(1) Simplified Bloons Tower Defense

Inspired by a student's Piazza question, we will think about the hardness of solving Bloons Tower Defense. In this game, you are in charge of a team of N monkeys armed with weapons they can use to pop "bloons", or balloons, as they appear on a path on the map. The goal is to prevent the balloons from reaching the end of that path, which requires strategically placing monkeys to do damage effectively to oncoming balloons. Here, we remove time from the problem: every balloon is fixed in place, and each monkey can distribute some total amount of damage to any balloons in range. Your goal is to see if it is possible to choose a set of M monkeys from your team and place them to successfully pop all of the balloons. This problem is **NP**-Complete!

In this simplified Bloons Tower Defense, you are given:

- N monkeys, where the mth monkey has some integer total damage output d_m
- B stationary balloons, where each balloon b_i has some integer total health h_i
- L locations, where each location ℓ has a set of balloons A_ℓ that are within its range. A monkey at this location can only target the balloons within the location's range.

You would like to place as few monkeys as possible throughout these locations and direct each of those monkeys you placed to target a set of balloons within its range at that location. You do not need to use all L locations. A balloon b_i is considered popped if the sum of the damage directed at b_i from the monkeys targeting b_i equals or exceeds its health h_i. Each monkey is allowed to distribute its total damage output among its target balloons in any way as long as it inflicts integer non-negative damage to each target balloon. Let $d_{m,i}$ be the damage done by monkey m to balloon b_i.

The BloonsTowerDefense problem asks: **does there exist a valid assignment of at most M monkeys to these locations and balloons such that all the balloons can be popped?**

Show that BloonsTowerDefense is **NP**-Complete.

Solution:

Lemma 1. BloonsTowerDefense *is in* **NP**.

Proof. BloonsTowerDefense is in **NP** because given an assignment of monkeys to locations and the way they distribute their power to balloons, we can verify it is correct in polynomial time with respect to the length of the problem input (which in this case is a polynomial function of N, B, and L). Note that $M < N$, so $O(M)$ time is also polynomial in the problem input size.

- checking that there are at most M monkeys takes $\mathcal{O}(M)$ time;
- checking each location is used only once takes $\mathcal{O}(M)$ time;
- checking that each monkey m deals only nonnegative damage, and nonzero damage only to balloons b_i within its location ℓ_m's range, takes $\mathcal{O}(MB)$ time in total ($\mathcal{O}(B)$ per monkey);
- checking that the total damage each monkey deals to its target balloons equals its total damage output, $\sum_{i=1}^{B} d_{m,i} = d_i$, takes $\mathcal{O}(MB)$ time assuming $\mathcal{O}(1)$ arithmetic operations; and
- checking that each balloon is popped, $\sum_{m=1}^{M} d_{m,i} \geq h_i$, takes $\mathcal{O}(MB)$ time ($\mathcal{O}(M)$ per balloon).

Alternate solution to proving the problem is in NP: it is also enough to have the certificate list the k monkeys and their locations. Now the certifier has to verify that the power of the monkeys is enough to pop all the bloons. This can be formulated as a flow problem: a source s connected to all locations used with the total power of the monkeys placed at that location; edge from a location to all balloons that a monkey from that location can shoot with large (infinite) capacity, and an edge from each balloon to a sink with capacity that is the health of the balloon. This network has a flow of total value equal the sum of the balloon healths if and only if the moneys at these locations can distribute their power to pop all balloons. □

We offer a few proofs for the hardness part:

Lemma 2. BloonsTowerDefense *is* **NP**-*Hard.*

Proof. We show reductions from SetCover, VertexCover, SAT, and SubsetSum.

Using SetCover. Given a collection of sets $C = \{S_1, \ldots, S_N\}$ where $\bigcup_{i=1}^{N} S_i = S$, SetCover asks does there exist a collection C' of at most k sets S_i such that the union of sets in C' equals S. We reduce SetCover to BloonsTowerDefense. Given an instance of SetCover, we create an instance of BloonsTowerDefense with N monkeys each with total damage output $|S|$, $|S|$ balloons each with total health 1 corresponding to each element of S, and N locations where each location can target the balloons corresponding to elements in set S_i. Also we let M in BloonsTowerDefense equal k from SetCover.

