Problem 1 - Complicated Class Scheduling

To prove that CCS is NP-complete begins with proving that CCS is in NP, which is simple enough - given a successful configuration of class intervals using C classrooms, it's easy to see that the CCS problem returns "yes" for that number C - simply, at no time in the schedule do more than C classes overlap.

NP-completeness will require a reduction, however, and we'll use k-coloring on undirected graphs for that, where k is the number of classrooms in the corresponding constructed CCS problem. Any given graph which can sustain a k-coloring can be transformed into a successful C-classroom CCS in polynomial time, and no answers to the k-coloring question, too, are no answers to CCS.

1. A k-coloring on a particular graph $G(V, E)$ corresponds to $k = C$ classrooms. $|V|$ vertices in G correspond to $|V| = n$ classes, and most importantly, in examining the $\frac{|V|^2-|V|}{2}$ *possible* edges between all pairs of vertices in the graph, those edges that are actually present in it ($(v_1, v_2) \in E$) correspond to an overlapping interval between those two vertex-classes in the CCS problem, and those that are not present ($(v_1, v_2) \notin E$) correspond to *no* overlap between those two vertex-classes in the CCS, strictly. Once a vertex-class is assigned a room, it must remain there for all other intervals it overlaps.
2. If a black box solver for this CCS problem returns "yes", a k-coloring on G is possible. If it returns no, such a k-coloring is not possible.

That'll do it. The reason this works is essentially the following: it is pretty obvious that if two vertices in G are connected by an edge, they can't both be the same color, and if they aren't, they can be the same color. In the constructed CCS problem, this corresponds to two classes being simultaneously scheduled and thus requiring at least two rooms, and being not simultaneously scheduled and thus able to be housed in the same room at different time periods. It's worth mentioning here that the actual start and finish times of each interval don't particularly matter - all we care about is whether or not certain classes need to or need to not overlap, and this means that for certain classes, whether or not their intervals are contiguous isn't important (consider the case where vertices $1, 2, 3$ form a cycle, and $2, 3, 4$ form a cycle - a 3-coloring is possible - and the corresponding CCS has overlapping $1, 2, 3$ and $2, 3, 4$ where 1 and 4 strictly do not overlap. In this particular case, it doesn't matter much whether 1 or 4 go first, or whether 2 and 3 are disjoint intervals or one long one each.)

So, if an edge connects two vertices, they aren't the same color; if an edge doesn't connect them, they can be. Consider the case of the complete graph for n vertices, in which all possible edges exist and as such the minimum k for a k-coloring is n (as all vertices touch all others). The corresponding CCS has all classes occur simultaneously and thus requires $n = C$ classrooms. Removing a single edge from the complete graph will make two vertices no longer adjacent, allowing them to be the same color and thus in the corresponding CCS allowing them to occur at different times - now, an $n - 1$-coloring is possible and a CCS is doable with $n - 1$ classrooms. This continues onwards - every time an edge is removed and as a result, a pair of vertices becomes no longer adjacent, the corresponding CCS graph can move the corresponding classes so that they're no longer overlapping.

In essence, every time that some CCS problem of this form returns "yes" on some C, it's because there's some formulation given the required number of intervals overlapping per class that the black box solver can successfully fit into C classrooms. The corresponding k-coloring problem will thus also return "yes" - some series of adjacent vertices translates into a series of overlapping classes, and because the condition for a successful k-coloring is just that any given vertex is not the same color as any of its adjacent vertices (and thus that any 'cluster' of vertices all adjacent to each other consist of at most k vertices), a successful k-classroom CCS is just one for which any given class does not overlap more than $k - 1$ other classes (and thus that any 'cluster' of classes all overlapping is of size at most k). Any yes answer for CCS will thus return a yes answer for k-coloring for a given k, and the reverse is true as well - a "yes" for a given k-coloring indicates that there is some way to color the vertices such that no adjacent vertices are the same color using k colors, and thus that the corresponding CCS problem constructed from the k-colored graph overlaps at most k classes at any given time, which follows directly from the fact that adjacent vertices need to be overlapped.

All told, then, k-coloring is reducible to CCS in polynomial time, and a black-box solver for CCS can solve k-coloring in whatever time the black box is running. CCS is thus NP-hard: this in tandem with the fact that CCS is in NP means that, like k-coloring, CCS is NP-complete.

