Grade: Minor revisions

Homework 2

September 14, 2017

Definitions & Notation

Definition 1. *A formula F is in r-*CNF *(conjunctive normal form) if for some positive integer n, $F = C_1 \wedge C_2 \wedge \cdots \wedge C_n$, where each C_i is a* clause *which consists of exactly r literals and/or their negations* OR*'ed together, i.e. $C_i = x_1 \vee \neg x_2 \vee \cdots \vee x_r$.*

Min-2-SAT
Input: A 2-CNF formula ϕ and a non-negative integer k
Parameter: k
Question: Is there an assignment for ϕ that satisfies at most k clauses?

Proofs Required

Theorem. *(Exercise 3.13 in Cygan et al.)* Min-2-SAT *can be solved in time $2^k n^{O(1)}$ using a bounded search tree algorithm.*

Proof. Given an instance of Min-2-SAT, (ϕ, k), where $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$ and each clause in ϕ is of four forms. Namely, $C_i = v_s \vee v_p$ or $C_i = v_s \vee \neg v_p$ or $C_i = \neg v_s \vee v_p$ or $C_i = \neg v_s \vee \neg v_p$, $i \in [1, \cdots, n], s, p \in [1, \cdots, r]$, where n is the number of clauses and r is the number of literals and negations. We first describe the main strategy of our algorithm. Since we want an assignment for the instance ϕ that satisfies at most k clauses, our strategy is trying to make the number of true clauses in ϕ as small as possible. Namely, we want to assign false clauses in ϕ as many as possible. Therefore, our method includes two steps.

In the first step, we search literals v_s's that appear as the same form in clauses of ϕ and mark v_s, $s \in [1, \cdots, r]$ to be false (i.e., for every literal v_s, if there doesn't exist negation $\neg v_s$ in any clause of ϕ but there exists v_s in some clauses, then we mark these v_s's to be "false"). Furthermore, we search negations $\neg v_s$'s that appear as the same for in clauses of ϕ and mark v_s, $s \in [1, \cdots, r]$ to be true (i.e., for every negation $\neg v_s$, if there doesn't exist literal v_s in any clause of ϕ but there exists $\neg v_s$ in some clauses, then we mark these v_ss to be "true").

Run time?

Safeness?

In the second step, we ignore those clauses in which both of two literals or negations have been marked by "true" or "false" in the first step, since this clause has been "false". Therefore, we focus on those clauses in which no literal or negation has been marked and those clauses in which one of two literals (or negations) has been remarked. Among these clauses, we design a bounded search tree algorithm for literals v_s's that have not been remarked. We assume that the literal v_s appears in one clause C_m and its negation appears in another different clause C_n. Therefore the intuition to design the bounded search tree algorithm is to make one of C_m and C_n to be "false" by setting v_s in C_m to be "false" or setting $\neg v_s$ in C_n to be "false" (namely setting v_s to be "true").

We now describe the recursive branching algorithm more formally. In the branch where $v \in C_m$ and its negation $\neg v \in C_n$, we recursively branch on two cases by considering either marking the literal $v \in C_m$ to be "false" and reduce the parameter k by 1 at least. In the second branch we set $\neg v \in C_n$ to be "false" (namely, mark v to be "true") and reduce the parameter k by 1. Note that if we set $v \in C_m$ to be "false", then C_m is "false", and if we set $\neg v \in C_n$ to be "false", then C_n is "false".

Finally we analyze the running time of our method. The first step can be done in time $n^{O(1)}$ because we can just search those literals or negations in ϕ, in time $n^{O(1)}$. The running time of the bounded search tree is bounded by

k only decreases when clause set to true

$$(\text{the number of leaves in the search tree}) \times (\text{time taken at each node})$$

Clearly the time taken at each node is bounded by $n^{O(1)}$ because we can still search those literals or negations v's that v is in C_m and its negation $\neg v \in C_n$. The bounded search tree runs with parameter k. Below its root, it has two same subtrees and each subtree for the same bounded search tree algorithm runs with parameter $k-1$. This means that we can define a recursive formula

$$T(i) = \begin{cases} T(i-1) + T(i-1) & \text{if } i \geq 2 \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

Then the number of leaves of the bounded search tree is bounded by $T(k)$. We prove that $T(k)$ is bounded by 2^k as follows.

$$T(k) = 2T(k-1) = 2^2T(k-2) = \cdots = 2^kT(1) = 2^k$$

Therefore the running time of our second step is $2^k \cdot n^{O(1)}$. Overall, Min-2-SAT can be solved in time $2^k \cdot n^{O(1)}$ using bounded search tree algorithm.

□

The algorithm is correct, but the branching argument is not. Set X to True, then ≥1 clause is satisfied. Set X to False, then again ≥1 clause satisfied. Either way, ≥1 clauses satisfied and k goes down.