On the surface, *Monotone Satisfiability with Few True Variables* (which we'll abbreviate *Monotone Satisfiability with Few True Variables*) is written in the language of the Satisfiability problem. But at a technical level, it's not so closely connected to *SAT*; after all no variables appear negated, and what makes it hard is the constraint that only a few variables can be set to true.

Really, what's going on is that one has to choose a small number of variables, in such a way that each clause contains one of the chosen variables. Phrased this way, it resembles a type of covering problem.

We choose *Vertex Cover* as the problem $X$, and show *Vertex Cover* $\leq_P$ *Monotone Satisfiability with Few True Variables*. Suppose we are given a graph $G = (V, E)$ and a number $k$; we want to decide whether there is a vertex cover in $G$ of size at most $k$. We create an equivalent instance of *Monotone Satisfiability with Few True Variables* as follows. We have a variable $x_i$ for each vertex $v_i$. For each edge $e_j = (v_a, v_b)$, we create the clause $C_j = (x_a \vee x_b)$. This is the full instance: we have clauses $C_1, C_2, \ldots, C_m$, one for each edge of $G$, and we want to know if they can all be satisfied by setting at most $k$ variables to 1.

We claim that the answer to the *Vertex Cover* instance is "yes" if and only if the answer to the *Monotone Satisfiability with Few True Variables* instance is "yes." For suppose there is a vertex cover $S$ in $G$ of size at most $k$, and consider the effect of setting the corresponding variables to 1 (and all other variables to 0). Since each edge is covered by a member of $S$, each clause contains at least one variable set to 1, and so all clauses are satisfied. Conversely, suppose there is a way to satisfy all clauses by setting a subset $X$ of at most $k$ variables to 1. Then if we consider the corresponding vertices in $G$, each edge must have at least one end equal to one of these vertices — since the clause corresponding to this edge contains a variable in $X$. Thus the nodes corresponding to the variables in $X$ form a vertex cover of size at most $k$.

[1] ex799.396.989

*Zero-weight-cycle* is in $\mathcal{NP}$ because we can exhibit a cycle in $G$, and it can be checked that the sum of of the edge weights on this cycle are equal to 0.

We now show that *Subset Sum* $\leq_P$ *Zero-weight-cycle*. We are given numbers $w_1, \ldots, w_n$, and we want to know if there is a subset that adds up to exactly $W$. We construct an instance of *Zero-weight-cycle* in which the graph has nodes $0, 1, 2, \ldots, n$, and an edge $(i, j)$ for for all pairs $i < j$. The weight of edge $(i, j)$ is equal to $w_j$. Finally, there is an edge $(n, 0)$ of weight $-W$.

We claim that there is a subset that adds up to exactly $W$ if and only if $G$ has a zero-weight cycle. If there is such a subset $S$, then we define a cycle that starts at 0, goes through the nodes whose indices are in $S$, and then returns to 0 on the edge $(n, 0)$. The weight of $-W$ on the edge $(n, 0)$ precisely cancels the sum of the other edge weights. Conversely, all cycles in $G$ must use the edge $(n, 0)$, and so if there is a zero-weight cycle, then the other edges must exactly cancel $-W$ — in other words, their indices must form a set that adds up to exactly $W$.

[1] ex642.498.819

**(a)** This problem can be solved using network flow. We construct a graph with a node $v_i$ for each cannister, a node $w_j$ for each truck, and an edge $(v_i, w_j)$ of capacity 1 whenever cannister $i$ can go in truck $j$. We then connect a super-source $s$ to each of the cannister nodes by an edge of capacity 1, and we connect each of the truck nodes to a super-sink $t$ by an edge of capacity $k$.

We claim that there is a feasible way to place all cannisters in trucks if and only if there is an $s$-$t$ flow of value $n$. If there is a feasible placement, then we send one unit of flow from $s$ to $t$ along each of the paths $s, v_i, w_j, t$, where cannister $i$ is placed in truck $j$. This does not violate the capacity conditions, in particular on the edges $(w_j, t)$, due to the capacity constraints. Conversely, if there is a flow of value $n$, then there is one with integer values. We place cannister $i$ in truck $j$ if the edge $(v_i, w_j)$ carries one unit of flow, and we observe that the capacity condition ensures that no truck is overloaded.

The running is the time required to solve a max-flow problem on a graph with $O(m+n)$ nodes and $O(mn)$ edges.

**(b)** When there are conflicts between pairs of cannisters, rather than between cannisters and trucks, the problem becomes NP-complete.

