Challenge Problem **#25.60 (b2) (c)**

Solution by **Aly Soliman** `alysoliman@uchicago.edu`

Contents

1. Question

This is an answer to challenge problems 25.60 (b2) and (c) given in Honors Algorithms, Winter 2021. Given an instance of the MAX3SAT problem, we can find an assignment that satisfies at least a 7/8 fraction of the clauses in polynomial time using the method of conditional expectations. Alternatively, we can use local-search as an approximation algorithm for the MAX3SAT problem. A t-local maximum is an assignment of the boolean variables so that flipping upto t of the variables in the assignment doesn't increase the number of satisfied clauses. Babai asked the following two questions in Algorithms (Prob 25.60 (b2) (c)).

(1) True or False: A 2-local maximum satisfies at least a 6/7 fraction of the clauses.
(2) True or False: A 3-local maximum satisfies at least a 7/8 fraction of the clauses.

The answer is false to both of those questions. We describe an algorithm that searches through the space of MAX3SAT instances to find counterexamples (we wrote a C++ implementation of the algorithm that can be found here). The algorithm found instances where the zero assignment is a 2-local maximum and satisfies a 4/5 fraction of the clauses - here's one such example.

$$\begin{gathered} x_0 \vee x_1 \vee x_2 \\ \neg x_0 \vee x_1 \vee x_3 \\ \neg x_1 \vee x_2 \vee x_3 \\ x_0 \vee \neg x_2 \vee x_3 \\ \neg x_3 \vee x_4 \vee x_5 \end{gathered} \tag{1}$$

Furthermore, the algorithm found an instance where the zero assignment is a 3-local maximum and satisfies a 6/7 fraction of the clauses.

$$\begin{aligned}
&x_0 \vee x_1 \vee x_2\\
&\neg x_0 \vee x_1 \vee x_2\\
&x_0 \vee \neg x_1 \vee x_2\\
&x_0 \vee x_1 \vee \neg x_2\\
&\neg x_0 \vee x_3 \vee x_4\\
&\neg x_1 \vee x_5 \vee x_6\\
&\neg x_2 \vee x_7 \vee x_8
\end{aligned} \tag{2}$$

2. The case for 1-local search

For the sake of completeness, we revisit a bonus problem from algorithms and show that a 1-local maximum satisfies at least a 3/4 fraction of the clauses. Suppose we have a list $\mathcal{C}$ of m 3-disjunctive clauses with a 1-local maximum and suppose this maximum is the zero assignment of the boolean variables without loss of generality. Then the number of satisfied clauses (call it n) is the number of clauses with at least one negated literal. We want to show that $n \geq 3m/4$. If $n < m$ then there's some clause C with no negated literals. Suppose $C = x_1 \vee x_2 \vee x_3$ without loss of generality. Flipping x_1 to be true doesn't increase the number of satisfied clauses, and so there must exist some clause C_1 that was satisfied at the zero assignment but became unsatisfied upon flipping x_1. Hence C_1 must contain the literal $\neg x_1$ and no other negated literals. There exist similarly defined clauses C_2 and C_3. Then the list $H = \{C, C_1, C_2, C_3\}$ has 3 satisfied clauses at the zero assignment. If a variable other than x_1, x_2, or x_3 is flipped then the state of clauses in H remains the same (C_i remains satisfied for $i = 1, 2, 3$ and C remains unsatisfied). Otherwise, if variable x_i is flipped for $i = 1, 2, 3$ then C becomes satisfied and C_i becomes unsatisfied (the state of the other two clauses remains the same). Hence, H will always have 3 satisfied clauses under any assignment of variables that's one flip away from the zero assignment. But this means that $\mathcal{C} \setminus H$ still has the zero assignment as a 1-local max. We may inductively suppose $\mathcal{C} \setminus H$ has a 3/4 fraction of it clauses satisfied (that's $3(m-4)/4$) and so $n \geq 1 + 3(m-4)/4 = 3m/4$. The base case for $m = 4$ has already been treated since we've shown a list like H must exist when the zero assignment is a 1-local max, but then our list is exactly H.

