CS 120: Intro to Algorithms and their Limitations

Problem Set 7

Due: November 15, 2023 11:59pm **Denny Cao**

Collaborators:
No. of late days used on previous psets: 2
No. of late days used after including this pset: 3

§1 Another Coloring Algorithm

(a) In `ps7.py`.

(b)

Algorithm	Exhaustive	ISET BFS	SAT Color
# Solvable Ring Instances	0	1	6
# Solvable Cluster Instances	9	15	18
# Solvable Hard Graphs	0	0	3

We see that `SAT Color` and `ISET BFS` are relatively faster than `Exhaustive`, as all scenarios in which `Exhaustive` returned a solution, the other two algorithms did as well, but `Exhaustive` did not return when one or both of the others did for some scenarios. We know that `SAT Color` is relatively faster than `ISEF BFS` for the same reason.

This aligns with the theoretical runtimes for the algorithms, as `Exhaustive-Search 3-Coloring` takes time $O(m3^n)$, as there are 3^n ways to pick a color for each of the n vertices with m edges which must be checked to see if there are conflicting colors, whereas `ISEF-BFS-Coloring` takes time $O(1.89^n)$ from the SRE, and a reduction to `SAT` from k-coloring would take $O(n + km)$, and thus for 3-coloring, would take $O(n + 3m)$.

(c)

§2 2SAT

Although viewing all clauses together as a CNF creates a paradox, each clause separately makes sense. We will explain for 3 clauses:

- $\neg D \vee \neg A \equiv D \to \neg A$: This is the statement that, if Alice landed the time machine at her grandparents' first date and ruined it, then Alice's grandparents do not get married. We assume that, if a first date goes bad, then they will not pursue a relationship.
- $D \vee A \equiv \neg D \to A$: This is the statement that, if Alice does not land the time machine at her grandparents' first date and ruined it, then Alice's grandparents get married. This is reasonable, as if Alice does not interfere with the date, it goes well and from our assumption, they pursue a relationship.

- $C \vee \neg D \equiv \neg C \to \neg D$: This is the statement that, if Alice does not create a time machine in order to travel to the past, then Alice does not land the time machine at her grandparents' first date and ruin it. This makes sense, as Alice cannot land the time machine without creating a time machine, and if she does create a time machine, then (only viewing this clause), Alice could've landed anywhere.

The implication graph of the CNF is below:

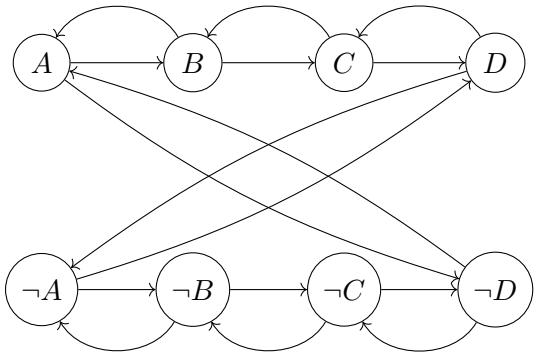

Figure 1: Implication Graph of the CNF Formula

The CNF is unsatisfiable, as there exists a "bad cycle", or path from some literal x to $\neg x$ and $\neg x$ to x. From our graph, there exists the path $\{A, B, C, D, \neg A, \neg B, \neg C, \neg D, A\}$. We see that the path goes from A to $\neg A$, and $\neg A$ to A, and therefore the CNF is unsatisfiable.

§3 Reductions to SAT

(a) *Proof.* We first describe the algorithm:

- Preprocess: We construct a CNF ϕ. In ϕ, we construct variables $x_{i,j}$, where $i \in [n]$ and $j \in [k]$ to represent if student s_i is the j-th person in the team of k.

  We will form clauses:

  - A_ℓ by iterating over S, and for each student $s_i \in S$, iterating over $K(s_i)$ and for each $\ell \in K(s_i)$, adding the clause $\bigvee_{j \in [k]} x_{i,j}$ to A_ℓ.
  - $B_{i,a,b} = (\neg x_{i,a} \vee \neg x_{i,b})$ for $i \in [n]$, for $a, b \in [k]$ where $a \neq b$.
  - $C_j = \bigvee_{i \in [n]} x_{i,j}$ for $j \in [k]$.
  - $D_{a,b,j} = (\neg x_{a,j} \vee \neg x_{b,j})$ for $a, b \in [n]$ where $a \neq b$, for $j \in [k]$.
- We make an oracle call to `SAT` on ϕ and get an assignment α.
- Postprocess: If $\alpha = \perp$, then return $\perp$. Otherwise, we use the assignment α to form the team $T = \{i \in [n] : \alpha_{i,j} = 1\}$.

