HW 7 - CS 120

Evan Jolley

October 2022

1

1.1

All code.

1.2

	Exhaustive	ISET	SAT
Ring	0	2	6
Cluster	4	4	6
Other	0	0	2

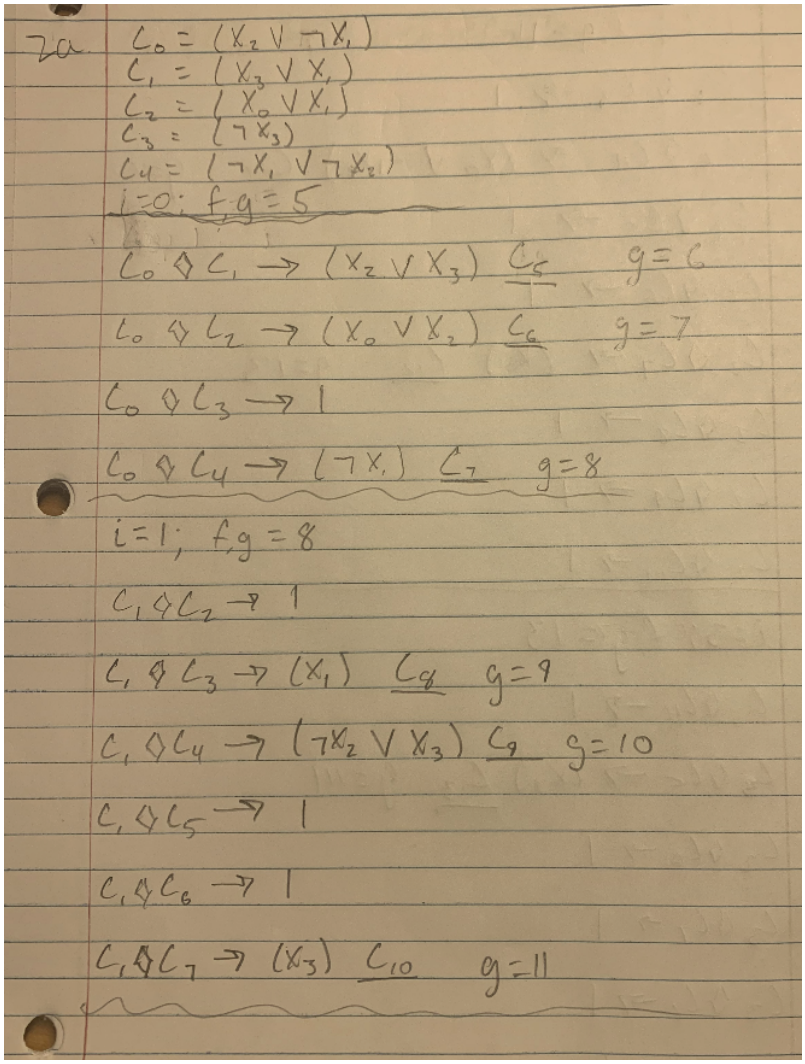
It makes sense that Exhaustive is the worst performer, as it truly brute forces the problem by trying each possible combination.