Let $\mathcal{C}$ be a list of clauses where the zero assignment is a 1-local maximum and let $\mathcal{C}_0$ be the list of clauses with no negated literals and $\mathcal{C}_1$ be the list of clauses with some negated literal. The key idea behind the proof we gave is that if we pick a clause $C \in \mathcal{C}_0$ then every variable in C has a unique associated clause in $\mathcal{C}_1$. Thus, we have a 1 to $\binom{3}{1} = 3$ map from $\mathcal{C}_0$ and $\mathcal{C}_1$, and so $|\mathcal{C}_1| \geq 3|\mathcal{C}_0|$ (hence $|\mathcal{C}_1| \geq (3/4)|\mathcal{C}|$).

If it was the case that, assuming the zero assignment is a 2-local maximum, every variable and every *pair* of variables in a clause $C \in \mathcal{C}_0$ has a unique associated clause in $\mathcal{C}_1$ then we would have $|\mathcal{C}_1| \geq \left(\binom{3}{1} + \binom{3}{2}\right)|\mathcal{C}_0| = 6|\mathcal{C}_0|$ (hence $|\mathcal{C}_1 \geq (6/7)|\mathcal{C}|$). Similarly, if also every triple of variables in $C \in \mathcal{C}_0$ had a unique associated clause in $\mathcal{C}_1$ then we would have $|\mathcal{C}_1| \geq \left(\binom{3}{1} + \binom{3}{2} + \binom{3}{3}\right)|\mathcal{C}_0| = 7|\mathcal{C}_0|$ (hence $|\mathcal{C}_1 \geq (7/8)|\mathcal{C}|$).

Another key idea in the above inductive proof is that we picked a clause with no negated literals and we constructed a “minimal” sub-instance of MAX3SAT around that clause that allowed us to reduce the problem inductively. This inspired the search procedure described next.

3. The search algorithm

An *instance* of MAX3SAT is a pair $(\mathcal{V}, \mathcal{C})$, where $\mathcal{C}$ is the list of clauses and $\mathcal{V}$ is the set of variables used in the clauses. For instances $I_1 = (\mathcal{V}_1, \mathcal{C}_1)$ and $I_2 = (\mathcal{V}_2, \mathcal{C}_2)$, we say $I_1 \subset I_2$ if $\mathcal{C}_1 \subset \mathcal{C}_2$ (this also implies $\mathcal{V}_1 \subset \mathcal{V}_2$). Let us say a MAX3SAT instance is a t-instance if the zero assignment is a t-local maximum and let us call it a *minimal* t-instance if it is a t-instance and removing any clause from the instance will cause the zero assignment to not be a t-local maximum anymore.

Our procedure will take as input a MAX3SAT instance I_0 and a positive integer t, and will search for minimal t-instances I that contain I_0. Put loosely, we will perform a BFS search through a tree where the nodes represent MAX3SAT instances and such that the leaves represent minimal t-local instances, and if a node I is not a t-local instance then its children are instances containing I that are “closer” to being t-local instances (the procedure constructing the children is described next).

3.1. **Children Procedure.** Before describing the children procedure, we describe a couple of sub-procedures first. First, let FRESH be a procedure that takes as input a set of variables $\mathcal{V}$ and a positive integer k, and returns as output a k-set of fresh variables (i.e non of which is in $\mathcal{V}$). Furthermore, for a 3-disjunctive clause C, let $\mathcal{V}(C)$ denote the set of variables in C (ex. $\mathcal{V}(x_1 \vee \neg x_2 \vee x_3) = \{x_1, x_2, x_3\}$). Next, consider the following procedure OBSTRUCTIONS. It takes as input a MAX3SAT instance $I = (\mathcal{V}, \mathcal{C})$ and a positive integer t, and returns as output a set $\mathcal{O} \subset \mathcal{V}$ of variables so that $|\mathcal{O}| \leq t$ and if we start at the zero assignment and flip all the variables in $\mathcal{O}$, we get a higher number of satisfied clauses in I than under the zero assignment. The set $\mathcal{O}$ can be thought of as a set of "obstructions" to I being a t-instance. No such set $\mathcal{O}$ exists if, and only if, I is a t-instance, in which case the procedure returns an empty set.

```
OBSTRUCTIONS
INPUT: $I = (\mathcal{V}, \mathcal{C})$, $t$
OUTPUT: $\mathcal{O}$ (: set of obstructing variables :)
$v_0$ := Number of satisfied clauses in $I$ at the zero assignment
**for** $k = 1$ **to** $t$ **do**
    **for** Every $k$-subset $\mathcal{O} \subset \mathcal{V}$ **do**
        $v$ := Number of satisfies clauses at the zero
        assignment but with all variables in $\mathcal{O}$ flipped.
        **if** $v > v_0$ **then**
            **return** $\mathcal{O}$
        **end**
    **end(for)**
**end(for)**
**return** $\emptyset$ (: If $I$ is a $t$-instance then return the empty set :)
```

