

CS599: Structure and Dynamics of Networked Information (Spring 2005)
02/16/2005: Uniform Labeling Problem
Scribes: Yuriy Brun and Nupur Kothari

1 Uniform Labeling Problem via LP Approximation

In the last lecture, we started studying the uniform labeling problem, where labels are to be assigned to nodes minimizing the combination of two cost objectives: a *labeling cost* for assigning a particular label at a particular node, and a *separation cost* for having different labels at adjacent nodes. The problem is known to be NP-hard for 3 or more labels, so we are looking for approximation algorithms. The first approach we will pursue is to formulate the problem as an integer LP, drop the integrality constraint, and try to round the fractional solution to an integer solution not much worse. Last time, we derived the following LP, used in [1].

$$\begin{array}{ll}
 \text{Minimize} & \sum_{v,a} x_{v,a} c(v,a) + \sum_e w_e \cdot y_e \\
 \text{Subject to} & \sum_a x_{v,a} = 1 \quad \text{for all } v, a \\
 & y_{u,v} = \frac{1}{2} \sum_a |x_{v,a} - x_{u,a}| \quad \text{for all } (u,v) \in E \\
 & x_{v,a} \in \{0,1\} \quad \text{for all } v, a.
 \end{array}$$

Here, $x_{va} = \begin{cases} 1, & \text{if } v \text{ is labeled } a \\ 0, & \text{otherwise} \end{cases}$, and $y_{u,v} = \begin{cases} 1, & \text{if } u \text{ and } v \text{ have different labels} \\ 0, & \text{otherwise.} \end{cases}$. The constraint on $y_{u,v}$ is not itself linear, but we saw in the previous class how to express it in terms of linear constraints and additional variables.

As usual, we relax the constraint $x_{va} \in \{0,1\}$ to $x_{va} \in [0,1]$, i.e., we allow the solutions to take on fractional values. Then, we can solve the LP in polynomial time. We will investigate how to round the solution to obtain a labeling which does not perform much worse than the fractional optimum. In particular, that will show that it is not much worse than the integral optimum as well.

For the underlying idea, notice that the x_{va} , summed over all a , add up to 1 for each v . Hence, we can view them as fractions to which v is given label a , or as probabilities of v having label a . A first approach would label each node v with a with probability x_{va} . Notice that we would do very well on the labeling cost $\sum_{v,a} x_{va} c(v,a)$ this way, as

$$E[\text{labeling cost of node } v] = \sum_a \text{Prob}[v \leftarrow a] c(v,a) = \sum_a x_{va} c(v,a),$$

Here (and later), we denote by $v \leftarrow a$ the fact that node v is assigned label a , and use the fact that $\text{Prob}[v \leftarrow a] = x_{va}$.

However, this does not ensure that we do well on separation costs, too. In fact, it can happen that we don't. Suppose that the graph consists of just two vertices, v_1 and v_2 , with a non-zero cost edge between them. There are two labels a and b , and all vertex labeling costs are $c \equiv 0$. Then, the fractional solution can choose any fractional assignment, so long as v_1 and v_2 have the same fractional share of a . One of the optimal solutions of the LP, which we may have to round, would thus be assigning all probabilities equal to $1/2$. The LP cost is then 0. But if v_1 and v_2 receive different labels, we pay a non-zero separation cost, and thus have infinitely bad approximation.

Thus, we need to make more coordinated choices between labels of nodes and their neighbors. Suppose that we label a node v with the label a . A first stab would be to label all neighbors u of a with $a_{ua} \geq x_{va}$ with a as well. But then, u 's label may conflict with that of one of its neighbors when it is labeled later. So we may also want to require that a be the label with the largest fractional value among u 's labels. But that

does not really solve the problem either. It seems that we may also have to label the neighbors of u , and so on recursively.

As a result, one thing we can do is that once we pick a node v and assign a to it, we pick x_{va} as a threshold, and assign a to *all* nodes u having $x_{ua} \geq x_{va}$. Then, when a conflict occurs between some such u and another node w which gets a different label b , at least we know that they didn't have exactly identical fractional values.

