarrow$: If size $k-1$ hitting set exist for the reduced problem, we can add back x and the removed B_i and recover to the original problem.

The contrapositive of above prove the cases of hitting set does not exist.

By the above lemma, we construct our algorithm,

1. If there is no any B_i, return such hitting set exists
2. If $k = 0$, return such hitting set does not exist
3. Pick a B_i
4. For each $x \in B_i$, we check whether there exists $k-1$ hitting set for the reduced problem without x
5. Report hitting set exists if hitting set exist in one of the subproblem, else return not exist

Since B_i has size at most c, there are at most c choices of x, each call of function at most spawn c recursive case. Since each recursive case reduce k by 1, and the recursion stop for $k = 0$, the recursion has depth at most k. For each recursive step, the algorithm loop through all $x \in B_i$ to compute the reduced problem and there are at most c of such x, where the reduction loop through all B_j to remove B_j that contains x, and there are total of m many B_j needed to be checked. Hence the run time of each recursive step has run time of $\mathrm{O}(cm)$. Let the run time be $T(k)$, we have

$$\begin{aligned} T(k) &\leq cT(k-1) + O(cm) \\ &\leq c(cT(k-2) + O(cm)) + O(cm) \\ &\leq c^2T(k-2) + (c+1)O(cm) \\ &\leq \dots \\ &\leq c^k + (c^{k-1} + \dots + c + 1)O(cm) \\ &= O(c^k kcm) = O(c^{k+1}km) \end{aligned}$$

Then we have $f(c,k) = kc^{k+1}$ and $p(m) = m$.

Problem 7

Problem 7a

Given an optimal tree decomposition of G with tree width w, we can prove the chromatic number by following algorithm.

Start with the root node, assign the $w+1$ vertices in it with $w+1$ distinct colors.

Recursively perform above step such that if vertex v in both child node and parent node, their assignment color should be the same.

Lemma: *Number of colors are enough*

Proof: Consider a node s and its child t, when there exist a new vertex v exists in t not in s, if s is size less than $w+1$, we have enough colors, else there is a vertex u in s not in t otherwise t has size greater than $w+1$, then we can use the color of u to color v

Lemma: *Any edge have different color at two ends*

Proof: By edge coverage property of tree decomposition, all edges have both ends contained in some tree node, and all vertices in same tree node are colored differently, this lemma is proved.

Lemma: *Vertices would not assigned with different colors*

Proof: By coherence property of the tree decomposition, given a vertex v, all tree nodes containing v are connected in the tree decomposition. Then when the algorithm can only see v consecutively, then color of v is assigned once only.

Problem 7b

We set up an algorithm to test whether a graph is k-chromatic.

Construct a dynamic programming recurrence relation similar to the weighted independent set on tree decomposition:

For a root node t of some subtree in the tree decomposition, V_t is the vertex set of the node, given a color assignment χ which is a mapping from any vertices to colors, we check whether we can assign colors to the vertices in node t respecting χ, and check whether subtrees under t with root $t_1, \ldots, t_d$ are also k-chromatic and respect χ.

We can check whether t respect χ by,

$$\forall u, v \in V_t, u \neq v, (u, v) \text{ is edge, color or u is different from v}$$

which has runtime of $O((w+1)^2)$

We can check whether all subtrees rooted at $t_1, \ldots, t_d$ is k-chromatic, and at the same time respect χ by: generate all the permutation of colors with length $w+1$, each validate it is aligned with χ, which have runtime $O(k^{w+1})$ and $O(w)$, for all subtrees the runtime is $O(k^{w+1}wd)$

Then for each node t, it takes $O((w+1)^2 + k^{(w+1)}wd) \leq O(k^{(w+1)}w^2d)$. As each t have $O(k^{w+1})$ possibility of input χ, and the d subtree cost can be charged once only during the building of dynamic programming table, the total runtime is expressible by
$O(k^{w+1}k^{w+1}w^2n) = O(w^{2w+4}n) = O(f(w)\text{poly}(n))$

Randomized Algorithm

Problem 1

Color each vertex randomly with the colors, each has same probability of $\frac{1}{3}$. Then consider an edge $e = (u, v)$, it satisfies the coloring condition by a probability of $\frac{2}{3}$ (for any color of u, v have 2 out of 3 choices to satisfy the coloring condition). Define X_i as a random variable such that

$$X_i = \begin{cases} 1 & \text{edge } i \text{ satisfy coloring condition} \\ 0 & \text{otherwise} \end{cases}$$

Then the expected number of edges X that satisfy the coloring condition is

$$E[X] = \sum_{e_i \in E} E[X_i] = \sum_{e_i \in E} \frac{2}{3} = \frac{2}{3}m \geq \frac{2}{3}c*$$

Then the algorithm is proved and its run line is linear to number of vertices.

Lemma: *The probability that a random assignment satisfies at least $\frac{2}{3}m$ is at least $\frac{1}{3}m$*

Proof: Let p_j be the probability that exactly j edges are satisfied

$$\begin{aligned} \frac{2}{3}m = E[X] &= \sum_{j \geq 0} jp_j \\ &= \sum_{j < \frac{2m}{3}} jp_j + \sum_{j \geq \frac{2m}{3}} jp_j \\ &\leq (\frac{2m}{3} - \frac{1}{3}) \sum_{j < \frac{2m}{3}} + m \sum_{j \geq \frac{2m}{3}} p_j \\ &\leq (\frac{2m}{3} - \frac{1}{3})1 + m \sum_{j \geq \frac{2m}{3}} p_j \\ \implies \sum_{j \geq \frac{2m}{3}} p_j &\geq \frac{1}{3m} \end{aligned}$$

If we repeat the algorithm $O(m)$ times, we amplifies the success probability to a constant.