Problem 2 - Linear Equations With Outliers

We don't have to prove this one is in NPC, just that's it's NP-hard, so let's jump right into it. Subset Sum is the problem we're going to reduce from, which is organized as the following: the input to Subset Sum is a set of natural numbers $\{w_1, w_2...w_n\}$ and a target number W - is it possible using the sum of a subset of the input set to add up to exactly W? This is a known NP-complete problem, and here, we're going to be constructing LEO instances out of it like so:

1. For the n variables in the Subset Sum problem, we'll have n variables and $2n + 1$ equations in LEO: n sets of parallel lines and one representing the actual Subset Sum problem. This will manifest as two linear equations for each w_i, those being $x_i = 1$ and $x_i = 0$. For each of the n variables, this is a total of $2n$ lines.
2. The last equation is just $w_1 * x_1 + w_2 * x_2 + ... + w_n * x_n = W$. Here, the weights of the items in the Subset Sum problem are the coefficients of the variables in the equation, and the final weight is the constant.
3. Lastly, define k for the LEO problem as n. Run the LEO black box solver on the resultant $2n + 1$ equations, allowing for n of them to fail, and if the other $n + 1$ have an intersection on some set of n variables, return "yes" for Subset Sum. Otherwise, return "no".

This one is fairly simple, as is its explanation - it is evident that in LEO, having $2n$ equations of the form $x_i = 0$ and $x_i = 1$ makes a series of parallel lines which can only be intersected one at a time - that is to say, the only equation not running as orthogonal lines in the ith dimension, the one with W as its constant, can only intersect either $x_i = 0$ or $x_i = 1$; not both. That automatically deliniates the $n = k$ outlier equations which are allowed to fail, meaning that to succeed, the equation with W as its constant must succeed (and additionally that each x_i value must be 0 or 1, of course - you can't violate both a parallel equation and its pair, or more than n equations will be outliers). Knowing that, the LEO black box basically just has to return some settings of all variables to 1 and 0 so that the W-constant equation is successful - if this can be done, "yes", if not, "no".

What makes this congruent with Subset Sum is the fact that $x_i = 1$ is basically the same as "inclusion" in the sum, and $x_i = 0$ is the same as "exclusion"; setting some $x_i = 1$ ensures that the corresponding w_i coefficient will be added towards W in LEO and setting it otherwise is just not using that particular w_i. Thus, meeting the required $n + 1$ equations is just the same as picking a series of inclusions(1) and exclusions(0) and finding which (if any) meet the W-constant equation. A "yes" answer for whether or not this is possible for a constructed LEO is a "yes" for the Subset Sum, as it corresponds to a series of inclusions and exclusions of items that add up to W. A "yes" on the Subset Sum problem, too, indicates that there exists an included subset of items whose sum is W, meaning that having some of the $2n$ parallel equations set to 1 and the others set to 0 - values required by the $n = k$ outliers constraint, as I said above - produces a series of $w_i * 0$ or $w_i * 1$ terms in the W-constant equation, which, when summed, equals W. A result of "no" in one indicates a result of "no" in the other, as the inclusions and exclusions correspond to intersections with a given variable's 1 line or its 0 line.

That said, it takes polynomial time to reduce a Subset Sum problem to a LEO problem (n time to set up the 0s, n time to set up the 1s, and then n variables to account for in the W-constant equation, so linear time), and a black box solver for constructed LEO equations of this form would give correct answers for Subset Sum. LEO is thus at least as hard as Subset Sum: Subset Sum $\leq_P$ LEO, and as Subset Sum is known to be NP-hard, so, too, is LEO.

Problem 3 - Fair Teams

The problem here is a little more interesting: we have some number p of people who among them possess s skills (some people with multiple skills and some not), and we are asked to create a fair team based on equivalent distributions of the same skill - which is to say, a polymath versus an army of people each possessing one of her skills is deemed fair. That in mind, let's prove the Fair Teams (FT) problem to be in NP: given a particular list of people denoted as being in team A, their combined skillholders of each skill must equal the combined skillholders for each skill in team B (or, really, team not-A). This is basically as simple as being given certain rows to calculate and hold a series of values for and then some other rows and to match those values - very clearly a polynomial operation. FT $\in$ NP.