We show how to reduce *3-Coloring* to this problem. Given a graph $G$ on $n$ nodes, we define a cannister $i$ for each node $v_i$. We have three trucks, each of capacity $k = n$, and we say that two cannisters cannot go in the same truck whenever there is an edge between the corresponding nodes in $G$.

Now, if there is a 3-coloring of $G$, then we can place all the cannisters corresponding to nodes assigned the same coloring in a single truck. Conversely, if there is a way to place all the cannisters in the three trucks, then we can use the truck assignments as colors; this gives a 3-coloring of the graph.

[1] ex460.602.46

There are two basic ways to do this. Let $G = (V, E)$; we can give the answer without using $\mathcal{A}$ if $V = \phi$ or $k = 1$, and so we will suppose $V \neq \phi$ and $k > 1$.

The first approach is to add an extra node $v^*$ to $G$, and join it to each node in $V$; let the resulting graph be $G^*$. We ask $\mathcal{A}$ whether $G^*$ has an independent set of size at least $k$, and return this answer. (Note that the answer will be yes or no, since $G^*$ is connected.) Clearly if $G$ has an independent set of size at least $k$, so does $G^*$. But if $G^*$ has an independent set of size at least $k$, then since $k > 1$, $v^*$ will not be in this set, and so it is also an independent set in $G$. Thus (since we're in the case $k > 1$), $G$ has an independent set of size at least $k$ if and only if $G^*$ does, and so our answer is correct. Moreover, it takes polynomial time to build $G^*$ and ask call $\mathcal{A}$ once.

Another approach is to identify the connected components of $G$, using breadth-first search in time $O(|E|)$. In each connected component $C$, we call $\mathcal{A}$ with values $k = 1, \ldots, n$; we let $k_C$ denote the largest value on which $\mathcal{A}$ says "yes." Thus $k_C$ is the size of the maximum independent set in the component $C$. Doing this takes polynomial time per component, and hence polynomial time overall. Since nodes in different components have no edges between them, we now know that the largest independent set in $G$ has size $\sum_C k_C$; thus, we simply compare this quantity to $k$.

[1] ex30.643.488

To show that *Number Partitioning* is in NP we use the subset $S \subset \{1, \ldots, n\}$ such that $\sum_{i \in S} w_i = \frac{1}{2} \sum_{i-1}^{n} w_i$ as the certificate. Given a certificate we can add the corresponding numbers in polynomial time and test if the claimed equation holds.

To prove that *Number Partitioning* is NP-complete, we show that *Subset-Sum* $\leq_P$ *Number Partitioning*. Consider an arbitrary instance of *Subset Sum* with numbers $w_1, \ldots, w_n$ and target sum $W$. We will construct an equivalent instance of *Number Partitioning*. Let $T = \sum_{i=1}^{n} w_i$ be the total sum of all numbers. Add two numbers $w_{n+1} = W + 1$ and $w_{n+2} = T + 1 - W$. Note that the sum of all $n + 2$ numbers is $2T + 2$. We claim that the partition problem with these $n + 2$ numbers is equivalent to the original *Subset Sum* instance. To prove this, assume first that the answer in the *Subset Sum* problem is "yes", there is a subset $S$ such that $\sum_{i \in S} w_i = W$. Now we can create a partition solution by adding $w_{n+2}$ to the subset $S$, and using all other numbers as the other part. Now assume conversely that the answer in the *Number Partitioning* problem is "yes", there is a partition where the two parts have equal sums, that is, they both sum to $T + 1$. Note that $w_{n+1}$ and $w_{n+2}$ cannot be in the same part as $w_{n+1} + w_{n+2} > T + 1$. Consider the part that contains $w_{n+2}$. The sum of all numbers in this part is $T + 1$, so the numbers other than $w_{n+2}$ must sum to $W$.

Note that it was important to add the $+1$ in both $w_{n+1}$ and $w_{n+2}$. A natural first idea would have been to use $w_{n+1} = W$ and $w_{n+2} = T - W$. However, this instance of *Number Partitioning* is always "yes", independent of the answer in the original *Subset Sum* problem, as now the total sum is $2T$ and $w_{n+1} + w_{n+2} = T$.

[1] ex123.267.365

**(a)** We'll say a set of advertisements is "valid" if it covers all paths in $\{P_i\}$. First, *Strategic Advertising* (SA) is in NP: Given a set of $k$ nodes, we can check in $O(kn)$ time (or better) whether at least one of them lies on a path $P_i$, and so we can check whether it is a valid set of advertisements in time $O(knt)$.

