CS3230 Assignment 3 Solutions

May 3, 2020

Question 1

Solutions: There are a few solutions to this problem, we'll show one such solution. First we'll build each gadget, then show how to link them up. Then present the proofs on their correctness.

The first solution as follows:

The gadgets for each variable and clause are shown in the diagram below.

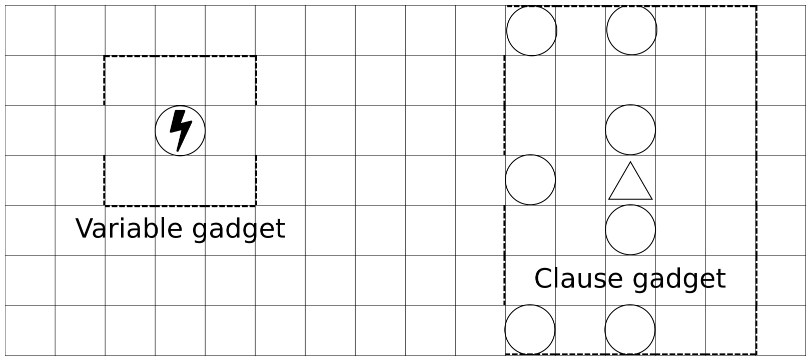

Figure 1: Variable and Clause gadgets

Note that later on for a variable gadget, powering the left broadcast node amounts to setting the variable to be TRUE, whereas powering the right amounts to setting the variable to be FALSE.

Now we'll place the variable gadgets at the top of the grid with some spare tiles in between them. We'll also place the clause gadgets on the right side of the grid, also with some spare tiles in between them. (We'll bound the amount of spare tiles needed later).

So thus far, what we have looks like this.

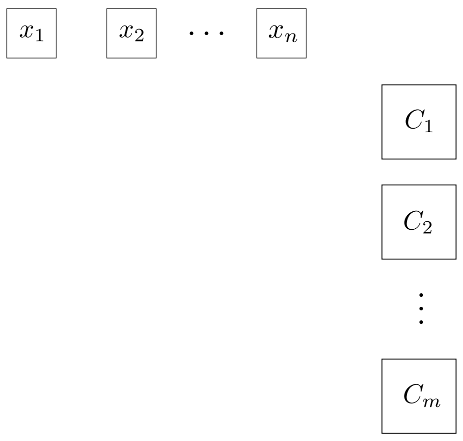

Figure 2: Placement of the variable and clause gadgets

Now to bridge the variable gadgets to the clause gadgets. First, extend from both ends of each variable gadget a long path that looks like this:

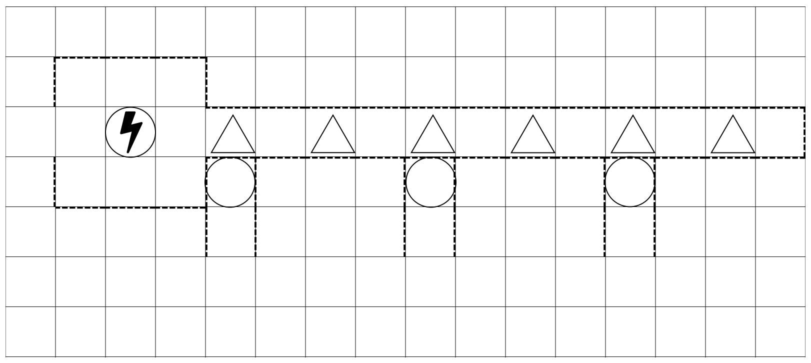

Figure 3: Example of a branch on one end

Now we need to talk about how to link the variables to the gadgets. Since we don't allow the wires to cross, we'll build one more gadget, that acts as a "bridge", shown below.

