LOCAL SEARCH

BASED ON LECTURES BY URI ZWICK AND HAIM KAPLAN

1. IDEA BEHIND SCHEME:

Remark: The document is written for maximization problems but the same thing works for minimization problems with the appropriate replacements (higher $\leftrightarrow$ lower etc.).

We want to approximate hard maximization problems. In order to do that we define for every configuration (=a solution, not necessarily the optimal solution) of the problem it's neighborhood (usually configurations which are almost the same). To run the algorithm we start from a random configuration and then find a configuration in the neighborhood whose value is higher and move to that configuration (sometimes we require to move to the best neighbor). If we reach a local maximum we stop and declare the last configuration as our approximation. We sometimes run the algorithm more than once from different starting configuration (usually random or a pertubation of the last result) and sometimes we start from a specific configuration. Another decision one needs to make before running the scheme is how to choose the neighborhood of a configuration. If the neighborhood is too large than the complexity of an iteration will be too high, and if the neighborhood is too small than many bad local optima might hurt our approximation.

2. ANALYZING COMPLEXITY AND QUALITY OF APPROXIMATION

2.1. **Indtroduction.** Because this is an approximation scheme, and because of the hardness of the problems (usually NP-Hard), we do not attempt to prove that we reached the optimal solution. Instead we attept to prove that our solution is not far from optimal (usually when we manage to prove something it is a multiplicative bound , i.e. we prove that are solution is at most C times the optimal for a constant C). Sometimes we do not manage to prove a bound on complexity, and sometimes we do not manage to prove that the approximation is of good quality, and the only thing that is known is that practically the algorithm works. But when we do, the proof usually looks the same and in the next section we will show the proof scheme.

2.2. **Proof Scheme Of The Quality Of Approximation.** Because of the nature of the approximation scheme, the only information we know is that we reached a local optima. Usually the way to utilize this information is to look at the the neighbors of the final configuration. We know that our configuration is a local optima and therefore the value of every neighbor is lower or equal to the value of our solution. After that we usually sum over the neighbors and if we look at the value the right way (for example in graphs we usually look at the information as a sum over the value for the edges) we can use the inequality we get to prove an inequality between our solution and the optimal one.

2.3. **Proof Scheme Of The Running Time Of The Scheme.** This is different than the proof of the quality of the approximation. On the one hand, proving the running time of every iteration is usually fairly easy and resembles proofs from the Undergraduate course about algorithms. On the other hand, finding a bound for the number of iteration is usually very hard, and for many important examples of local search it is not known (for some it is known that a better bound than an exponential one doesn't exist). We didn't see in this course a scheme to find a bound for the number of iteration (we did kind of see a scheme to limit the number of steps - if we only take steps which cause large improvments) , but a fairly simple one can be found when the values are integers and there are a lower and an upper bound for the value of a configuration (this is usually the case). In examples which satisfy the aforementioned conditions, a very simple bound can be found, namely the difference between the optimal value (usually a simple bound can be found that is better then the maximum/minimal value of a configuration) and the minimal/maximal bound of a value of a configuration.

3. Examples

3.1. **Max Cut.**

3.1.1. *Settings.* Let $G = (V, E)$ be a graph. Then a cut is a splitting of V into two distinct sets $S, \bar{S}$. We define the value of a cut to be $|\{(u, v) : u \in S, v \in \bar{S}\}|$ (we count every edge only once, and we call edges in that set crossing edges). The problem of finding a cut with a minimal value is a fairly easy one and is solved in the Undergraduate course with an algorithm by Dinic. The problem of finding a cut with a maximal value is a lot harder (It is actually NP-Hard). We will also show a case when the edges are weighted and then the value is the sum of weights of crossing edges.

3.1.2. *The Local Search Scheme.* We define the neighborhood of a cut $S, \bar{S}$ to be all cuts $S', \bar{S}'$ such that $\big||S| - |S'|\big| = 1$ (i.e. $S', \bar{S}'$ can be acheived from $S, \bar{S}$ by moving one vertex from one of the sets to the other). We start from a random cut and run the scheme

3.1.3. *Analysis Of The Quality Of Approximation.* If we define for every vertex v $\gamma(v)$ to be the set of edges incident to v that do not cross the cut, and we define $\delta(v)$ to be the set of edges incident to v that do cross the cut, Then because we reached a locally optimal cut we have that for every vertex v, $|\gamma(v)| \leq |\delta(v)|$ (otherwise transferring v will be an improving step). If we sum over the verices we have that $2|\{(u, v) : u, v \in S \vee u, v \in \bar{S}\}| \leq 2|\{(u, v) : u \in S \wedge v \in \bar{S}\}|$. Therefore, because the optimal solution cannot be larger than $|E|$, which is clearly equal to the number of the edges that cross our cut, plus the number of edges that do cross our cut, and since we proved that there are more edges that cross our cut, then edges that do not, we have that the value of our cut is at least $\frac{|E|}{2}$,and therefore, if we denote the value of an optimal solution by OPT , and our cut's value by L, than $2OPT \leq L$. This analysis also works with the weighted case if we replace $|E|$ by $W = \sum_e w(e)$.
In the weighted case we sometimes have a problem with the running time and then we take and improving step only if it improves the value by at least $\frac{2\varepsilon L}{|V|}$ with L being our current value and ε a constant value. If we analyze that we have that for a locally optimal value L, for every vertex v $w(\gamma(v)) \leq w(\delta(v)) + \frac{2\varepsilon L}{|V|}$ and then if we sum over the veryices we have that $2w(\{(u, v) : u, v \in S \vee u, v \in \bar{S}\}) \leq 2w(\{(u, v) : u \in S \wedge v \in \bar{S}\}) + 2\varepsilon L$ and since the left hand side is $2(W - L)$ and the right hand side is $2(1 + \varepsilon)L$ and $OPT \leq W$ we have that $OPT \leq (2 + \varepsilon)L$