Now for the fun part. I'm going to reduce a perfect 3-Dimensional Matching (3DM, as described on p. 481 of Kleinberg-Tardos) to FT, solving 3DM by creating a polynomial algorithm that could, if given a black-box solver for FT, solve 3DM. 3DM involves a series of three sets of vertices, X, Y, and Z, and a set of ordered triples (x, y, z), or edges, and asks if it's possible to touch every vertex in the problem with one single edge. We build an FT problem out of a 3DM problem as follows:

1. Each vertex in all sets corresponds to a single skill: for the n vertices between X, Y, and Z, there are $n = s$ skills, or columns in the FT table. Each edge, or triple, across the sets corresponds to a single person: if there are m edges through the 3DM problem, there are $m = p$ people, or rows, in the FT table. Fill in each $T(a, c)$ value in the table where a is a person and c is a skill with 1 if the edge corresponding to a touches the vertex corresponding to c, and with 0 otherwise.

2. That is, to start. Just doing the first step will give us a lot of "no" answers where the corresponding 3DM works just fine - as such, we need to add fake people and fake skills. Find the vertex in 3DM that has the most edges touching it (= the skill held by the most people) and set that number of edges as x, then add $x - 2$ fake or 'dummy' people $d_1...d_{x-2}$ (who are going to end up on team A) and an additional, single fake or 'dummy' person (d_β, or the beta dummy, who is going to end up on team B. If $x - 2$ is -2, then x is 0, and the answer automatically shorts to "no", as no vertex is touched and so there's no 3D matching. If $x - 2$ is -1, then x is 1, meaning that all vertices have at most one edge touching them, and only the beta dummy d_β is added. From $x - 2 = 0$ and up, add $x - 2$ dummies $d_1...d_{x-2}$ and the beta dummy d_β as specified above).

3. Iterate over the skills from 1 to n, counting the number of people for each skill who possess it. If the skill is possessed by 0 people, it is an untouched vertex and as above the algorithm should return "no" - this will be implemented by giving the beta dummy d_β that skill, making it unmatchable by the FT solver and thus returning "no" for that instance. If the skill is possessed by 1 person, that real person will end up on team A and the beta dummy d_β (which will definitively end up on team B) should be given the skill as well, to match teams. If the skill is possessed by anywhere from 2 to x (the maximum number of people who possess a single skill, as defined earlier) people, note that only one of these real people/edges will be in the solution for the 3DM problem, and thus only one of these real people can end up on team A. All of the others will thus end up in team B, and (non-beta) dummies $d_1...d_{x-2}$ are needed to match them on team A. If some $y = 2$ to x number of people hold a skill, $y - 1$ of them will be on team B and so $y - 2$ (as $y - 2 + 1$ real person $= y - 1$) dummies must get this skill as well. If $y = 2$ this is 0, but that's fine because if two people are the only holders of a skill one naturally goes to A and the other to B. Recall that the number of non-beta dummies is $x - 2$, and that the maximum number required for skill balancing is $y - 2 = x - 2$.

4. The last step before actually running the FT algorithm is to create a guarantee that all the non-beta dummies $d_1...d_{x-2}$ end up on team A and that the beta dummy d_β ends up on team B. We do this by adding $x - 2$ dummy skills $s_1...s_{x-2}$. All real people are unskilled in every dummy skill, but each dummy is skilled in exactly one of them: d_1 is skilled in s_1, d_2 in s_2, so on and so forth to d_{x-2}'s ability to do s_{x-2}. The beta dummy d_β is skilled in all of them - thus, when the FT black box solver tries to make two teams with equivalently skilled populations, it must put all of the non-beta dummies in one team and the beta dummy in the other (and with the calculations from step 3, all of the non-beta dummies will be skilled in such a manner that only a single real person can be picked up into their team, which is exactly what we want).

5. Run the FT black box (finally). All of the non-beta dummies will end up in one team with a single real person for each skill, and the beta dummy will end up in another with all the other real people for that skill. If the FT algorithm returns "yes", there's a 3DM using the people/edges in team A. If it

returns no (indicating that satisfying the equality of one skill disrupts the equality of another), there isn't (corresponding to the case in which not all vertices can be reached using edges that don't overlap, preventing a 3DM).