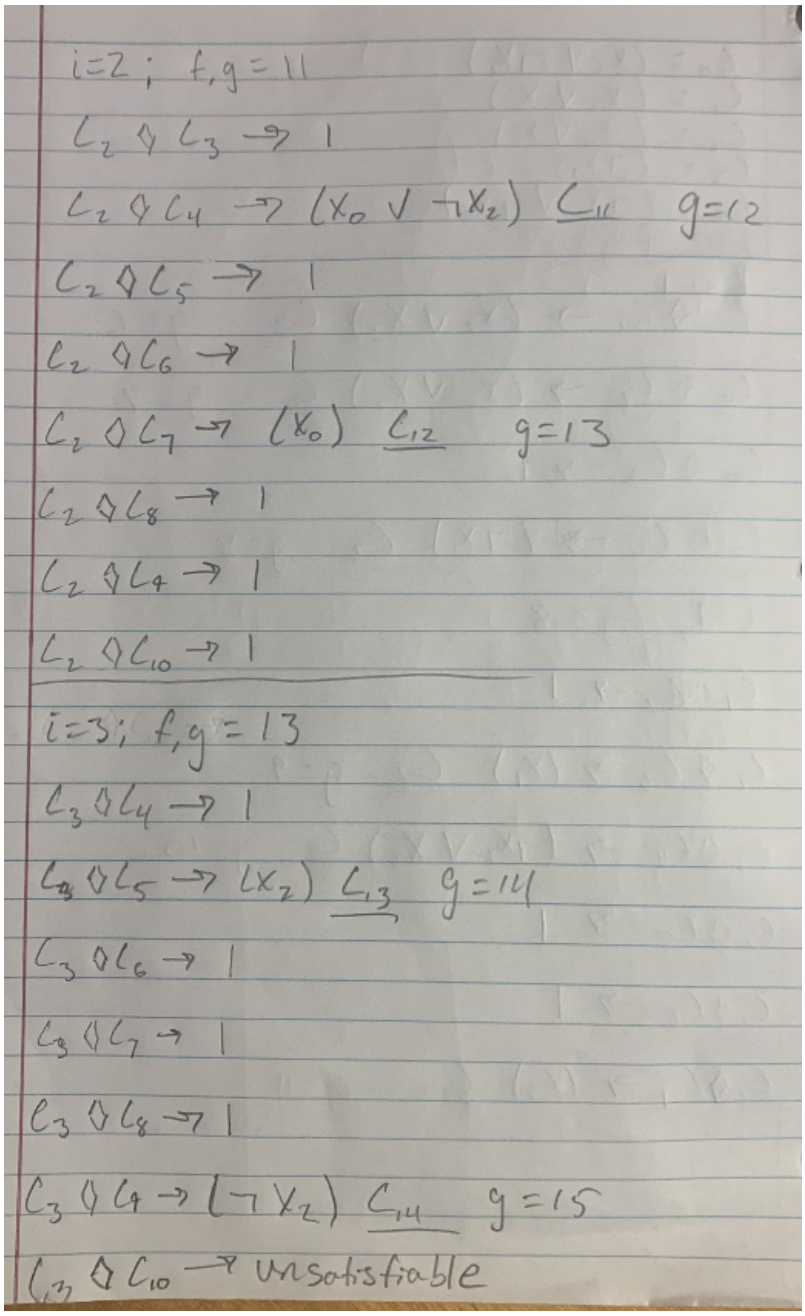
ISET and SAT are a bit mote optimized, with SAT being the more optimal of the two given that the Glucose function for the pysat package is more efficient.

Our experimental results follow and corroborate the hierarchy of theoretical runtimes of each that we have discussed in lecture and past problems sets. This won't always be true in experimentation, but it was for our tests in PS7.