We now show that *Vertex Cover* $\leq_P$ SA. Given an undirected graph $G = (V, E)$ and a number $k$, produce a directed graph $G' = (V, E')$ by arbitrarily directing each edge of $G$. Define a path $P_i$ for each edge in $E'$. This construction involves one pass over the edges, and so takes polynomial time to compute. We now claim that $G'$ has a valid set of at most $k$ advertisements if and only if $G$ has a vertex cover of size at most $k$. For suppose $G'$ does have such a valid set $U$; since it meets at least one end of each edge, it is a vertex cover for $G$. Conversely, suppose $G$ has a vertex cover $T$ of size at most $k$; then, this set $T$ meets each path in $\{P_i\}$ and so it is a valid set of advertisements.

**(b)** We construct the algorithm by induction on $k$. If $k = 1$, we simply check whether there is any node that lies on all paths. Otherwise, we ask the fast algorithm $\mathcal{S}$ whether there is a valid set of advertisements of size at most $k$. If it says "no," we simply report this. If it says "yes", we perform the following test for each node $v$: we delete $v$ and all paths through it, and ask $\mathcal{S}$ whether, on this new input, there is a valid set of advertisements of size at most $k - 1$. We claim that there is at least one node $v$ where this test will succeed. For consider any valid set $U$ of at most $k$ advertisements (we know one exists since $\mathcal{S}$ said "yes"): The test will succeed on any $v \in U$, since $U - \{v\}$ is a valid set of at most $k - 1$ advertisements on the new input.

Once we identify such a node, we add it to a set $T$ that we maintain. We are now dealing with an input that has a valid set of at most $k - 1$ advertisements, and so our algorithm will finish the construction of $T$ correctly by induction. The running time of the algorithm involves $O(n + t)$ operations and calls to $\mathcal{S}$ for each fixed value of $k$, for a total of $O(n^2 + nt)$ operations.

[1] ex685.1.698

The problem is in $\mathcal{NP}$ since we can exhibit a set $X$ and check the size of its intersection with every set $A_i$.

We now show that *3-Dimensional Matching*$\leq_P$ *Intersection Inference*. Suppose we are given an instance of *3-Dimensional Matching*, consisting of sets $X$, $Y$, and $Z$, each of size $n$, and a set $T$ of $m$ triples from $X \times Y \times Z$. We define the following instance of *Intersection Inference*. We define $U = T$. For each element $j \in X \cup Y \cup Z$, we create a set $A_j$ of these triples that contain $j$. We then ask whether there is a set $M \subseteq U$ that has an intersection of size 1 with each set $A_j$.

Such sets are precisely those collections of triples for which each element of $X \cup Y \cup Z$ appears in exactly one: in other words, they are precisely the perfect three-dimensional matchings. Thus, our instance of *Intersection Inference* has a positive answer if and only if the original instance of *3-Dimensional Matching* does.

[1] ex803.795.220

Let us call this problem *Trade*. *Trade* is in NP, since a pair of subsets $A_i$ and $A_j$ can serve as a certificate. We can verify that

- $\sum_{a \in A_j} v_i(a) > \sum_{a \in A_i} v_i(a)$
- $\sum_{a \in A_i} v_j(a) > \sum_{a \in A_j} v_j(a)$

by summing up the valuation of each person, which is clearly polynomial time doable.

We will prove *Trade* is NP-complete by reduction from *Subset Sum* problem. Suppose we have an instance of *Subset Sum*, that is, $n$ integers $w_1, w_2, \ldots, w_n$, and another integer $W$. The goal is to find a subset $S \subseteq w_1, w_2, \ldots, w_n$, s.t. $\sum_{w_k \in S} w_k = W$.

Construct an instance of *Trade* as follows: there are $n+1$ objects $a_1, a_2, \ldots, a_{n+1}$, $p_i$ possesses objects $a_1, a_2, \ldots, a_n$, and $p_j$ possesses $a_{n+1}$. The first $n$ objects are corresponding to the $n$ numbers in *Subset Sum* problem, and the valuations for them are $v_i(a_k) = v_j(a_k) = w_k \ \ (k = 1, 2, \ldots, n)$. The valuations of the last object are $v_i(a_{n+1}) = W + 1$, and $v_j(a_{n+1}) = W - 1$.

Since $p_j$ only possesses a single object $a_{n+1}$, so if there exist $A_i$ and $A_j$, then $A_j$ must be $\{a_{n+1}\}$. $A_i$ will be a subset of $\{a_1, a_2, \ldots, a_n\}$.

Now we will prove that if there is a subset of numbers that sums up to $W$, if and only if there is a subset $A_i \subseteq \{a_1, a_2, \ldots, a_n\}$, together with $A_j = \{a_{n+1}\}$, satisfying the two inequalities stated before.