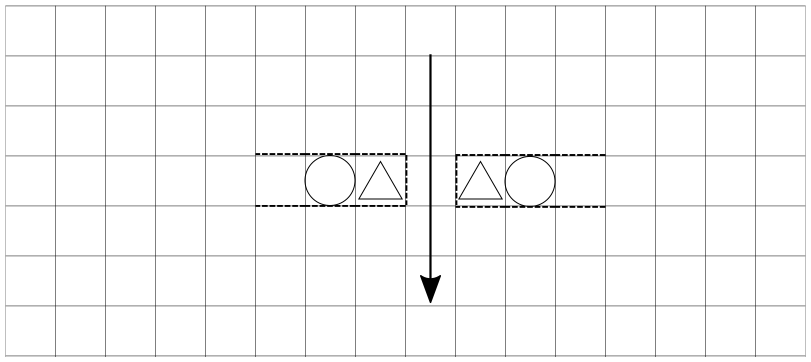

Figure 4: Bridge gadget

Below each of these paths we will place one branch for each clause that contains it. The point is that each of these branches is going to go to one clause gadget. i.e. The one that is associated with that variable. So the left branches of a variable should go to clauses that contain the variable itself, whereas the right branches of a variable should go to clauses that contain the negated version of the variable.

The general idea is that we're going to lead these branches down, then whenever we need to make a turn to its clause, we use a socket. And then sometimes notice you might need to skip over other paths. To do that, you use the bridge gadget. An example is shown below:

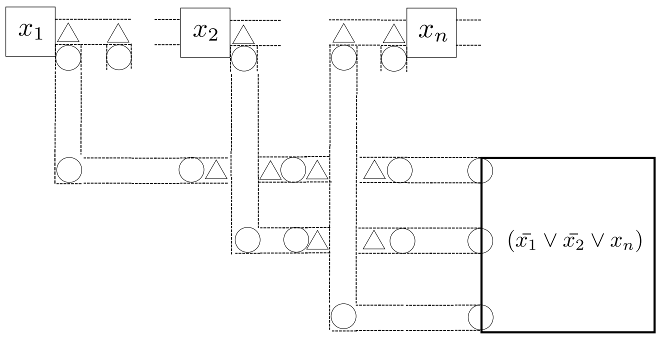

Figure 5: Example

And here's an example of what happens to $(\neg x_1 \vee \neg x_2 \vee x_n)$ when we set x_1 to true, and x_2, x_n to false.

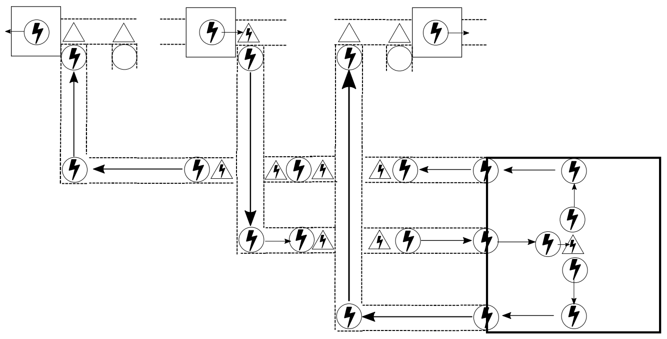

Figure 6: Solved example

Finally since in the "worst case" some variable is part of all the clauses, so we need to make sure there's enough branches for it. As well as being able to build bridges if needed. So we'll leave the space as $5m$, so each branch has about 5 tiles in between them. Also we'll put about 5 tiles in between the clause gadgets as well.

Now we need to show the usual things: (1) The reduction runs in polynomial time; (2) satisfiable instances of 3SAT get reduced to solvable instances of `PLUGIT` and (3) unsatisfiable instances of 3SAT get reduced to unsolvable instances of `PLUGIT`.

Note that based on our reduction, the number of tiles that need to be put down, blank or not, is bounded by the dimenions of the grid. This happens to be $O(mn \cdot m)$ which boils down to still being polynomial in the size of the 3SAT instance. An example algorithm can simply place down the variable gadgets based on the spacing first, then place down the clause gadgets next, then draw the branches in between them. To do this, simply lead a branch down to a clause that contains that literal, then turn right, and build bridges if necessary.

To show (2), assume that the input 3SAT variable was satisfiable. Then we know there exists some variable assignment that satisfies it. Let that be given as $a_1, a_2, \ldots, a_n$ where $a_i = T$, ($a_i = F$) means we're assigning the i^{th} variable the value of TRUE (FALSE respectively). Then note that this means that all the sockets on the corresponding branches will be powered as well. Now from those sockets, we simply power all the sockets along those paths until they reach their clause gadgets.