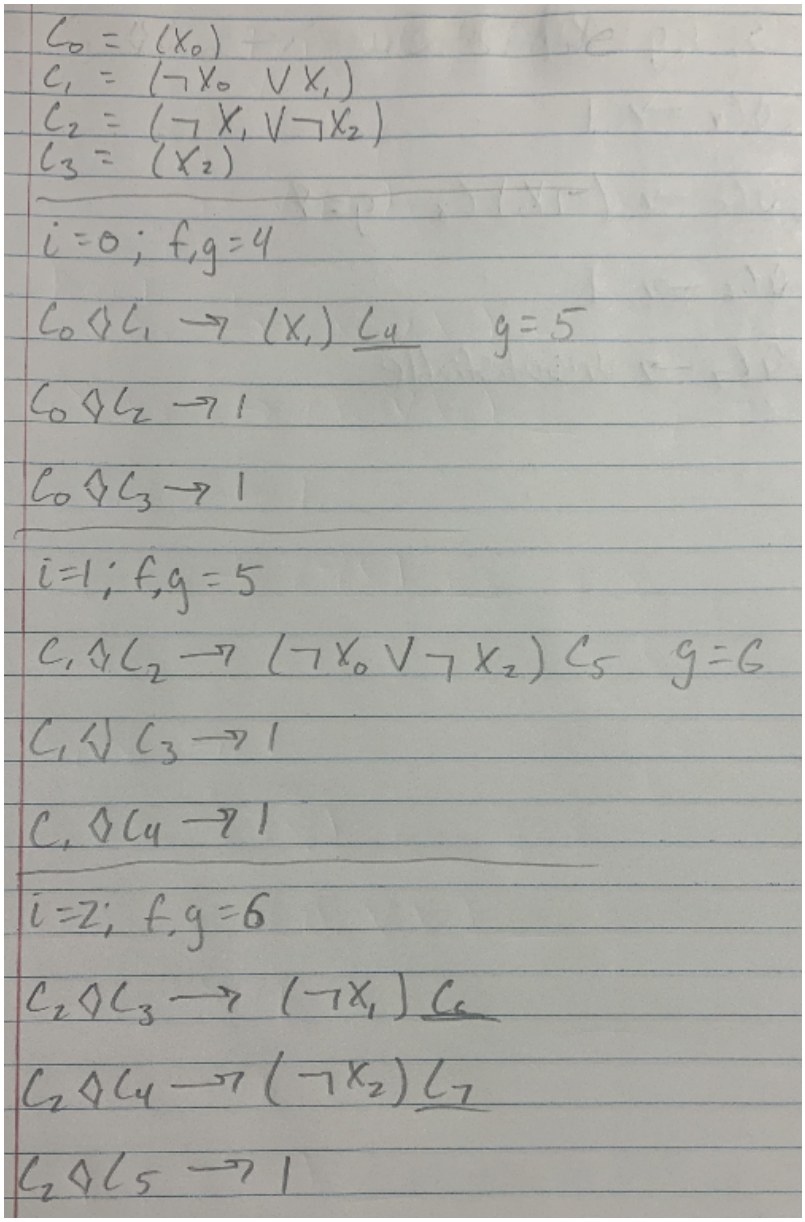
2

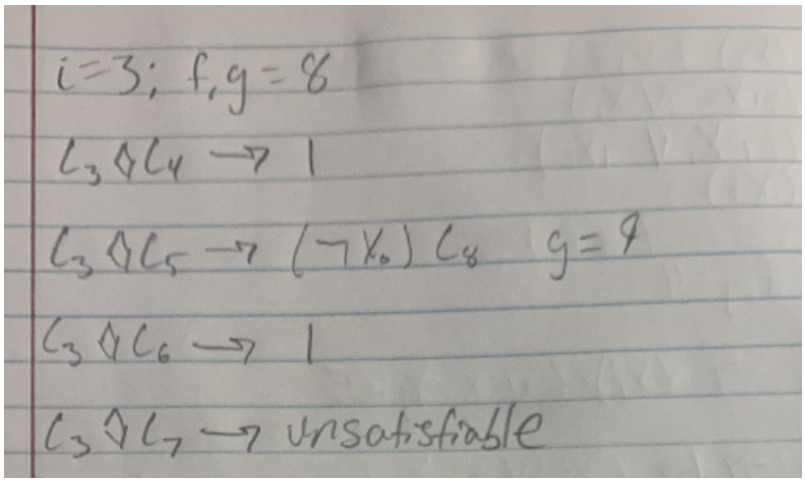
2.1


2a
$C_0 = (X_2 \vee \neg X_1)$
$C_1 = (X_3 \vee X_1)$
$C_2 = (X_0 \vee X_1)$
$C_3 = (\neg X_3)$
$C_4 = (\neg X_1 \vee \neg X_2)$
$i=0$; $f,g=5$
$C_0 \Diamond C_1 \rightarrow (X_2 \vee X_3)$ C_5 $g=6$
$C_0 \Diamond C_2 \rightarrow (X_0 \vee X_2)$ C_6 $g=7$
$C_0 \Diamond C_3 \rightarrow 1$
$C_0 \Diamond C_4 \rightarrow (\neg X_1)$ C_7 $g=8$
$i=1$; $f,g=8$
$C_1 \Diamond C_2 \rightarrow 1$
$C_1 \Diamond C_3 \rightarrow (X_1)$ C_8 $g=9$
$C_1 \Diamond C_4 \rightarrow (\neg X_2 \vee X_3)$ C_9 $g=10$
$C_1 \Diamond C_5 \rightarrow 1$
$C_1 \Diamond C_6 \rightarrow 1$
$C_1 \Diamond C_7 \rightarrow (X_3)$ C_{10} $g=11$