If there exists a collection C' of size at most k, then we simply choose to place $|C'|$ monkeys in the locations corresponding to sets in C'. Since each monkey has enough damage to destroy all balloons on the map if they were in range, we know that this assignment of monkeys to locations and target balloons will pop all the balloons.

On the other hand, if there does not exist a collection C' of size at most k, this mean there does not even exist a set of at most k locations that can cover every balloon on the map. Since we need more than k locations to even possibly cover every balloon on the map, and we need at least one monkey at each location in order to cover the balloons within the range of that location, we definitely need more than k monkeys to even possibly pop all the balloons. Thus there does not exist an assignment of at most k monkeys to locations and balloons such that all the balloons can be popped.

This is a polynomial reduction because it takes $\mathcal{O}(N)$ time to initialize the locations and the monkeys, $\mathcal{O}(|S|)$ time to initialize the balloons, and $\mathcal{O}(N|S|)$ time to create the set of balloons in each location's range for each location.

Using VertexCover. The reduction from VertexCover is completely analogous to SetCover: the possible locations L will be the vertices of the graph, the balloons are the edges, edge E with health 1. We have k monkeys each with defense power the number of edges $m = |E|$, and we want to place k monkeys to pop all m.

Using SAT. Consider a SAT problem with n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$. We'll have $n + m$ balloons, one for each variable and one for each clause, each with health 1. We have n monkeys with $k = n$ with $2n$ possible positions, two for each literal: for each variable x_i one position corresponding to x_i, and one corresponding to $\neg x_i$. Monkeys each have total damage output $m + 1$. The balloon at variable x_i can be popped from the location x_i or $\neg x_i$; the balloon corresponding the clause c_j can be popped from any location corresponding to a literal that makes the clause true.

We claim that for any instance of the SAT problem, the instance is satisfiable if and only if the BloonsTowerDefense defined above is satisfiable.

Suppose the SAT formula is satisfiable, and consider a satisfying assignment. Now locate a monkey at the version of x_i or $\neg x_i$ which is set to true, use this monkey to pop the balloon at x_i, and then use the monkey at one of the true literals of each clause to pop the balloon at the clause.

Next assume that the BloonsTowerDefense problem is solvable. Note that the balloon at variable x_i can only be popped from either x_i or $\not{x}_i$, and with only n monkeys, we need to use exactly one of the pair of locations $\{x_i, \not{x}_i\}$ for each variable i. Now we claim that the selected locations must define a satisfying assignment of the SAT formula. Consider a clause c_j, and the corresponding balloon. This balloon was popped, and it can only be popped from a location that sets the clause true, proving our claim.

Using SubsetSum. Consider a SubsetSum problem with n integers $w_1, \ldots, w_n$ and an integer W. The subset sum problem is asking if there is a subset $I \subset \{1, \ldots, n\}$ such that $\sum_{i \in I} w_i = W$. Let $\bar{W} = \sum_i w_i - W$. We define a BloonsTowerDefense as follows. We'll have two locations, say called location 1 and 2, and two balloons. Balloon 1 can be targeted from location 1 and has health W, while

balloon 2 can be targeted from location 2 and has health $\bar{W}$. We'll have n monkeys where monkey i has defense power w_i, and will have $k = n$. We claim that for any instance of the SubsetSum problem, the instance is satisfiable if and only if the BloonsTowerDefense defined above is satisfiable.

Suppose SubsetSum is satisfiable, then put monkey i to location 1 if $i \in I$ and otherwise to location 2. Now both locations the moneys allocated to that location have enough power to pop the one balloon they can target.

Next suppose the BloonsTowerDefense problem is solvable, and let I denote the set of moneys assigned to location 1. There monkeys have enough power to pop the balloon at location 1, so we have $\sum_{i \in I} w_i \geq W$. On the other hand, the monkeys at location 2 have enough power to pop balloon 2, so we get that $\sum_{i \notin I} w_i \geq \bar{W}$. However $\sum_i w_i = \sum_{i \in I} w_i + \sum_{i \notin I} w_i \geq W + \bar{W} = \sum_i w_i$, implying that we must have equation in both inequalities, and hence have a solution for the SubsetSum problem. □

Common Mistakes

- Reduction in the wrong direction. It's important to reduce from a known NP-complete problem to our new problem! In this problem, reduction in the wrong direction is very hard!
- Some students only show how to make certifiers in the special kind of problem we get via the reduction (say using VertexCover: all monkeys have large power and the health of the baloons is 1). Certifiers should be for the general problem.
- some students have a correct reduction but missing one of the two directions of the if and only if proof showing the validity of the reduction
- some students forget to point out that the reduction is done in polynomial time.

