1 Introduction

This lecture is again about energy minimization. Jon wrote a nice paper about it [1]. The paper uses a very powerful technique used in approximatio algorithms called LP rounding. Assuming many CS664 students are not familiar with it, he spent most of the lecture explaining what a linear program is and how can one use it to find approximate solutions to hard optimization problems. The last third of the lecture gives a sketch of how the LP rounding algorithm for energy minimization in the Pots model works.[1]

2 What is a LP

As an example, consider the following two inequalities:

$$\begin{aligned} x_1 + x_2 &\leq 1 \\ x_2 &\leq 2 \end{aligned}$$

each inequality corresponds to a halfplane. Since we want both inequalities to be satisfied, we are interested in the intersection of the two halfplanes. In two dimensions (i.e. with two variables), an LP is just that – intersection of a bunch of halfplanes. In higher dimensions:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &\geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &\geq b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &\geq b_m. \end{aligned}$$

Each inequality corresponds to a n dimensional halfspace, and we are interested in the intersection of all m halfspaces. Our goal is to find a point (n-dimensional vector) x that satisfies all inequalities and minimizes (or maximizes) some linear objective function $c_1x_1+c_2x_2+\ldots+c_nx_n$.

Linear programs were probably invented in the forties in military logistics. A typical problem of that sort might look like this: given n types of food and m nutritients, for each type of food i we are given the amount a_{ij} of nutritient j it contains. Each type of food i has unit price c_i. We require that a soldier gets at least b_j of each nutritient j. Subject to these constraints, we want to find a set of foods (and their amounts) to buy to minimize their total price.

Note. Linear programs can contain both types of inequalities: $\leq$ and $\geq$, as well as equalities. It can always be converted to a normal form, containing, for example, only $\geq$: For example, we can replace the inequality $a_1x_1 + a_2x_2 \leq b$ by $-a_1x_1 - a_2x_2 \geq -b$. An equality $a_1x_1 + a_2x_2 = b$ can be replaced by $a_1x_1 + a_2x_2 \geq b$ and $a_1x_1 + a_2x_2 \leq b$. Linear programs can be maximization or minimization; we can easily switch between the two. For example, maximizing $c_1x_1 + c_2x_2$ is the same as minimizing $-c_1x_1 - c_2x_2$.

[1] The paper contains a more general result for arbitrary metric on labels.

2.1 How does one solve LP's?

First let's get some intuition. Fig. 1 shows a region that is an intersection of five halfplanes. In this region, we want to minimize some objective function, say $x_1 + x_2$. We can do this by drawing a line $x_1 + x_2 = K$ for some big constant K, and then moving the line "down" by decreasing K. We stop at the last point where the line has an intersection with the region. The value of the constant K for which this happens is the minimum of the objective function. Another way to wiew this is to imagine our 2d polygon tilted in 3d, so that if we drop a ball on it, it will roll down until it stops at the minimum point.

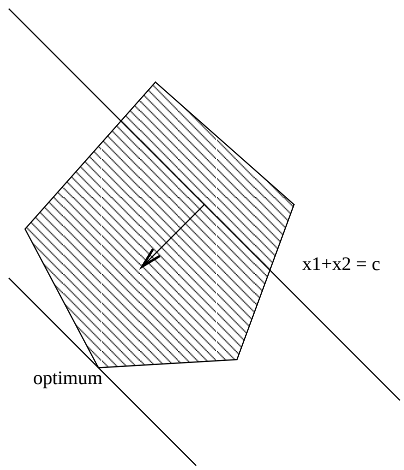

Figure 1: Solving a LP

Today's best LP solvers are able to handle problems with 10^4 to 10^5 variables and constraints. It is still not enough to process a full sized image (with 10^6 pixels, each of them can have 10^2 possible labels). The most widely used methods today are

1. Simplex method. Invented by Dantzig in the 1940s. It is basically a gradient descent on the edges of the polytope. This method is the working horse, most widely used in practice. All known variants may require exponential time on pathological instances, although are pretty fast on typical problems.

2. Ellipsoid method. Invented by Khachyian (1978). The first known poynomial time method. Too slow to be practical, has numerical problems, but has nice properties for theoreticians.