This motivates the following rounding algorithm, which picks a label a uniformly at random, as well as a random threshold, and labels all nodes with that particular label if their probability for the chosen label is above the threshold. This process is repeated until there are no more nodes left to be labeled.

Algorithm 1 LP Rounding algorithm

- 1: **repeat**
 - 2: Pick a random label $a \in L$, (let $k = |L|$) and a random $\alpha \in [0, 1]$.
 - 3: Label all v with $x_{va} \geq \alpha$ with a , and remove them.
 - 4: **until** no more nodes are left
-

Theorem 1 *Algorithm 1 is a 2-approximation.*

Proof. We prove the theorem by analyzing the assignment cost and separation cost separately. We show that for both cases, the cost incurred using our approximation algorithm is within a factor of two of the optimum fractional solution of the LP, and hence also within the same factor of the integral solution (which can be no better than the best fractional one).

1. To analyze the assignment cost, we first notice that in any particular iteration, v is labeled a with probability $\frac{x_{va}}{k}$. That is because label a is picked with probability $1/k$, and conditioned on picking label a we label v iff the randomly threshold is at most x_{va} .

Because in each iteration, label a is assigned to v with probability proportional to x_{va} , the overall probability of assigning a to v is x_{va} . We can thus use the same argument as above to obtain that the expected labeling cost is $\sum_a \text{Prob}[v \leftarrow a]c(v, a) = \sum_a x_{va}c_{va}$.

2. To analyze the separation cost, we first notice that we only incur cost $w_{u,v}$ for the edge $e = (u, v)$, when u and v get different labels. But that can only happen when u, v are labeled in different iterations of the algorithm. Therefore, there must have been an iteration labeling exactly one of $\{u, v\}$. We bound the probability of that happening.

Given a label a , exactly one of $\{u, v\}$ is labeled a iff the threshold $\alpha \in (x_{ua}, x_{va}]$ (assuming $x_{ua} < x_{va}$). So the probability of labeling exactly one of u and v with label a is $|x_{va} - x_{ua}|$. Summing over all labels a , the probability of labeling exactly one of $\{u, v\}$ with any label is $\sum_a \frac{|x_{ua} - x_{va}|}{k}$.

For any time t , we let F_t be the event that at least one of $\{u, v\}$ is labeled in iteration t and E_t the event that exactly one is labeled. We are thus interested in $\text{Prob}[E_t | F_t]$. First off, notice that $E_t \subseteq F_t$, so $\text{Prob}[E_t] = \text{Prob}[E_t \cap F_t] = \text{Prob}[E_t | F_t] \cdot \text{Prob}[F_t]$. Rearranging yields that $\text{Prob}[E_t | F_t] = \frac{\text{Prob}[E_t]}{\text{Prob}[F_t]}$. But we already calculated $\text{Prob}[E_t]$ above. And the probability of F_t , the event that at least one of u, v is assigned a label, is at most $1/k$, the probability that u is assigned a label. Hence, we have that

$$\text{Prob}[E_t | F_t] \leq \frac{\sum_a \frac{1}{k} |x_{ua} - x_{va}|}{\frac{1}{k}} = \sum_a |x_{ua} - x_{va}| = 2y_{u,v}.$$

Applying this at the time t where the first of u, v is actually assigned a label gives us that they are separated with probability at most $2y_{u,v}$. (Notice that t is a random variable, but this does not really hurt us here.)