Okay, that was pretty long-winded, but now we have a table we can talk about with terminology and a way to case-by-case show that whenever 3DM returns "yes", the corresponding FT table returns "yes", and vice versa. My multiple references above to vertices untouched by any edge are perhaps the easiest case to get started with - any vertex that isn't touched by any edge in a 3DM problem automatically nulls the entire problem, because there can't be a subset of the set of triples/edges that touches every vertex once if there's a vertex touched 0 times. 3DM would in this case thus return "no". The FT problem that corresponds has a column for a given skill with all 0s and no 1s (at least, for real skills involving real people) also has the beta dummy d_β get that skill, leaving the eventual FT solver with no real person to match against the beta dummy and thus returning a "no" as well.

Let's move on to the case where each vertex is touched at most one time. If there are any that are touched 0 times, both 3DM and the corresponding FT instance return "no", as we've seen, so what's really interesting here is the case in which each vertex is touched exactly once - which is to say, the three skills that each person has do not overlap at all. In this case, the corresponding FT instance just has a beta dummy with every skill, creating a perfect 1-to-1 match when the FT black box runs on the instance between each skill (held by a lone person) and the beta dummy, and the teams generated in the process are all the real people in A and the beta dummy in B, which is exactly the same answer as for the 3DM instance - each edge touches exactly three vertices, none overlap, and thus the solution on which a "yes" output is predicated is the full set of triples/edges, just like the solution for the FT on which its "yes" is predicated is the full set of people.

The more complex cases are those where the maximum number of people touching a vertex / the maximum amount of people who share a skill is $x \geq 2$, which require 0 or more non-beta dummies to balance the teams out, in addition to the beta dummy. In these cases there may be skills only held by one person, but including one of these skills means including all the others that the person holds - this can create a conflict. In the case that two people share a skill, only one of them can be in team A (as dummies are added to team A to balance out the other holders of the skill so as to correspond to a single edge on a vertex in the 3DM problem), but each of them could also be the only holders of separate skills. If this is the case (which is to say, the only edges touching two distinct vertices also share a vertex), one has to be "picked" and the other won't have a match, corresponding to a missed vertex in 3DM, resulting in a "no" for both. This is the only way that a 3DM instance can fail when all vertices are touched at least once - "edge interference" forces a choice between edges that are the sole contactors of certain vertices, breaking the matching. Barring this case, all successes of the FT build from a 3DM instance are also successes of the 3DM instance - for any given skill with y holders, if $y = 1$, the beta dummy receives the corresponding skill and the individual real holder of the skill ends up in team A opposite the beta dummy. If $y \geq 2$, the saturation of dummies on the A team ($y-2$ of them) allows one skillholder to be picked by the FT black box while the other $y-1$ end up in team B - this is what we want for the 3DM, as well - a vertex being touched by only one person. This is guaranteed by the fact that the dummy skills force all dummies into team A and the beta dummy into B - each dummy has a single dummy skill and the beta has all of them, forcing the non-beta dummies to operate as a bloc or a meta-dummy, if you will, which has skill values of $y-2$ for each skill held by at least two real people. Viewed like this, it becomes clear that the FT algorithm can only pick 1 and discards the $y-1$ others for each skill; that's the only way to result in $y-1$ on each team for any skill. If the FT algorithm returns "yes" in the case that $x \geq 2$, then, it's because there's a set of people/edges that can be matched with an appropriate amount of dummies while avoiding the pitfall of interfering people/edges - which is only possible if the edges connecting the vertices in the 3DM problem have a subset that touches each vertex once. The reverse is true, as well - a "yes" in 3DM is a "yes" in FT because some subset of the edges is a 3-dimensional perfect matching, meaning that the people who will constitute team A in the corresponding FT are the same subset - all of the other edges/people end up in team B, balancing out the non-beta dummies.

This indicates that a "yes" is a "yes" in both directions, in all cases, and as such through a polynomial algorithm (n to add all vertices to the table as skills, m to do the same with all edges as people, another n-ish to find the vertex in the most edges, and then a series of n sums of m 0s or 1s, followed by the addition of dummy people and skills, correlated to the number of edges, and then the black box) 3DM is reducible to FT or $3DM \leq_P FT$. This indicates that FT is NP-hard; this fact in conjunction with the fact that FT is in NP means that FT is NP-complete.