3. Interior point method of Karmarkar (1984). Often competitive with simplex in speed (not always).

3 Integer programming

Integer linear programming is a generalization of linear programming. An integer program (IP) also has a set of linear inequalities and a objective function to be maximized or minimized. It also contains integrality constraints: for some variables x_i we may require that they can only have integer values. A pure integer program is an integer program in which all variables are required to be integer.

Integer programs have a lot of expressive power, but are also much harder to solve. An intuitive reason is, that our previous approaches, "moving the line" and "rolling the ball" don't work. This is because they find a corner of the polytope, which may not be an integral point. Hence, we need to work back, and find an integer point that is "close enough" to the optimum. However, there may be many close integer points and there is no way to choose the best.

A more rigorous reason why integer programming is hard is that many NP complete problems can be naturally expressed as integer programs.

Theorem 1 *Integer programming is NP complete.*

Proving that a problem is in NP is usually the easy part. It suffices to exhibit a solution (an integer vector satisfying all inequalities with small cost). However, it is not a priori clear that if the IP has a solution, it has one of polynomial size (i.e. number of bits). We won't go into the details, just note that if the IP has a solution, then it has one of polynomial size.

For proving IP to be NP hard, we chose VERTEX COVER as a running example. Given an undirected graph $G = (V, E)$ as an instance of VERTEX COVER, we construct an integer program as follows. For each vertex $v \in V$, we create a variable x_v and we require each x_v to have integer values. We want x_v have only 0-1 values, meaning that $x_v = 1$ if v is in the vertex cover S and $x_v = 0$ otherwise. We achieve this by inequalities $0 \leq x_v \leq 1$ for every $v \in V$. We want that at least one endpoint of each edge (uv) to be selected; we achieve this by the inequality $x_u + x_v \geq 1$.

$$\text{minimize} \sum_{v \in V} x_v$$

$$\begin{aligned} x_v &\geq 0 & & \\ x_v &\leq 1 & & \forall v \in V \\ x_u + x_v &\geq 1 & & \forall e = (u, v) \in E \end{aligned}$$

Clearly, each vertex cover corresponds to a solution to the IP and vice versa, the cost of the IP is equal to the cost of the vertex cover. Hence VERTEX COVER can be soved using integer linear programming.

Note. Just by replacing each inequality $x_u + x_v \geq 1$ by $x_u + x_v \leq 1$ and minimizaition by maximization, one gets an integer program for INDEPENDENT SET. This shows the great expressive power of IP's.

4 LP rounding

Integer programming is NP hard, so one cannot hope for exact solutions. But sometimes, we can find approximate solutions that are good enough. A powerful technique for finding approximate solutions to integer linear programs is *LP rounding.* The idea is: take an IP, ignore all integrality constraints. What we get is a linear program, which can be solved efficiently. Find an optimal solution x to this LP. If x happens to be integral, it is obviously also an optimal solution to the IP. If not, try to round the coordinates of x to obtain a new solution $\overline{x}$. Of course we need to make sure that $\overline{x}$ still satisfies all inequalities. The hope is that $\overline{x}$ will not be much more expensive than x, and hence not much more expensive than the optimal IP solution.

As our running example, consider the IP for VERTEX COVER. Treat all variables as real numbers and find an optimal solution to the resulting LP. Fig. 2 shows a possible optimal solution to this LP.

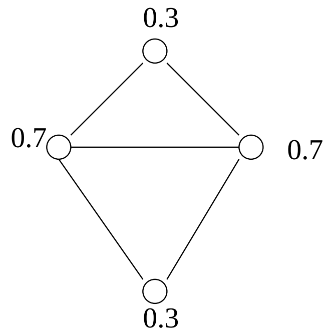

Figure 2: Solution to the VERTEX COVER LP.

Let us denote the optimal solution by x^*. Consider the following rounding rule. For each $v \in V$, if $x^* \geq 1/2$, let $\overline{x}_v = 1$ and if $x^* < 1/2$, let $\overline{x}_v = 0$. We claim that $\overline{x}$ satisfies all inequalities of the LP. Each $\overline{x}_v$ is trivially between 0 and 1. Consider an edge $e = (u, v)$. Since x^* was feasible, we know that $x_u^* + x_v^* \geq 1$. Hence one of x_u^*, x_v^* must be at least $1/2$. This means that at least one of $\overline{x}_u$, $\overline{x}_v$ must be 1, and hence $\overline{x}_u + \overline{x}_v \geq 1$ holds. This means that $\overline{x}$ is a feasible solution of the LP, and since integer, also a feasible solution of the IP.