Based on the variable assignment, we know that every clause is satisfied. What this means is that every clause gadget has at least one socket that is powered, since that clause contains a variable that satisfied it.

Now all that's left is to show is that if we can do this, then we can power all remaining sockets. (Note that up till this point, the only sockets that are left unpowered, are the ones on the opposite branches of the variable assignment and the bridges and elbows on their paths.)

As for the last part, since every clause has at least one socket that is powered, then we know that the broadcast node inside it can also be powered. Now note that any branches that are unpowered must lead to a clause gadget that is powered. So what we will do is power the remaining branches using the now powered sockets from the clause gadgets instead.

To show (3), notice that the only way the only source of power are from the variable gadgets. i.e. Therefore there needs to at least be one branch that is powered. And notice that variable sockets can only power at most one side of branches each.

Now assume that all the sockets managed to be powered, that there exists a wiring that does so (Note that we are not assuming that the 3SAT instance is solvable. Powering the clauses here has been argued independently of 3SAT.). We know that every variable gadget can only power one branch at most. Also note that by out assumption, all the clause sockets need to be satisfied as well. This can only be done if at least one of the branches leading to it are powered, which means some variable gadget must have chosen the corresponding branch. Now the last part of the argument is that notice that even when a clause is powered, it cannot power any other sockets besides the ones that are on the branch that lead to it. This is an important note since we need to make sure that the variable assignments correctly satisfy the clauses (and it was not because of "side effects"). What this means is that the clauses were powered based on the choice of branch to power by each variable gadget alone. Now consider the corresponding truth value assignment where we set the variable to TRUE if the left branch was powered, else we set it to FALSE. There might exist variable gadgets that need not power their left or right branches, for these, we will power either the left or the right (it doesn't matter).

Now the corresponding variable assignment must satisfy the 3SAT instance that was reduced to this PLUGIT instance. Firstly note that we will never double assign a variable to be true or false. Secondly notice that every clause has at least one branch leading to it that must have powered it (and not because of any other clause). And such a branch must have been powered due to the socket in the variable gadget, (and not because of any other clause).

Thus for any clause in the 3SAT instance, we have that there must be at least one of the variables that satisfies it. Therefore the 3SAT instance is satisfiable.

Grading scheme: Out of 15 marks:

1. 6 marks allocated for the reduction itself
2. 1 marks are for an argument that it runs in polynoimal time
3. 4 marks for correctly proving the reduction reduces satisfiable instances of 3SAT to solvable instances of PLUGIT
4. 4 marks for correctly proving the reduction reduces unsatisfiable instances of 3SAT to unsolvable instances of PLUGIT

For the reduction, 3 marks are given for the variable gadget, 3 marks are given for the clause gadget.

Partial credit and deductions are done in the following manner: Marks are capped at 10 for reductions from problems that are outside of the scope of CS3230. $2m$ are deducted for each of the 3 properties that the reduction fails to satisfy (out of the $6m$) on top of the marks deducted for the proofs.

E.g. If someone demonstrates a fully correct reduction but botches the proof that unsatisfiable instances of 3SAT get reduced to unsolvable instances of PLUGIT, then the submission is awarded at least $6 + 1 + 4$ marks (depending on the partial marks we can give for the last part).

On the other hand, if someone demonstrates a faulty reduction that runs in polynomial time but only reduces satisfiable instances of 3SAT to solvable instances of PLUGIT, then the submission is be awarded at most $4 + 1 + 4$ marks (assuming that the proofs were correct).

