Assignment #: 1

Thejesh Venkata Arumalla and Sai Shashank Gaddam

September 16, 2022

1. **P1 - Atleast one is true**
   We need to add a new clause which is disjunction of all propositions

   $P_1 \lor P_2 \lor P_3....P_n$
   Code execution -
   I ran the below commands in ubuntu

   - Compiling: g++ p1.cpp -o p1.out
   - Enter inputs in p1Input.txt
   - Execute: ./p1.out
   - Output will be in p1Output.dimacs
   - Test it by running: picosat p1Output.dimacs

2. **P2 - Atmost one is true**
   Notations

   - Let the clauses of the given CNF ϕ be $C_1, C_2...C_m$, there are m clauses.
   - Let the propositions of the given CNF be ϕ be $P_1, P_2, ...P_n$, there are n propositions.
   - If the output is a tautology, then I am representing it as $P_1 \lor \neg P_1$
   - If the output is a unsatisfiable, then I am representing it as $P_1 \land \neg P_1$

   Logic

   - We have n iterations once for each proposition.In each iteration we generate a literal or a constant bool(i.e. 0 or 1).
     - In the i^{th} iteration we set all propositions except P_i to be false i.e. $P_j = 0 \quad \forall j != i$
       and compute a value which is either 1 or 0 or P_i or $\neg P_i$. This is similar to unit substitutions.
     - Basically the above encodes the statement where all propositions except P_i are false, P_i can either be 1 or 0, satisfying the atmost 1 constraint.
   - Finally after all iterations we do a disjunction ($\lor$) of all the outputs of the above iterations.
     So final output has only one clause.
   - The size of the final output will be $O(\text{number of distinct propositions})$ as in each iteration we atmost get one literal and add it to our clause
   - The run time is $O((\text{size of } \phi) \times \text{ number of distinct propositions})$

   Code execution -

   - Compiling: g++ p2.cpp -o p2.out

- Enter inputs in p2Input.txt
- Execute: ./p2.out
- Output will be in p2Output.dimacs

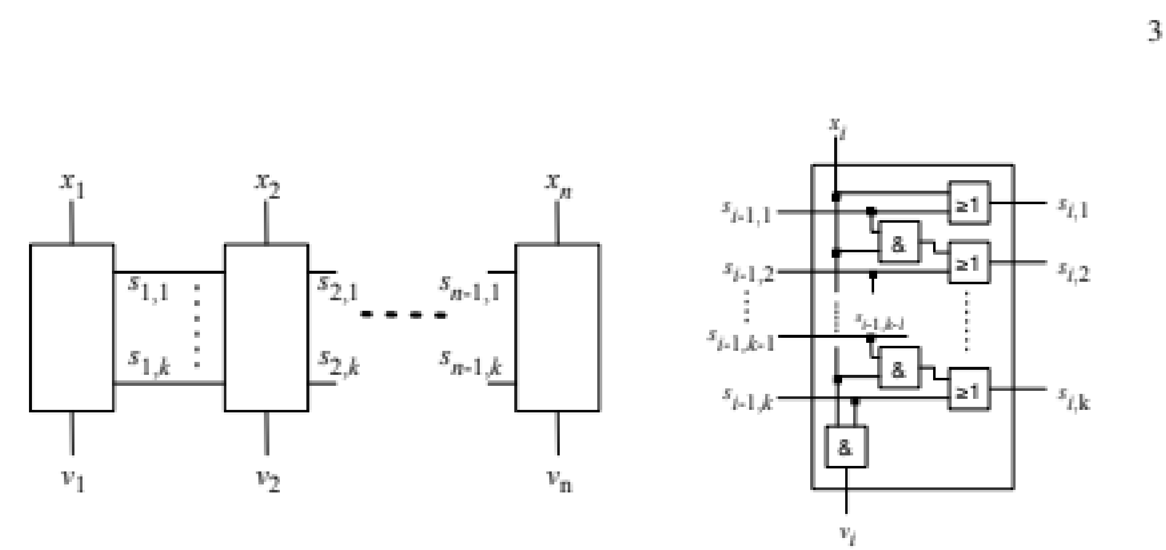
Example test cases -

- Input: $P1 \lor P2 \lor P3 \implies$ Output: $P1 \lor P2 \lor P3$
- Input: $P1 \land P2 \implies$ Output: unsatisfiable as both needs to be one
- Input: $P1 \lor \neg P1 \implies$ Output: tautology as it always true

3. **P3 - Atmost K**
   Notations used in code -

   - Let N be the number of propositions, K be the constraint.
   - $S[i][j]$ where $i \in [1, N]$ and $j \in [1, K]$ are the additional propositions we introduced in addition to the existing propositions

   Logic -
   We try to encode the logic as a circuit
   I used the the research paper by Sinz: `https://www.carstensinz.de/papers/CP-2005.pdf`
   It basically introduces new propositions and reduce the overall size of the CNF.
   In the paper the author constructs the following circuit