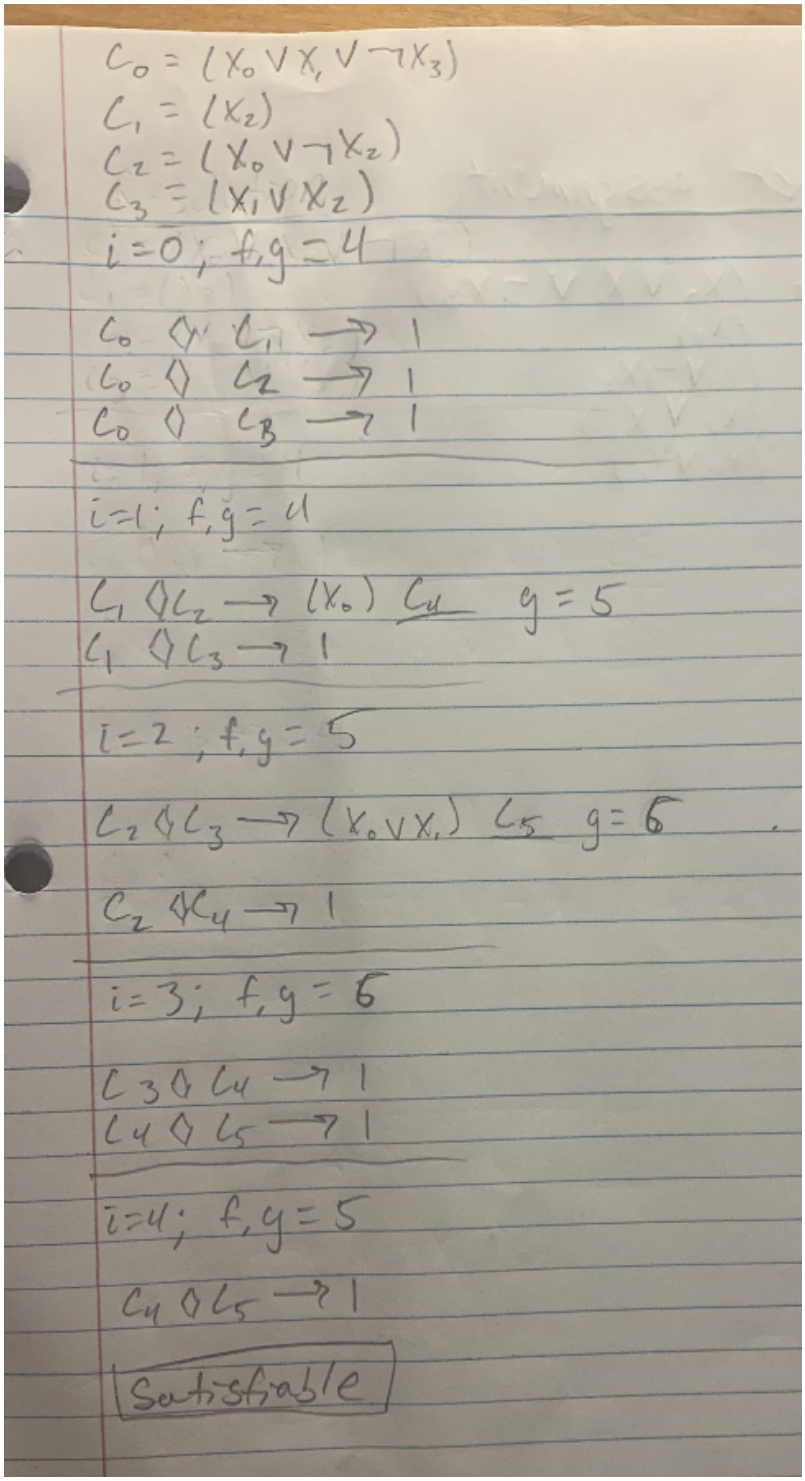

i=2; f,g=11
C2 ◊ C3 → 1
C2 ◊ C4 → (X0 ∨ ¬X2) C11 g=12
C2 ◊ C5 → 1
C2 ◊ C6 → 1
C2 ◊ C7 → (X0) C12 g=13
C2 ◊ C8 → 1
C2 ◊ C4 → 1
C2 ◊ C10 → 1
i=3; f,g=13
C3 ◊ C4 → 1
C3 ◊ C5 → (X2) C13 g=14
C3 ◊ C6 → 1
C3 ◊ C7 → 1
C3 ◊ C8 → 1
C3 ◊ C4 → (¬X2) C14 g=15
C3 ◊ C10 → unsatisfiable