Grader Comments: A lot of reductions constructed PLUGIT instances that were always solvable no matter what. An issue was that gadgets were made for each clause, but were independent and any related variables were not linked. This means one could set $x_i = T$ for the clauses that needed it, then $x_i = F$ for the other clauses that needed it. This breaks the constraint that variables can only take on one of two truth values. Along those lines some reductions had this other issue where double assigments were indeed prevented, but satisfying a clause, say $(x_1 \vee x_2 \vee \neg x_3)$, for example by setting $x_1 = F$, $x_2 = T$, $x_3 = T$, in the corresponding PLUGIT grid, x_2 would be the one powering the gadget clause. But then because of that any clause that relies on x_1 being true or x_3 being false are now satisfied by extending wires from the first clause gadgets. A lot of attempts on proofs for the above submissions completely handwave what was supposed to be done and simply claimed "no instances are reduced to no instances". Or some people applied a circular argument or converse error instead.

One other issue was that some of the reductions gloss over details like how to connect variable gadgets to clause gadgets. Such reductions work from planar-3-SAT were awarded 12 marks.

There were a few people who tried to perform a reduction from HAMPATH instead. The issue here is that none of them specified how to draw edges between vertices when the degree of the vertex was larger than 4. Besides that, details are missing on how to be able to place them on the grid. i.e. deciding which row to put the sockets etc.

Other issues include: (1) Giving little to no details on how the reduction should be done; (2) No proofs on why it works; (3) Not understand the basic rules of the problem laid out in the pdf; (4) Doing a reduction from PLUGIT to 3SAT instead (wrong direction).

One interesting thing to note though, most working solutions were unique. Save for a few carbon copies between submissions.

Question 2

Solution: There can be a few possible certificates. The one we will detail here is as a set of pairs $(c1, c2)$ such that $c1$ is the pair of the x, y coordinates of the socket that is giving power, and $c2$ is the pair of the x, y coordinates of the socket that is receiving it.

Now the first thing we should do is to iterate through this list to make sure that the rules are fulfilled. Checking axis alignment can be done in $O(W)$ time where W is the number of wired tiles (not that there cannot be more wires than the dimensions of the grid. Likewise for making sure no two wires double power a socket, or some socket powers more than one other item, checks can be done in $O(W)$ time.

To make sure there are no collisions: Simply hash all the coordinates of all the tiles that are not empty. Then for each tile that has a wire, lookup the hash table to see whether it has been occupied. If not then add the coordinates of that tile into the hash table and repeat the process. In the worst case this takes $O((MN)^3)$ time, where the dimensions of grid are $M \times N$.

Now to make sure that every node is powered, we simply run a BFS with all powered sockets as sources, where the wires are directed edges. Also when the BFS reaches a broadcast node, enqueue any other broadcast nodes or sockets within the radius (takes $O(1)$ time list them). Then we just need to make sure that every socket has been visited by the BFS. This takes at most $O(MN)$ time.

(Note: There is a far more technical solution that has much tigher bounds on the size of the cert if one is strict about representation but that requires far far far more work so here we'll be lenient and relax the contraints)

Grading Scheme: Out of 10 marks, we'll be a little more liberal here. 3 marks for specifying a certificate that works. 3 marks for arguing why this certificate is polynomially sized compared to the instance. 4 marks for giving the verification algorith that runs in polynomial time.

-2 marks for giving an algorithm that runs slower than poly time. -4 marks instead if the verification algorithm clearly doesn't work. -3 for a certificate that doesn't work (I can't imagine how this might happen but I guess I might be surprised.) Also -3 marks if the argument that the certificate being polynomially sized isn't correct.

Grader Comments: Most people gave the algorithm without the certificate nor bounding the size of it, whereas some people gave the algorithm and the cert without bounding its size.

Question 3

Let $S = \{a_1, a_2, \ldots, a_n\}$ be the instance of the PARTITION problem. The reduction algoritm sets $A = S$, and $B = [\frac{T}{2}, \frac{T}{2}]$ where $T = \sum_{i=1}^{n} a_i$, and lastly set k to be n. This runs in $O(n)$ time since the algorithm just needs to copy the input once, then sum up the values and output half of the sum twice. The number of bits needed to represent the sum is at most $\log(nR) = \log(n) + \log(R)$, where R is the largest value that appears.