Hence, the expected total separation cost is

$$\text{E}[\text{separation cost}] = \sum_{e=(u,v)} \text{Prob}[e \text{ separated}] \cdot w_e \leq \sum_{e=(u,v)} 2y_{u,v}w_e.$$

Summing up over the two cases, the total cost is at most

$$\sum_{v,a} x_{va} c_{va} + 2 \sum_{e=(u,v)} w_e y_{u,v} \leq 2 \text{cost}(\text{LP-OPT}).$$

Thus, the algorithm is a 2-approximation. ■

Having obtained a 2-approximation, we naturally want to know if we can do better. While this question has not been entirely resolved, we will show here that we cannot do better than a 2-approximation if the bound on the optimum uses only the LP solution. We exhibit an integrality gap of 2, i.e., we show that there are instances where the best integral solution is worse than the best fractional one by a factor arbitrarily close to 2.

The example consists of a complete graph K_n with edge weights of 1 for each edge. There are n labels, one corresponding to each vertex. The labeling cost is $c(v, "v") = \infty$, and $c(v, "u") = 0$ for all nodes $u \neq v$. That is, no node can be labeled with its own name.

The best integral solution assigns each node label "1", except for node 1, which is given label "2". Then, exactly the edges incident with node 1 are cut, so the total cost is $n - 1$.

On the other hand, a fractional solution can assign $x_{va} = \begin{cases} 0, & \text{if } a = v \\ \frac{1}{n-1}, & \text{if } a \neq v. \end{cases}$ Then, the cost incurred by each edge (u, v) is $y_{u,v} = \frac{1}{n-1}$, so the total cost is $\frac{1}{n-1} \binom{n}{2} = \frac{n}{2}$. Hence, the integrality ratio is $\frac{n-1}{\frac{n}{2}}$. As $n \rightarrow \infty$, this integrality ratio approaches 2.

2 Uniform Labeling Problem via Local Search

We can avoid having to solve an LP (which, while polynomial, tends to be fairly slow), by using a different 2-approximation algorithm based on Local Search where search moves use Min Cut computations [2]. In local search algorithms, we start with a solution, and repeatedly apply one of several simple moves, as long as it leads to an improvement. When no more improvement is possible with the simple moves, the algorithm terminates. It is rare that provable guarantees can be shown for efficient local search algorithms, but this is one of the cases.

Here, a local search move works as follows: We pick a label a and try if converting other vertices to that label will reduce the total cost. So we may label an arbitrary additional set of vertices with a , but no vertex gets its label changed to anything except a in one step. Among all such new labelings, we choose the best one, and then iterate over all labels. We thus obtain the following algorithm:

Algorithm 2 Local Search

- 1: Start with an arbitrary labeling f (e.g., the best individual vertex labeling $f(v) = \text{argmin}_a c(v, a)$).
 - 2: **repeat**
 - 3: **for** all labels a in turn (e.g., round robin) **do**
 - 4: Let f_a be the best assignment with $f_a(v) = \begin{cases} a, & \text{if } f(v) = a \\ a \text{ or } f(v), & \text{if } f(v) \neq a. \end{cases}$
 - 5: Update $f = f_a$.
 - 6: **end for**
 - 7: **until** no more improvement
-

Notice that in this algorithm, the labeling f_a can be found using a single Min-Cut computation.

For a given labeling f , we let $\gamma(f) = \sum_v c(v, f(v)) + \sum_{e=(u,v)} w_e [f(v) \neq f(u)]$ denote its total labeling cost. (We use Iverson's Convention here, writing $[a] = \begin{cases} 1 & \text{if } a \text{ is true;} \\ 0 & \text{otherwise.} \end{cases}$) The algorithm only chooses f_a over f if it is better, so whenever changes happen, γ is strictly decreasing. As a result, the algorithm cannot

loop. Further, if all w_e and $c(v, a)$ are integers, then the decrease will be at least 1 in each iteration, so there can be at most $\sum_e w_e$ iterations. Therefore, the algorithm runs in pseudopolynomial time.

Claim 2 *The algorithm produces a 2-approximation.*

Proof. The idea behind the proof is to show that the termination condition — the fact that no single-label relabeling yielded any more improvement — is enough to be close to the optimum solution. Roughly, we will do this by “decomposing” the optimum solution into our solution.