2.2


$C_0 = (X_0)$
$C_1 = (\neg X_0 \vee X_1)$
$C_2 = (\neg X_1 \vee \neg X_2)$
$C_3 = (X_2)$
$i=0$; $f,g=4$
$C_0 \Diamond C_1 \rightarrow (X_1)\ C_4$ $g=5$
$C_0 \Diamond C_2 \rightarrow 1$
$C_0 \Diamond C_3 \rightarrow 1$
$i=1$; $f,g=5$
$C_1 \Diamond C_2 \rightarrow (\neg X_0 \vee \neg X_2)\ C_5$ $g=6$
$C_1 \Diamond C_3 \rightarrow 1$
$C_1 \Diamond C_4 \rightarrow 1$
$i=2$; $f,g=6$
$C_2 \Diamond C_3 \rightarrow (\neg X_1)\ C_6$
$C_2 \Diamond C_4 \rightarrow (\neg X_2)\ C_7$
$C_2 \Diamond C_5 \rightarrow 1$


i=3; f,g=8
$C_3 \Diamond C_4 \rightarrow 1$
$C_3 \Diamond C_5 \rightarrow (\neg X_6)\, C_8 \quad g=9$
$C_3 \Diamond C_6 \rightarrow 1$
$C_3 \Diamond C_7 \rightarrow$ unsatisfiable

2.3


$C_0 = (X_0 \vee X_1 \vee \neg X_3)$
$C_1 = (X_2)$
$C_2 = (X_0 \vee \neg X_2)$
$C_3 = (X_1 \vee X_2)$
$i=0$; $f,g=4$
$C_0 \Diamond C_1 \rightarrow 1$
$C_0 \Diamond C_2 \rightarrow 1$
$C_0 \Diamond C_3 \rightarrow 1$
$i=1$; $f,g=4$
$C_1 \Diamond C_2 \rightarrow (X_0)$ C_4 $g=5$
$C_1 \Diamond C_3 \rightarrow 1$
$i=2$; $f,g=5$
$C_2 \Diamond C_3 \rightarrow (X_0 \vee X_1)$ C_5 $g=6$
$C_2 \Diamond C_4 \rightarrow 1$
$i=3$; $f,g=6$
$C_3 \Diamond C_4 \rightarrow 1$
$C_4 \Diamond C_5 \rightarrow 1$
$i=4$; $f,g=5$
$C_4 \Diamond C_5 \rightarrow 1$
Satisfiable

Extract assignment

C: C_0 $(X_0 \vee X_1 \vee \neg X_3)$
C_1 (X_2)
C_2 $(X_0 \vee \neg X_2)$
C_3 $(X_1 \vee X_2)$
C_4 (X_0)
C_5 $(X_0 \vee X_1)$

(X_0) exists $\rightarrow$ $\alpha_0 = 1$

C: C_1 (X_2)
C_3 $(X_1 \vee X_2)$

(X_1) DNE $\rightarrow$ $\alpha_1 = 0$

C: C_1 (X_2)
C_3 $(X_1 \vee X_2)$

(X_2) exists $\rightarrow$ $\alpha_2 = 1$

C: $\{\}$

X_3 DNE $\rightarrow$ $\alpha_3 = 0$

C: $\{\}$

$\alpha = [1, 0, 1, 0]$

3

3.1

Reduction

Let x_{yz} where $1 \leq y \leq n$ and $1 \leq z \leq k$ be the variable representing if student y is in the zth slot on the team. That is, the variable equals 1 if the specified student is in the slot and 0 if no. There will be kn total of these variables.

There will be a few types of clauses in present in our SAT algorithm. Let's discuss each:

Each team spot must be filled
Every slot on the team must be filled, so we will create clauses to ensure this. For example, to insure that the zth slot is full, our clause would look like this:

$$((x_{0z}) \vee (x_{1z}) \vee ... \vee (x_{2z}))$$

This type of clause will exist for all k spots on the team, so k total clauses.