We claim that the cost of $\overline{x}$ is at most twice the cost of the optimum solution of the IP. We show it in two steps. First, the optimal solution to the IP is also a solution to the LP; hence the best LP solution x^* can be only better. Second, we claim that $\sum_{v \in V} \overline{x}_v \leq 2 \sum_{v \in V} x_v^*$. This is easy, since for each v, $\overline{x}_v \leq x_v^*$.

5 Graph labeling

Finally, we get to the main point of the lecture, which is a 2-approximation algorithm for our beloved graph labeling problem. (Once again) we are given an undirectef graph $G = (P, N)$, where P is the set of vertices (known also as pixels), and the set of edges N, expressing the neighborhood relation between vertices. We also have a finite set L of labels. For each pixel p we incur cost $c_p(a)$ for assigning it label a. We also incur cost w_{pq} for every pair of pixels p, q such that $pq \in N$ and p and q are assigned different labels. Thus, our goal is to assign each pixel $p \in P$ its label $f(p) \in L$ so that the energy function

$$E(f) = \sum_{p \in P} C_p(f(p)) + \sum_{pq \in N} w_{pq} T(f(p), f(q))$$

is minimized.

Let's try to write an IP for this problem. The crucial question is, what shall be our variables. The first shot would be assume that the set of labels is $1, \ldots, k$ and have a variable x_p for each pixel that has value a iff the pixel p has label a. However, it would be very hard to express the energy function as a linear function of these variables. Instead, we will have a variable x_{pa} for every pixel p and every label a. The idea is, that $x_{pa} = 1$ if p has label a and $x_{pa} = 0$ otherwise. Clearly we require that $\sum_{a \in L} x_{pa} = 1$ for every p.

The IP might look like this:
minimize

$$\sum_{p \in P} \sum_{a \in L} c_p(a) x_{pa} + \frac{1}{2} \sum_{pq \in N} \sum_{a \in L} |x_{pa} - x_{qa}|$$

subject to

$$\forall p \in P \qquad \sum_{a \in L} x_{pa} = 1$$

This looks almost ok, except that we use absolute values, which is not allowed in an IP. For each term with absolute value, we create a variable Z_{pqa} and add inequalities $Z_{pqa} \geq x_{pa} - x_{qa}$ and $Z_{pqa} \geq x_{qa} - x_{pa}$. Thus the final form of our IP is the following.

minimize

$$\sum_{p \in P} \sum_{a \in L} c_p(a) x_{pa} + \frac{1}{2} \sum_{pq \in N} \sum_{a \in L} Z_{pqa}$$

subject to

$$\begin{array}{rrcl}
\forall p \in P & \sum_{a \in L} x_{pa} & = & 1 \\
\forall pq \in N, a \in L & Z_{pqa} & \geq & x_{pa} - x_{qa} \\
\forall pq \in N, a \in L & Z_{pqa} & \geq & x_{qa} - x_{pa} \\
\forall p \in P, a \in L & x_{pa} & \geq & 0
\end{array}$$

Dropping integrality constraints, we obtain a vector labeling problem: each pixel is assigned a vector of $|L|$ numbers: each label can be assigned fractionally, so that the fractions sum up to 1. We can find an optimal vector labeling problem using linear programming.

Now we want to round all variables so that

- Probability of assigning a label a to pixel p is x_{pa}. We need this in order to make sure that the rounded solution doesn't pay more for the data term than the LP solution
- The number of edges cut is minimized.

The first condition can be satisfied by independently choosing, for each pixel p its label a with probability p_a. The second condition can is best satisfied if all pixels have the same label. We need something in between. This motivates the following algorithm:

1. choose a label $a \in L$ at random and uniformly at random choose a number y from the interval $[0, 1]$.
2. Look at all pixels p that have no label assigned, such that $x_{pa} \geq y$. Assign all those pixels the label a
3. repeat until all pixels have been assigned a label.

For proof that this algorithm produces output whose cost is within a factor of 2 from the optimal solution, read [1].

References

[1] J. Kleinberg and É. Tardos, "Approximation Algorithms for Classification Problems with Pairwise Relationships: Metric Partitioning and Markov Random Fields," *Proceedings of the 40th Annual IEEE Symposium on the Foundations of Computer Science*, (1999).