(2) Monotone Almost-All-True SAT

A monotone SAT formula is a SAT formula with no negated variables. So, for example,

$$\Phi = (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_4)$$

is a monotone formula: each of x_1, x_2, x_3, x_4 only appears as a positive literal, never with a negation. A monotone formula is easy to satisfy: we can simply set all variables to be `true`, and this way all variables in each clause become true. Since we only need one variable to be `true` in each clause to satisfy the formula, this valuation easily satisfies the formula.

In this problem we have a stronger goal than typical satisfiability: we want to ensure that in each monotone SAT clause, at most one variable evaluates to false (we assume each clause has at least 2 variables). This is still easy to do by setting all variables to `true`, but becomes nontrivial if we also require our assignment to set some number of variables to `false`.

In the Monotone Almost-All-True SAT problem, we are given a monotone formula Φ and an integer k. We ask if there is a way to set k variables to `false` (and the rest to `true`) such that no more than 1 variable in each clause is `false`. For example, for the formula above, if $k = 1$, then we can find an assignment satisfying these constraints by setting any one variable to `false`. It is impossible, however, to find an assignment that sets $k = 2$ variables to `false` while ensuring that at most one variable in each clause is `false`.

Show that the Monotone Almost-All-True SAT problem is NP-complete.

Solution. Almost-All-True SAT is clearly in NP. For our certificate to verify a solution, all we need is a list of the k variables to set to `false`; to verify, all we need to do is to show the k variables keep all the literals in all clauses Almost-All-True. The list of variables is linear in the number of variables and the certifier program takes time linear in the number of terms.

To prove that the problem is NP-hard (and thus NP-complete), we show that Independent Set $\leq$ Monotone Almost-All-True SAT.

(i) **Reduction.** Independent Set is given by an undirected graph G and integer k. To reduce, we need to transform the input of the independent set problem to an equivalent input to Monotone Almost-All-True SAT. To do this, we use a variable x_v for each node v in the graph, and add a clause $(x_v \vee x_w)$ for all edges (v, w). Let $\Phi = \wedge_{(v,w)\in E}(x_v \vee x_w)$, and let k be the same as in the independent set problem.

(ii) **Polynomial-time?** This reduction requires writing out $2m$ total terms for an edge with m graphs and is thus polynomial-time.

(iii) **Proof of correctness.** Now, we prove that a solution to the constructed Monotone Almost-All-True SAT problem corresponds 1:1 to a solution to the original Independent Set problem:

- Given a set of k nodes S in V that form an independent set, setting the corresponding variables x_v to `false` must satisfy the constraints: if there were more than one term set to `false` in a clause, then there would have been an edge between the corresponding vertices in the original graph, and thus there would not have been an independent set.
- If the above constraints are satisfiable by setting k variables false for the above formula Φ, then the nodes corresponding to these false variables form an independent set: if two of these variables had an edge between their corresponding vertices in the original graph, then there would have been a clause in Φ that had both of variables negated, exceeding our limit of 1 negated variable.

Common Mistakes

- Reduction in the wrong direction. It's important to reduce from a known NP-complete problem to our new problem! In this problem, reduction in the wrong direction is just as simple: one can turn an input to our Almost-All-True SAT problem into an equivalent independent set problem with the variables as nodes of the graph. But this is only proving that Almost-All-True SAT $\leq$ Independent Set, but this only proves that Independent set is hard, which we know.
  Here is how a backwards reduction goes: We now have an formula as input and need to create a graph G: variables of the formula correspond to vertices, and connect two vertex by an edge if they two occur together in a clause. Use the same k. The resulting graph has an independent set of size k if and only if the Almost-All-True SAT has a solution.