Balanced Committee Election

June 12, 2018

In a balanced committee election, we have m candidates $C = \{1, \ldots, m\}$, n voters and a desired size $k \in \{1, \ldots, m\}$ of a committee to be elected. Each voter can select at most k candidates. This gives each candidate i a total number of votes w_i. A committee $S \subseteq C$ of size k is considered to be good if it receives many votes, i.e., if the sum of all of the votes for the candidates in S, $\sum_{i \in S} w_i$, is large. At the same time, we want to guarantee that the selected committee S is "balanced" according to important features such as gender and locality. For example, we might want S to satisfy the following two criteria:

(1) 50% male and 50% female;

(2) 80% candidates from cities and 20% candidates from the countryside.

Overall, our goal is to elect a committee $S \subseteq C$ of size k that satisfies the above criteria. Furthermore, amongst all possible committees that do so, we would like to elect the one with the most votes, i.e., with maximal $\sum_{i \in S} w_i$.

We can encode this problem mathematically by using a 0/1 variable x_i to represent whether the candidate i is selected ($x_i = 1$) or not ($x_i = 0$). To give an example, suppose that $k = 10$. Then, the requirement that the committee S is of size 10 is equivalent to saying that we must satisfy

$$\sum_{i \in C} x_i = k,$$

i.e., exactly 10 candidates must have $x_i = 1$. Suppose $P_m \subseteq C$ represents the collection of male candidates and $P_f \subseteq C$ the collection of female candidates. The gender criteria (1) is equivalent to

$$\sum_{i \in P_m} x_i = 5, \quad \text{and} \quad \sum_{i \in P_f} x_i = 5,$$

i.e., exactly 5 male candidates must have $x_i = 1$. Similarly, suppose $P_c \subseteq C$ represents the collection of candidates living in the city and $P_t \subseteq C$ the collection of candidates living in the countryside. The locality criteria (2) is equivalent to

$$\sum_{i \in P_c} x_i = 8, \quad \text{and} \quad \sum_{i \in P_t} x_i = 2.$$

If candidate i is selected, then it contributes $w_i = w_i \cdot x_i$ votes to the committee (as $x_i = 1$). Otherwise, it contributes $0 = w_i \cdot x_i$ votes (as $x_i = 0$). Thus, the total votes received by the selected committee captured by the equation

$$\sum_{i \in C} w_i \cdot x_i.$$

Using the above formulation, we can write down an "integer linear program" or ILP to encode the mathematical problem of finding the best committee that satisfies this criteria as follows:

$$\begin{aligned}
\text{maximize } & \sum_{i \in C} w_i \cdot x_i && \text{(maximize the total votes)} \\
\text{subject to } & x_i \in \{0,1\} \quad \text{for all } i \in C, && \text{(0/1 variables)} \\
& \sum_{i \in C} x_i = 10, && \text{(committee size constraint)} \\
& \sum_{i \in P_m} x_i = 5, \sum_{i \in P_w} x_i = 5, && \text{(gender constraints)} \\
& \sum_{i \in P_c} x_i = 8, \sum_{i \in P_t} x_i = 2. && \text{(locality constraints)}
\end{aligned} \tag{1}$$

In general, solving an integer linear program is NP-hard and hence can take a prohibitively long time. However, we can use the well-known and publicly avaiable package **CPLEX** which uses a "branch-and-cut" approach to speed up the computation while still ensuring that we get the optimal solution – this allows us to produce a result within seconds, even when there are 100s of candidates.

We now give a brief introduction to the branch-and-cut techniques.[1] We call a solution (i.e., a given output of x_is) "feasible" if it satisfies all of the constraints defined above. The first step is to "relax" the 0/1 variable constraints, so that now, instead of $x_i \in \{0,1\}$, we simply have $x_i \in [0,1]$, i.e., x_i can be any real number between 0 and 1, e.g., $x_1 = .5$. The ILP (1) above with this relaxed constraint is called a "linear program" LP. Such linear programs are useful because they can be solved very quickly via standard techniques, e.g., the simplex algorithm or the interior point method. The problem is that the optimal solution of LP might not be 0/1, which means we can no longer interpret the results into a winning committee – if candidate i has $x_i = .5$, it is not clear if they should end up in the winning committee or not. Still, working with this LP is useful for the branch-and-cut method, which, on a high level, boils down to two combined techniques "branch-and-bound" and "cutting plane".