Specifically, let f be our solution and f^* the optimal solution. For each a , we let $S_a = \{v \mid f^*(v) = a\}$ be the set of all nodes that the optimum labeled a . We define a labeling f_a that “interpolates” between f and f^* , by saying that $f_a(v) = a$ if $f^*(v) = a$ and $f_a(v) = f(v)$ otherwise. That is, the interpolating labeling agrees with f^* whenever that chose a , and with our solution otherwise.

Because f_a was a candidate for relabeling from f , the termination of the algorithm implies that $\gamma(f) \leq \gamma(f_a)$ for all labels a . We now divide the cost of f into that incurred inside S_a , outside S_a and across the boundary. So we define

$$\gamma_S(f) := \sum_{v \in S} c(v, f(v)) + \sum_{(u,v) \in S \times S} w_{(u,v)} [f(u) \neq f(v)]$$

for any set S . As a result, we can rewrite $\gamma(f)$ and $\gamma(f_a)$ as

$$\begin{aligned} \gamma(f) &= \gamma_{S_a}(f) + \gamma_{\bar{S}_a}(f) + \sum_{(u,v) \in S_a \times \bar{S}_a} w_{(u,v)} [f(u) \neq f(v)], \\ \gamma(f_a) &= \gamma_{S_a}(f_a) + \gamma_{\bar{S}_a}(f_a) + \sum_{(u,v) \in S_a \times \bar{S}_a} w_{(u,v)} [f_a(u) \neq f_a(v)]. \end{aligned}$$

Above, we argued that $\gamma(f) \leq \gamma(f_a)$, and because the two labelings agree on the set \bar{S}_a , we have that $\gamma_{\bar{S}_a}(f) = \gamma_{\bar{S}_a}(f_a)$. On the other hand, f_a and f^* agree on S_a , so $\gamma_{S_a}(f_a) = \gamma_{S_a}(f^*)$. Taken together, this implies that

$$\gamma_{S_a}(f) + \sum_{(u,v) \in S_a \times \bar{S}_a} w_{(u,v)} [f(u) \neq f(v)] \leq \gamma_{S_a}(f^*) + \sum_{(u,v) \in S_a \times \bar{S}_a} w_{(u,v)} [f_a(u) \neq f_a(v)].$$

But under the last sum, whenever $f_a(u) \neq f_a(v)$, then also $f^*(u) = a \neq f^*(v)$, so we can upper bound the sum by replacing $f_a(u)$ and $f_a(v)$ with $f^*(u)$ and $f^*(v)$. Thus, we have derived

$$\gamma_{S_a}(f) + \sum_{(u,v) \in S_a \times \bar{S}_a} w_{(u,v)} [f(u) \neq f(v)] \leq \gamma_{S_a}(f^*) + \sum_{(u,v) \in S_a \times \bar{S}_a} w_{(u,v)} [f^*(u) \neq f^*(v)].$$

Because the S_a for all a form a disjoint cover of all nodes, summing up over all a now gives us that

$$\begin{aligned} \gamma(f) &\leq \sum_a \gamma_{S_a}(f) + \sum_a \sum_{(u,v) \in S_a \times \bar{S}_a} w_{(u,v)} [f(u) \neq f(v)] \\ &\leq \sum_a \gamma_{S_a}(f^*) + \sum_a \sum_{(u,v) \in S_a \times \bar{S}_a} w_{(u,v)} [f^*(u) \neq f^*(v)] \\ &\leq 2\gamma(f^*), \end{aligned}$$

because in the last sum, each edge between differently labeled nodes is counted exactly twice, while the assignment cost for each node is counted exactly once. So the algorithm is a 2-approximation. ■

References

- [1] J. Kleinberg and E. Tardos. Approximation algorithms for classification problems with pairwise relationships: Metric partitioning and markov random fields. In *Proc. 40th IEEE Symp. on Foundations of Computer Science*, 1999.
- [2] O. Veksler. *Efficient Graph-Based Energy Minimization Methods in Computer Vision*. PhD thesis, Cornell University, 1999.