Define the procedure NEG1 that takes as input an instance $I = (\mathcal{V}, \mathcal{C})$, a set of obstructions $\mathcal{O} \subset \mathcal{V}$ and a variable $v \in \mathcal{O}$, and produces as output all the clauses of the form $\neg v \vee a \vee b$ where a and b are literals with no negations, and their variables are *not* in $\mathcal{O}$. The variables in a and b might both be from $\mathcal{V}$ or at least one of them might be fresh.

```
NEG1
INPUT: $I = (\mathcal{V}, \mathcal{C})$, $\mathcal{O} \subset \mathcal{V}$, $v \in \mathcal{O}$
OUTPUT: C (: List of clauses of the form $\neg v \vee a \vee b$ :)
C := $\emptyset$
$f_1, f_2$ = FRESH($\mathcal{V}$, 2)
(: Add all clauses of the form $\neg v \vee a \vee b$ where $a, b \in \mathcal{V} \setminus \mathcal{O}$ :)
**for** $\{a, b\} \subset \mathcal{V} \setminus \mathcal{O}$ **do**
    $C := \neg v \vee a \vee b$
    **if** $C \notin \mathcal{C}$ **then** (: Only add clauses that don't already exist :)
        **append**(C, $C$)
    **end**
**end(for)**
(: Add all the clauses where $a \in \mathcal{V} \setminus \mathcal{O}$ and $b$ is fresh :)
**for** $a \in \mathcal{V} \setminus \mathcal{O}$ **do**
    **append**(C, $\neg v \vee a \vee f_1$)
```

```
12   end(for)
13   append(C, $\neg v \lor f_1 \lor f_2$) (: The clause where $a$ and $b$ are fresh :)
14   return C
```

The following procedures, NEG2 and NEG3, are similar NEG1 but NEG2 accepts to two variables $v_1, v_2 \in \mathcal{O}$ and produces clauses of the form $\neg v_1 \lor \neg v_2 \lor a$ (i.e with two negated literals instead of 1), and NEG3 accepts three variables $v_1, v_2, v_3 \in \mathcal{O}$ and returns $\neg v_1 \lor \neg v_2 \lor \neg v_3$.

```
NEG2
INPUT: $I = (\mathcal{V}, \mathcal{C})$, $\mathcal{O} \subset \mathcal{V}$, $v_1, v_2 \in \mathcal{O}$
OUTPUT: C (: List of clauses of the form $\neg v_1 \lor \neg v_2 \lor a$ :)
00   C := $\emptyset$
01   $f$ = FRESH($\mathcal{V}$, 1)
02   for $a \in \mathcal{V} \setminus \mathcal{O}$ do
03       $C := \neg v_1 \lor \neg v_2 \lor a$
04       if  $C \notin \mathcal{C}$ then
05           append(C, $C$)
06       end
07   end(for)
08   append(C, $\neg v_1 \lor \neg v_2 \lor f$)
09   return C
```

```
NEG3
INPUT: $I = (\mathcal{V}, \mathcal{C})$, $\mathcal{O} \subset \mathcal{V}$, $v_1, v_2, v_3 \in \mathcal{O}$
OUTPUT: C (: List containing the clause $\neg v_1 \lor \neg v_2 \lor \neg v_3$ :)
00   C := $\emptyset$
01   $C := \neg v_1 \lor \neg v_2 \lor \neg v_3$
02   if  $C \notin \mathcal{C}$ then
03       append(C, $C$)
04   end
05   return C
```

We can now describe the CHILDREN procedure. It takes as input a MAX3SAT instance $I = (\mathcal{V}, \mathcal{C})$, and an integer t, and produces as output a list of MAX3SAT instances, each of which are "closer" to being a t-instance - if I is already a t-instance then the output list will be empty. The following is a description of the procedure meant to be read along side the pseudocode in the next page.