   The circuit is reduced to a CNF using Tseitin Transformations

$$
\begin{array}{ll}
(\neg x_1 \vee s_{1,1}) & \\
(\neg s_{1,j}) \quad \text{for } 1 < j \le k & \\
\left.\begin{array}{l}
(\neg x_i \vee s_{i,1}) \\
(\neg s_{i-1,1} \vee s_{i,1}) \\
\left.\begin{array}{l}(\neg x_i \vee \neg s_{i-1,j-1} \vee s_{i,j}) \\ (\neg s_{i-1,j} \vee s_{i,j})\end{array}\right\} \quad \text{for } 1 < j \le k \\
(\neg x_i \vee \neg s_{i-1,k})
\end{array}\right\} & \text{for } 1 < i < n \\
(\neg x_n \vee \neg s_{n-1,k}) &
\end{array}
$$

The proof is not shared by the author, I tried my best to prove it in the below pdf, Proof:

`https://drive.google.com/file/d/1wof9ZK8ozgYWhQqGJr80H5PbMe0NjQkg/view?usp=sharing`

Code execution -

- Compiling: g++ p3.cpp -o p3.out
- Enter inputs in p3Input.txt
  - The first line in the input is K
  - Then from the next line, we have CNF in dimacs format
- Execute: ./p3.out
- Output will be in p3Output.dimacs

References
- Main formula - `https://www.carstensinz.de/papers/CP-2005.pdf`
- Tseitin Transform, used to derive the formula in the above research paper : `https://people.cs.umass.edu/~marius/class/h250/lec2.pdf`

4. **P4 - Subgraph isomorphism**
   Notations used in code -

   - Let the graphs be H and G
   - Let H_n,G_n denote the number of vertices in H and G
   - Let H_m,G_m denote the number of edges in H and G.
   - Our proposition variables are $X_{i,j}$ where it gets the value 1 if there is a mapping between i^{th} node in H with j^{th} node in G.

   Logic -

   - Every vertex in H must be mapped to atleast one vertex in G.
     - For every $i \in [1..H_n]$ add a clause $(X_{i,1}, X_{i,2}, X_{i,3}.....X_{i,G_n})$
   - No two vertices in H can be mapped to same vertex in G.
     - For every two vertices $i, j \in [1..H_n]$ graph H and every vertex $k \in [1, 2..G_n]$ graph G add a clause
       i.e $X_{i,k}$ & $X_{j,k}$ can't be true $\implies \neg(X_{i,k} \ \& \ X_{j,k}) \implies \neg X_{i,k} \vee \neg X_{j,k}$

- A vertex in H can't be mapped to more than one vertex in G.
  - For every two vertices $k1, k2 \in [1..G_n]$ graph G and every vertex $i \in [1, 2..H_n]$ graph H add a clause
    i.e $X_{i,k1} \ \& \ X_{i,k2}$ can't be true $\implies \neg(X_{i,k1} \ \& \ X_{i,k2}) \implies \neg X_{i,k1} \vee \neg X_{i,k2}$
- Every edge in H should be mapped to some edge in $G \implies$ Every edge in H should **not be mapped to any non edge in G**.

Code execution -

- Compiling: g++ p4.cpp -o p4.out
- Enter graph inputs in graphH.txt for graph H and graphG.txt for graph G
- Execute: ./p4.out
- Output will be in p4Output.dimacs

References - `https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.190/Mitarbeiter/toran/fin.pdf`

5. **P5 - Graph partitioning**

Notation -

- There are N vertices and M edges
- Let there be 2 partitions P_0 and P_1
- Let $X_{i,0}$ denote the proposition where it gets the value 1 if the vertex i is in partition P_0.
- Let $X_{i,1}$ denote the proposition where it gets the value 1 if the vertex i is in partition P_1.
- We introduce new propositions, one for each edge $[Z_1, Z_2, ...Z_{|E|}]$ where Z_i is 1 if both the vertexes of that edge is accross partitions, i.e $Z_k \iff ((\neg X_{i,0} \ \& \ X_{j,0}) \vee (X_{i,0} \ \& \ \neg X_{j,0}))$

  Using sympy module in python to find the CNF of the above formula

```
>>>
>>> to_cnf( (Z >> ( ( A & Not(B) ) | ( Not(A) & B ) )) & (( ( A & Not(B) ) | ( Not(A) & B ) ) >> Z),True )
(A | B | ~Z) & (A | Z | ~B) & (B | Z | ~A) & (~A | ~B | ~Z)
```

Logic -

- A node can't be in two partitions
- A node should be present in atleast one partition
- The partitions should be equal in size, i.e $\sum_{i=1}^{N} X_{i,0} = \frac{N}{2}$
  - We can modify the atmost K solution
    * There should be atmost K true propositions **AND** There should be atleast K true propositions
    * The statement : "There should be atleast K true propositions" is equivalent to "There should be atmost (N-K) false propositions"

        * Therefore - There should be atmost K true propositions **AND** There should be atmost (N-K) false propositions

- The number of edges accross partions should be less than or equal to K
    - i.e $\sum_{i=1}^{|E|} Z_i <= K$
    - we can use the atmost K solution to solve this.

Code execution -

- Compiling: g++ p5.cpp -o p5.out
- Enter graph inputs in graph.txt
    - The first line contains the integer K
    - The next line contains N,M where N is number of nodes, it must be a multiple of 2, M is the number of edges
    - Each of the next M lines contain the edges in the form "x y"
- Execute: ./p5.out
- Output will be in p5Output.dimacs

Test Case used -
The below graph will be unsatisfiable for K=1,2,3 but satisfiable for $K > 3$