We now want to prove our reduction algorithm:

1. has the desired amount of variables and clauses for ϕ and
2. is correct.

- We construct variables $x_{i,j}$ to represent if s_i is the j-th person on the team. As there are n students and k people on the team, it follows that ϕ has kn variables. We will now find the amount of clauses:
  - As we form a clause A_ℓ for each language, and there are m languages, there are m clauses.
  - As we form a clause $B_{i,a,b}$ for each pair of unique slots on the team for each student, there are $O(k^2)$ clauses for each student, and thus there are $O(k^2 n)$ clauses.
  - As we form a clause C_j for each slot on the team, and there are k slots on the team, there are k clauses.
  - As we form a clause $D_{a,b,j}$ for each unique pair of students for each slot on the team, there are $O(n^2)$ clauses for each slot on the team, and thus there are $O(n^2 k)$ clauses.

  We can sum this to obtain $m + O(k^2 n) + k + O(n^2 k)$ clauses. As $n > k$, we can simplify further to obtain $m + O(kn^2)$ clauses.
- We will prove the correctness of our reduction algorithm by analyzing the assignment α to ϕ. $\alpha \neq \perp$ if all clauses are satisfied. We will observe what each clause entails.
  - The clauses $A_\ell, \ell \in [m]$ ensure that, for each language, there exists one student on the team that knows it. If $\exists A_\ell$ such that A_ℓ is unsatisfied, then a team that knows all m languages cannot be formed, as there exists a language that no student knows.
  - The clauses $B_{i,a,b}, i \in [n], a, b \in [k], a \neq b$ ensure that each student is in at most 1 slot of the team. If $\exists B_{i,a,b}$ such that $B_{i,a,b}$ is unsatisfied, then there exists slots a, b such that s_i is in both. We can observe this by rewriting the clause as $\neg(x_{i,a} \wedge x_{i,b})$, which will only be false when $x_{i,a} = x_{i,b} = 1$, and thus the team of unique members cannot be formed.
  - The clauses $C_j, j \in [k]$ ensure that each slot on the team has a member. If $\exists C_j$ such that C_j is unsatisfied, then there does not exist a student s_i in slot j of the team, and thus the team of size k can be formed.
  - The clauses $D_{a,b,j}, a, b \in [n], a \neq b, j \in [k]$ ensure that each slot on the team has only 1 member. If $\exists D_{a,b,j}$ such that $D_{a,b,j}$ is unsatisfied, then there exists a slot j on the team with two members, which is not a valid result. We can observe this by rewriting the clause as $\neg(x_{a,j} \wedge x_{b,j})$, which will only be false when $x_{a,j} = x_{b,j} = 1$.

  The clauses ensure that the team consists of k unique members where, for every language $\ell \in L$, there exists a member t such that $\ell \in K(t)$. If $\alpha \neq \perp$, then then there exists a combination of k students such that this is true. If $\alpha = \perp$, then there does not. When $\alpha \neq \perp$, then for each $j \in [k]$, only one $x_{i,j} = 1$, which represents placing student s_i in the j-th slot of the team. In our postprocessing, we return a set of each student for each j, resulting in k students that satisfy the clauses above, and thus when the algorithm returns, the output is correct.

We will now analyze the runtime. For preprocessing, initializing kn variables takes time $O(kn)$. For each clause:

- Iterating over S takes time $O(n)$. For each student s_i, we iterate over each $\ell \in K(s_i)$, and in the worst case, each student knows m languages, thus taking $O(m)$. We then add k variables associated for each student to A_ℓ, thus taking $O(mk)$ for each student. Thus, in total, these clauses take $O(mkn)$ to build.
- Creating $O(k^2)$ clauses for each student takes time $O(k^2 n)$ to build.
- Creating a clause of all students will take $O(k)$ for each slot on the team. As there are n slots, it will take time $O(kn)$ to build.
- Creating $O(n^2)$ clauses for each slot on the team takes time $O(n^2 k)$ to build.

We combine the runtimes to get $O(kn) + O(mkn) + O(k^2 n) + O(kn) + O(n^2 k) = O(mkn + n^2 k)$, which is the time it takes to build ϕ. We make a single oracle call to `SAT` which will take $O(1)$. If the assignment α of ϕ is not $\perp$, then we iterate through each $\alpha_{i,j}$, and if $\alpha_{i,j} = 1$, then we add it to the output T. Iterating through α takes $O(kn)$, as there are kn variables, and for each kn, we perform a basic operation $O(1)$, and thus the postprocessing takes $O(kn)$.

Thus, the total runtime of the algorithm is $O(mkn + n^2 k) + O(1) + O(kn) = O(mkn + n^2 k)$. □