If there is a subset $S$ for *Subset Sum* problem, s.t. $\sum_{k \in S} w_k = W$, then we let

$$A_i = \{a_k : w_k \in S\}$$

According to our construction:

$$\sum_{a \in A_j} v_i(a) = v_i(a_{n+1}) = W + 1 > W = \sum_{a \in A_i} v_i(a)$$

and

$$\sum_{a \in A_i} v_j(a) = W > W - 1 = v_j(a_{n_1}) = \sum_{a \in A_j} v_j(a)$$

so $A_i$ and $A_j$ satisfy the requirement of $Trade$ problem.

If there is a subset $A_i \subseteq \{a_1, a_2, \ldots, a_n\}$, and $A_j = \{a_{n+1}\}$ satisfying the two inequalities, then we will let

$$S = \{w_k : a_k \in A_i\}$$

According to our construction, $\sum_{a \in A_i} v_j(a) > W - 1$, $\sum_{a \in A_i} v_i(a) < W + 1$, and $\sum_{w_k \in S} w_k = \sum_{a \in A_i} v_i(a) = \sum_{a \in A_i} v_j(a)$, so $\sum_{w_k \in S} w_k = W$.

[1] ex508.142.105

*Galactic Shortest Path* is in NP: given a path $P$ in a graph, we can add up the lengths and risks of its edges, and compare them to the given bounds $L$ and $R$.

*Galactic Shortest Path* involves adding numbers, so we naturally consider reducing from the *Subset Sum* problem. Specifically, we'll prove that *Subset Sum* $\leq_P$ *Galactic Shortest Path*.

Thus, consider an instance of *Subset Sum*, specified by numbers $w_1, \ldots, w_n$ and a bound $W$; we want to know if there is a subset $S$ of these numbers that add up to exactly $W$. *Galactic Shortest Path* looks somewhat different on the surface, since we have *two kinds* of numbers (lengths and risks), and we are only given *upper bounds* on their sums. However, we can use the fact that we also have an underlying graph structure. In particular, by defining a simple type of graph, we can encode the idea of choosing a subset of numbers.

We define the following instance of *Galactic Shortest Path*. The graph $G$ has a nodes $v_0, v_1, \ldots, v_n$. There are two edges from $v_{i-1}$ to $v_i$, for each $1 \leq i \leq n$; we'll name them $e_i$ and $e'_i$. (If one wants to work with a graph containing no parallel edges, we can add extra nodes that subdivide these edges into two; but the construction turns out the same in any case.)

Now, any path from $v_0$ to $v_n$ in this graph $G$ goes through edge one from each pair $\{e_i, e'_i\}$. This is very useful, since it corresponds to making $n$ independent binary choices — much like the binary choices one has in *Subset Sum*. In particular, choosing $e_i$ will represent putting $w_i$ into our set $S$, and $e'_i$ will represent leaving it out.

Here's a final observation. Let $W_0 = \sum_{i=1}^{n} w_i$ — the sum of all the numbers. Then a subset $S$ adds up to $W$ if and only if its complement adds up to $W_0 - W$.

We give $e_i$ a length of $w_i$ and a risk of 0; we give $e'_i$ a length of 0 and a risk of $w_i$. We set the bound $L - W$, and $R - W_0 - W$. We now claim: there is a solution to the *Subset Sum* instance if and only if there is a valid path in $G$. For if there is a set $S$ adding up to $W$, then in $G$ we use the edges $e_i$ for $i \in S$, and $e'_j$ for $j \notin S$. This path has length $W$ and risk $W_0 - W$, so it meets the given bounds. Conversely, if there is a path $P$ meeting the given bounds, then consider the set $S = \{w_i : e_i \in P\}$. $S$ adds up to at most $W$ and its complement adds up to at most $W_0 - W$. But since the two sets together add up to exactly $W_0$, it must be that $S$ adds up to exactly $W$ and its complement to exactly $W_0 - W$. Thus, $S$ is valid solution to the *Subset Sum* instance.

[1] ex129.970.939

This problem can be decided in polynomial time. It helps to view the $QSAT$ instance as a $CSAT$ instance instead: Player 1 controls the set $A$ of odd-indexed variables while Player 2 controls the set $B$ of even-indexed variables. Our question then becomes: can Player 1 force a win?

We claim that Player 1 can force a win if and only if each clause $C_i$ contains a variable from $A$. If this is the case, Player 1 can win by setting all variables in $A$ to 1. If this is not the case, then some clause $C_i$ has no variable from $A$. Player 2 can then win by setting all variables in $B$ to 0: in particular, this will cause the clause $C_i$ to evaluate to 0.

[1] ex63.946.695