First we run the procedure OBSTRUCTIONS on I and t to obtain a set $\mathcal{O}$ of variables obstructing I from being a t-instance. If $\mathcal{O}$ is empty then the we terminate and return an empty list (lines 02-04). Otherwise, if all the variables in $\mathcal{O}$ are flipped then we obtain a higher number of satisfied clauses in I than under the zero assignment. To balance this out, any t-instance containing I must contain *at least one* clause C that was initially satisfied at the zero assignment but becomes unsatisfied when all the variables in $\mathcal{O}$ are flipped. This means that $\mathcal{V}(C) \cap \mathcal{O} \neq \emptyset$ and all the literals in C with variables in $\mathcal{O}$ must be negated while the literals with variables not in $\mathcal{O}$ are not negated. We will generate all the possible candidates for C and, for each, create a child instance $(\mathcal{V} \cup \mathcal{V}(C), \mathcal{C} \cup \{C\})$ (a copy of I with C added to the list of clauses). Note that $1 \leq |\mathcal{V}(C) \cap \mathcal{O}| \leq 3$. All the candidates for C such that $|\mathcal{V}(C) \cap \mathcal{O}| = 1$ are generated by NEG1 (lines 05-09) since these are the clause of the form $\neg v \vee a \vee b$ for some $v \in \mathcal{O}$ and $a, b \notin \mathcal{O}$. Similarly, all candidates for C such that $|\mathcal{V}(C) \cap \mathcal{O}| = 2$ are generated by NEG2 (lines 13-17) and all candidates such that $|\mathcal{V}(C) \cap \mathcal{O}| = 3$ are generated by NEG3 (lines 21-25). This completes the procedure.

```
CHILDREN
INPUT: $I = (\mathcal{V}, \mathcal{C})$, $t$
OUTPUT: C (: List of instance closer than $I$ to being $t$-instances :)
00  C := $\emptyset$
01  $\mathcal{O}$ := OBSTRUCTIONS($I, t$)
02  **if** $\mathcal{O} = \emptyset$ **then**
03      **return** C
04  **end**
05  **for** $v \in \mathcal{O}$ **do**
06      **for** $C \in$ NEG1($I, \mathcal{O}, v$) **do**
07          **append**(C,$(\mathcal{V} \cup \mathcal{V}(C), \mathcal{C} \cup \{C\})$
08      **end(for)**
09  **end(for)**
10  **if** $|\mathcal{O}| < 2$ **then**
11      **return** C
12  **end**
13  **for** $\{v_1, v_2\} \subset \mathcal{O}$ **do**
14      **for** $C \in$ NEG2($I, \mathcal{O}, v_1, v_2$) **do**
15          **append**(C, $(\mathcal{V} \cup \mathcal{V}(C), \mathcal{C} \cup \{C\})$
16      **end(for)**
17  **end(for)**
18  **if** $|\mathcal{O}| < 3$ **then**
19      **return** C
20  **end**
21  **for** $\{v_1, v_2, v_3\} \subset \mathcal{O}$ **do**
22      **for** $C \in$ NEG3($I, \mathcal{O}, v_1, v_2, v_3$) **do**
23          **append**(C, $(\mathcal{V} \cup \mathcal{V}(C), \mathcal{C} \cup \{C\})$
        end(for)
    end(for)
    return C
```

3.2. **BFS Procedure.** We can now perform the BFS procedure described below. Note that we're not keeping track of which node has been visited before since we're assuming a tree structure (i.e every instance produced by a CHILDREN procedure call at some point is taken to be a unique node that has not been reached before). The procedure produces a list of leaf nodes that have been reached (these will correspond to minimal t-instances) and the procedure will terminate once a minimum threshold of leaves has been reached.

```
BFS
INPUT: $I_0 = (\mathcal{V}, \mathcal{C})$, $t$, threshold
OUTPUT: M (: List of minimal $t$-instance containing $I$ :)
    Q := Empty queue
    M := $\emptyset$
    enqueue(Q, $I_0$)
    while Q is not empty do
        $I$ := dequeue(Q)
        C := CHILDREN($I$, $t$)
        if C is empty do
            append(M, $I$)
            if |M| $\geq$ threshold do
                return M
            end
        end
        for child $\in$ C do
            enqueue(Q,child)
        end(for)
    end(while)
    return M
```

Let I be the instance containing the single clause $x_0 \vee x_1 \vee x_2$. Executing BFS(I, 2, 1) returns counterexample (1) in Section 1. Executing BFS(I, 3, 1) returns counterexample (2).