Multiple students can not be assigned to the same spot
No two students can occupy the same spot on the team. In other words, x_{s0} and x_{t0} can not both be true. Thus, these clauses will look like:

$$((\neg x_{s0}) \vee (\neg x_{t0}))$$

Clauses like this must be created for every possible pair of students across all k spots. Thus, there will be $\binom{k*n}{2}$ clauses.

$$k * \binom{n}{2} = \frac{k * n!}{2! * (n-2)!} = \frac{k * n * (n-1)}{2}$$

Students can not be assigned to multiple spots
Students can only be assigned to one spot on the team. In other words x_{y0} and x_{y1} can not both be true. We can create clauses that look like this to ensure this does not happen:

$$((\neg x_{y0}) \vee (\neg x_{y1}))$$

Clauses like this must be created for every possible pair of teams for every student. Thus, there will be $\binom{n*k}{2}$ clauses.

$$n * \binom{k}{2} = \frac{n * k!}{2! * (k-2)!} = \frac{n * k * (k-1)}{2}$$

All languages must be represented
We must have clauses for each language to ensure that it is represented in our group of students. We know what languages each student knows from our input, but we must manipulate this to access the information from the other direction (what students know each languages.

To accomplish this, we can create a mapping M such that $M(l_i)$ is all students that know language i. We can create this mapping by iterating over each student, and adding them to $M(l_i)$ if language i is in $K()$ for that student.
We can then create our clauses algorithmically:

1. Iterate over each of the m languages
2. With each language, iterate over all students that know the language using our mapping, adding them to a clause that guarantees that at least one of them will have a spot on the team. If there are three students who know a language (let's say students 1, 2, and 3) the clause will look something like this:

$$((x_{10}) \vee ... \vee (x_{1k-1}) \vee (x_{20}) \vee ... \vee (x_{2k-1}) \vee (x_{30}) \vee ... \vee (x_{3k-1}))$$

This clause will be satisfied if and only if one of these students is in the team, guaranteeing that the language is represented. There will be m total of these clauses as there are m total languages.

Calling a SAT solver (oracle of the reduction) will output a solution or $\perp$ depending on if there exists a solution.

Post processing is simple with this reduction. If the SAT solver returns $\perp$, we can return $\perp$ as well. If the SAT solver returns a solution, we can iterate over our variables, adding a student to the team if one of their corresponding variables is set to 1.

Correctness

If there is a way to form a team of k students knowing m languages, then our reduction must give a valid solution. If our reduction gives us a valid solution, then there must be a way to form a team of k students knowing m languages.

1st statement: We defined our clauses to all return true if and only if there is a valid solution for us to find: each spot must be filled, multiple students can not be in the same spot, students can not be assigned to multiple spots, and all languages must be represented. If there is a solution, these will all be satisfied, and SAT will give a valid solution.

2: If our SAT reduction returns a solution to this problem, we know that all of the conditions we defined through our clauses have been satisfied. The four types of clauses described above are have all reduced to true, and there is indeed a team of k students who know all m languages for our team.

Runtime

- kn total variables, $O(kn)$ time to initialize them
- "Each team spot must be filled" clause has k total with n literals each, $O(kn)$ again
- "Multiple students can not be assigned to the same spot" clause has $\frac{k*n*(n-1)}{2}$ total with 2 literals each, $O(kn^2)$ runtime
- "Students can not be assigned to multiple spots" clause has $\frac{n*k*(k-1)}{2}$ total with 2 literals each, $O(k^2n)$ runtime
- In creating our mapping we will need to iterate over all m languages and in the worst case all n students will know each languages meaning an $O(mn)$ runtime.
- "All languages must be represented" clause has m total, one for each language, with maximum kn literals each if every student knew a particular language, $O(mkn)$ runtime
- The oracle call is time $O(1)$ by definition
- Post processing takes time $O(kn)$ to iterate over all variables if there is a solution and $O(1)$ to return $\perp$ if there is not

By summing all of these runtimes and reducing to only what's needed, we achieve our desired runtime of $O(kn^2 + knm)$. Note that kn^2 is greater than k^2n, and the latter is not included in our worst case because their big Os are equivalent.