Problem 7

Problem 7a

Consider a clause C_i of n variables, the probability that C_i is not satisfied is $\frac{1}{2^n}$. Consider the worst case that $n = 1$ for all clauses, each clause has probability of $\frac{1}{2}$ to be satisfied, then we can expect total of $\frac{k}{2}$ clauses to be satisfied.

There is not assignment that can satisfy more than 1 clauses for the following 2 clauses of 1 variable .

$$\begin{aligned} C_1 &= x_1 \\ C_2 &= \overline{x_1} \end{aligned}$$

Problem 7b

We assign x_i that appears in some single variable clause with higher probability $p \geq \frac{1}{2}$ to satisfy that clause, while other variable assigned $\frac{1}{2}$ probability. Consider clauses with $n \geq 2$ variables, probability to get satisfied, since $p \geq \frac{1}{2}$ the worst case is $(1 - \frac{1}{2^n}) \geq (1 - p^2)$ where it is a 2 variable clause that both variable exist in other single variable clause conflicting to it. Then we solve for p such that the equation also apply to $n = 1$, we have $p = 1 - p^2$, then $p = 0.618... \geq 0.6$. Then for clauses of any length, we have probability to get it satisfy of at least 0.6. Therefore as there are k clauses, the expected number of satisfied clauses is at least $0.6k$.

Problem 7c

The optimal assignment cannot satisfy the both conflicting clauses, then we can remove one of the conflicting clauses for each conflicting pair such that the optimal assignment remains the same. Suppose there are m pairs of conflicting single variable clauses,
$\text{OPT} = \text{total number of remaining clauses} = k - m$. Then we remove all the conflicting single variable clauses to satisfy the assumption in 7b, we can apply same algorithm to expect at least $0.6(k-2m)$ clauses satisfied. Then, for each conflicting single variable clauses, we add back one of them according to the variable assignment before, we can expect at least
$0.6(k-2m) + m \geq 0.6(k-m) = 0.6\text{OPT}$ clauses to be satisfied.

Lemma: *The probability that a random assignment satisfies at least 0.6OPT clauses is at least* $\frac{1}{5OPT}$

Proof:

Let p_j be the probability that exactly j clauses are satisfied, Z is the random variable of number of clauses satisfied, let $h = \text{OPT} = (k-m)$

$$\begin{aligned} 0.6OPT = E[Z] &= \sum_{j \geq 0} jp_j \\ &= \sum_{j < 0.6h} jp_j + \sum_{j \geq 0.6h} jp_j \\ &\leq (0.6h - 0.2) \sum_{j < 0.6h} p_j + h \sum_{j \geq 0.6h} p_j \\ &\leq (0.6h - 0.2)(1) + h \sum_{j \geq 0.6h} p_j \\ \implies \sum_{j \geq 0.6h} p_j &\geq \frac{1}{5h} = \frac{1}{5\text{OPT}} \end{aligned}$$

Since OPT is linear with k, we can repeat $O(k)$ times to reach a constant probability of getting the expected value.

Problem 11

Problem 11a

Define random variable Z_i as

$$Z_i = \begin{cases} 1 & \text{machine i does not receive any jobs} \\ 0 & \text{otherwise} \end{cases}$$

The probability that a machine does not receive one of the jobs = $(1 - \frac{1}{k})$

Since the job assignment is independent, a machine does not receive any jobs have probability $(1-\frac{1}{k})^k$, then $E[Z_i] = (1-\frac{1}{k})^k$.

Then the expected total number of machines does not assigned with any jobs

$$\begin{aligned} N(k) &= E[Z] \\ &= \sum_{i=1}^{k} E[Z_i] \\ &= \sum_{i=1}^{k} (1 - \frac{1}{k})^k \\ &= k(1 - \frac{1}{k})^k \end{aligned}$$

Then we have the expected fraction when $k \to \infty$

$$\lim_{k\to\infty} N(k)/k = (1-\frac{1}{k})^k = \frac{1}{e}$$

Problem 11b

Let $A(k)$ be the expected number of accepted jobs, then $R(k) = k - A(k)$. If we consider the expected number of machines that do not receive any jobs, k is the total number of machines, $A(k)$ is equivalent to number of machine that accept a job, then we have $A(k) = k - N(k)$. By combining the two equations, we have

$$\begin{aligned} R(k) &= k - A(k) \\ &= k - (k - N(k)) \\ &= N(k) \\ &= \frac{k}{e} \\ \frac{R(k)}{k} &= \frac{1}{e} \end{aligned}$$

Problem 11c

From above, expected number of machines assigned exactly 0 jobs = $\frac{k}{e}$

Probability that a machine is assigned exactly 1 job = $k\frac{1}{k}(1-\frac{1}{k})^{k-1} = (1-\frac{1}{k})^{k-1}$ (k possible choice of the job, $\frac{1}{k}$ that job assigned $(1-\frac{1}{k})^{k-1}$ not assigned)

Expected number of machines is assigned exactly 1 job =

$$\begin{aligned} &k(1-\frac{1}{k})^{k-1} \\ =&k(1-\frac{1}{k})^{k}\frac{k}{k-1} \end{aligned}$$

As $k \to \infty$, it becomes $\frac{k}{e}$

Then the rest of the machines is expected to have 2 jobs, which is $k - \frac{2k}{e}$.

Therefore, the total number of rejected jobs = total jobs - expected number of single job machines - 2*expected number of double job machines =

$$\begin{aligned} R_2(k) =&k - \frac{k}{e} - 2(k - \frac{2k}{e}) \\ =&\frac{k}{e}(e - 1 - 2e + 4) \\ =&\frac{k}{e}(3 - e) \end{aligned}$$

Then the fraction is

$$R_2(k)/k = \frac{3}{e} - 1$$