The branch-and-bound method first computes an optimal solution $x = (x_1, \ldots, x_m) \in [0,1]^m$ to the LP. If x is integral, then x must also be an optimal solution of the ILP (1), and hence we need not proceed further. Otherwise, there must be at least some candidate i for which x_i is neither 0 nor 1. E.g., if $x_1 = 0.5$ we would create two branches: One is a new linear program LP_0 that is a copy of the current LP but additionally sets $x_1 = 0$. Similarly, the other branch is a linear program LP_1 which is a copy of the current LP but additionally sets $x_1 = 1$. Any feasible solution of ILP (1), which must be integral by definition, must also be a feasible integral solution of either LP_0 or LP_1 as x_i must be set to either 0 or 1. Hence, if we know the the optimal integral solutions of LP_0 and LP_1, whichever one is better would give us the optimal solution to the original ILP. Thus, we have reduced the ILP program to two "smaller" programs (because now x_i is fixed, so there is one fewer variable to optimize over). We could recurse and continue branching LP_0 and LP_1 in this manner, but that would quickly blow up computationally, requiring 2^m branches, and could take years to solve even on a supercomputer. To attempt to sidestep this issue, we can make some observations that can help us "prune" some branches so that we only follow the paths that have some chance of leading to an optimal integral solution, and ignore paths which we can already rule out as sub-optimal.

[1] For more details, we refer interested readers to read an explanation by the IBM knowledge center (`https://www.ibm.com/support/knowledgecenter/SSSA5P_12.6.2/ilog.odms.cplex.help/refcppcplex/html/branch.html`) or watch a related video (`https://www.youtube.com/watch?v=jgQhzl3djM8`)

To give a concrete example of how this could work, consider the following. We first construct *any* feasible integral solution $y = (y_1, \ldots, y_m) \in \{0,1\}^m$ to the original ILP (note that y is also a feasible solution the relaxed LP). By definition, we know that the optimal value of the ILP (1) is at least the value attained by y, call this value ℓ, i.e.,

$$\ell = \sum_{i \in C} w_i \cdot y_i.$$

Now, consider LP_0. Similar to LP, we can compute an optimal solution $x^0 = (0, x_2^0, \ldots, x_m^0) \in [0,1]^m$ and a feasible integral solution $y^0 = (0, y_2^0, \ldots, y_m^0) \in \{0,1\}^m$ of LP_0. The optimal integral solution of LP_0 must have a value at least

$$\ell_0 = \sum_{i \in C} w_i \cdot y_i^0,$$

and in fact it is at most

$$u_0 = \sum_{i \in C} w_i \cdot x_i^0.$$

If $u_0 < \ell$, i.e., the optimal value of LP_0 is less than the global lower bound of Program (1), it implies that any feasible integral solution of LP_0 is not the optimal solution of Program (1). Thus, we safely ignore LP_0 in this case, and only consider LP_1. Otherwise, we continue branching LP_0 by additionally setting $x_2 = 0$ or $x_2 = 1$. In fact, since y^0 is also a feasible solution of Program (1), if $\ell_0 > \ell$, we can update the global lower bound $\ell \leftarrow \ell_0$; this allows us to prune even more branches and further speed up the computation.

In addition to the above branching procedure, at each step of solving an LP, a "cutting plane method" is introduced which can help further prune branches. In effect, the goal is to carefully add constraints to the LP, and hence "cutting away" part of the feasible space *without* cutting any part of the ILP.[2] This can make the optimal value u_0 of LP_0 smaller (without reducing the optimal value of the ILP), and hence increases the opportunity that $u_0 < \ell$ – the condition that we can safely ignore the branch LP_0. As an example, first consider the following constraints involving three 0/1 variables:

$$x_1 + x_2 \leq 1, \quad x_1 + x_3 \leq 1, \quad x_2 + x_3 \leq 1.$$

This can be strengthened by adding the following constraint:

$$x_1 + x_2 + x_3 \leq 1.$$

No feasible 0/1 solutions are ruled out by this extra constraint, but some fractional solutions, e.g., (0.5, 0.5, 0.5), which satisfy the first set of inequalities are now ruled out by the additional constraint.

By combining the branching, pruning, and cutting, the problem can be solved quickly. We provide an example implementation that is publicly available here: `https://github.com/huanglx12/Balanced-Committee-Election`.

[2]There is a simple way to find such constraints called the Gomory's method.