Let S be partitionable, and let S_1 and S_2 be the two partitions. We know that $\sum_{x \in S_1} x = \sum_{x \in S_2} x = \frac{T}{2}$, and $S_1 \cup S_2 = \emptyset$. Now just take the same people who owe the amounts in S_1, and get them to pay all their debts to the first person. Also people in S_2 can pay the second person. Since each person only made one transaction, we know that exactly n transactions are made.

To show the opposite direction (solvable in n transactions implies partitionable), note that first of all, n transactions are needed at least. This is because each of the borrowers needs to repay their debt, so each of them have make at least 1 transaction. **If it was doable in exactly n transactions, then we know each borrower can only make exactly 1 transaction. Therefore this means each borrower must choose one of the two people in B to repay, (and not both)**. For each person, if they paid the first person, we put the corresponding value they paid in S_1 and if they paid the second person, we put that corresponding value in S_2. By our previous arguments, we know that everyone belongs in either S_1 or S_2 but not both at the same time. Thus $S_1 \cup S_2 = \emptyset$. Also we know that both lenders correctly receive their amounts of $\frac{T}{2}$ (by the assumption) and thus $\sum_{x \in S_1} x = \sum_{x \in S_2} x = \frac{T}{2}$. This shows that S is partionable.

Without loss of generality assume that $\max\{n, m\} \leq k \leq nm$. Otherwise if $k < \max\{n, m\}$ always output NO (because for every element of A and B there must be at least one transaction), and if $k \geq mn$ always output YES (one can also argue about $m + n$ upper bound which is provided in the solution of the next question). The certificate needed is that one simply lists (s, t, v), to denote person s paying person t a value of v. There are only $|A| \times |B|$ many such triplets at most. But we need to argue about this in terms of bits. *Let R be the largest value in A and B. Then we need only $\log R$ bits to represent this value.* Then to encode the triplet in bits, we only need $\log n + \log m + \log R$ bits per triplet, and thus $(nm)(\log n + \log m + \log R)$ bits overall which is polynomial with respect to the input size (assuming $\max\{n, m\} \leq k \leq nm$). The verification algorithm simply makes a linear pass to make sure everyone pays/receives the correct amount. Then another linear pass to make sure there are at most n transactions.

Grading Scheme: 2 marks for the reduction, 1 mark for proving forward direction and 2 marks for proving the converse. 1 mark for the certificate, 1 mark for showing why it is polynomial in size, 1 mark for the verification algorithm.

Grader Comments: Most people missed out on the the portion of the proof laid out in bold, and instead had rather just claimed that "a partition exists". The issue with this is that when it is assumed that the transactions can be made, it does not immediately imply that we can partition one set of the people into two groups. What if someone made a double payment? Which set should we put that person in?

The second most common issue is bounding the certificate size. The issue here is that the values that need to be repaid are independent of the number of people involved. This means values in the certificate such as 2^{2^n} are possible, which means the number of bits required to encode that transaction alone would be about $\log(n) + \log(m) + 2^n$. The point of the italicised line above is to show that despite this, our certificate size is actually still polynomial with respect to the length of the encoded input.

Question 4

Let us start with an empty set T (to store transactions) and also initialize $k = 0$. Then consider the following Greedy strategy: Arbitrarily remove two elements $a \in A$ and $b \in B$. If $a < b$ then add $b - a$ to B, and add this transaction to T. Else if $a > b$ add $a - b$ to A, and add this transaction to T. Increment k by one. Repeat this greedy strategy until both the sets A, B become empty. Finally output k (and also set T if you want to output set of transactions as well, which we did not ask in this question).

Since at each greedy step we remove at least one element either from A or B, our algorithm terminates after at most $|A| + |B| = n + m$ steps. So output $k \leq n + m$. Note, at the termination the set T contains a valid set of transactions so that no body owes and no body is owed.

Let k^* be the minimum number of transactions. Observe that $k^* \geq \max\{n, m\}$ because for every element of A and B there must be at least one transaction.

So $k \leq n + m \leq 2\max\{n, m\} \leq 2k^*$. Hence the approximation ratio of the above algorithm is 2. Furthermore, our algorithm runs in linear time.

Grading Scheme: 6 marks for designing the algorithm. 4 marks for proving approximation ratio is 2, and 2 marks for analyzing the running time.