3.1.4. *Analysis Of The Running Time.* In this analysis we will focus on analyzing the number of iterations. It is clear that if we are in the unweighted case than $|E|$ is a good bound since this example satisfies the conditions mentioned in the section about the analysis scheme. In the weighted case we have the same bound only with W instead of $|E|$ if all the weights are integers , and a bad example if we wish to have a good bound with respect to the size of the graph is in the slides, with the actual numbers and analysis in homework number 2. On the other hand, if the weights are not integers there are examples in which the scheme doesn't even halt. What we can do is what was mentioned in the previous subsubsection, i.e. to take an improving step only if it improves by at least $\frac{2\varepsilon w(S,\bar{S})}{n}$ for a constants ε and $n = |V|$. Then every step improves by a multiplicative factor of $1 + \frac{2\varepsilon}{n}$ and then the number of steps is bounded by $\log_{1+\frac{2\varepsilon}{n}} W = \frac{\log W}{\log(1+\frac{2\varepsilon}{n})} \overset{\log(x) \geq 1 - \frac{1}{x}}{\leq} (1 + \frac{n}{2\varepsilon}) \log W = O(\frac{n}{\varepsilon} \log W)$

3.1.5. *Practically.* The 2 factor is not impresive since there are simpler deterministic algorithms that get that factor, and since this is the expectation for a random cut. But still practically the Local Search is better. Trying to swap two verices or one every time instead of just one will get bigger cuts but will increase dramaticaly the running time, we see that the running time of every iteration is important.

3.2. **Min - Bisection.**

3.2.1. *Settings.* Let $G = (V, E)$ be a graph. Then we define a cut and a value of a cut the same way we did in the previous problem. Our goal is to find a minimum cut $(S, \bar{S})$ subject to the condition that $|S| = |\bar{S}|$ (i.e. S is a bisection). This is a NP-Hard problem as well.

3.2.2. *A Scheme By Lin and Kernighan.* Let $(S, \bar{S})$ be a bisection. We find a pair $x \in S, y \in \bar{S}$ such that if we switch them the value is the lowest of all the switches (can be larger than current value). After that we fix that couple after the switch and find the next best couple and iterate this until S and $\bar{S}$ are fully replaced. We define the neighborhood of $(S, \bar{S})$ to be all prefixes of that process (notice that the second best couple in the beggining might not be the best couple after we switched the best couple). If the difference between the value of the bisection before we switched the i'th couple and the bisection after is Δ_i than the values of the neighborhood are $\Delta_1, \Delta_1 + \Delta_2, ..., \sum_{i=1}^{n} \Delta_i$, and if one of them is negative than we have an improving neighbor.

3.2.3. *Practically.* There is no section with analysis because little is known about this scheme theoretically, but practically it is much better than the trivial local search (use single swaps and punish unbalanced cuts) and when compared to simulated anneling it is almost the same (it takes about 4 second and simulated annealing takes about 6 minutes, so for fair comparison we run this 90 times for every run of simulated annealing).

4. Computer graphics

4.1. **Image Segmantation.**

4.1.1. *Settings.* We have a grid of pixels, and we wnat to separate them to background and foreground. For each pixel v we have $c_b(v)$, the penalty we pay if we put v in the background, and $c_f(v)$, the penalty we pay if we put v in the foreground. For every neighboring pixels in the grid - v, u, we have a penalty $p(v, w)$ if we do not put them together. So we have a grid and we are searching for a cut (F, B) that will minimize $\sum_{v \in B} c_b(v) + \sum_{v \in F} c_f(v) + \sum_{v \in B, u \in F, (v,w) \in E} p(v, w)$.

4.1.2. *Solution.* This is not local search and this is actualy solved with flow. We make a graph of the grid and we add a source and a sink s, t and we we add to the edges $\{(s, u), (u, t) | u \in V \backslash \{s, t\}\}$. The capacity of an edge (u, v) is $p(u, v)$ if $\{u, v\} \cap \{s, t\} = \emptyset$, $c_b(v)$ if $u = s$ and $c_f(u)$ if $v = t$. We calculate a flow and use it to calculate a minimum cut and that cut is the desired cut.

4.2. **Multi Label Problem.**

4.2.1. *Settings.* We have a graph $G = (V, E)$ which represents a grid of pixels, and a small set of labels D. For every vertex v and for every label d we have $c_d(v)$ - the penalty if we give the label d to v, and for every two verices (v, u) with and edge beween them, we have $p(v, u)$ - the penalty we pay if v and u are assigned different labels (it does'nt matter which labels). We denote the assigment of labels as $f : V \to D$. We wish to minimize $\sum_{d \in D} \sum_{\{v | f(v) = d\}} c_d(v) + \sum_{\{(v,u) | f(v) \neq f(u)\}} p(v, u)$.

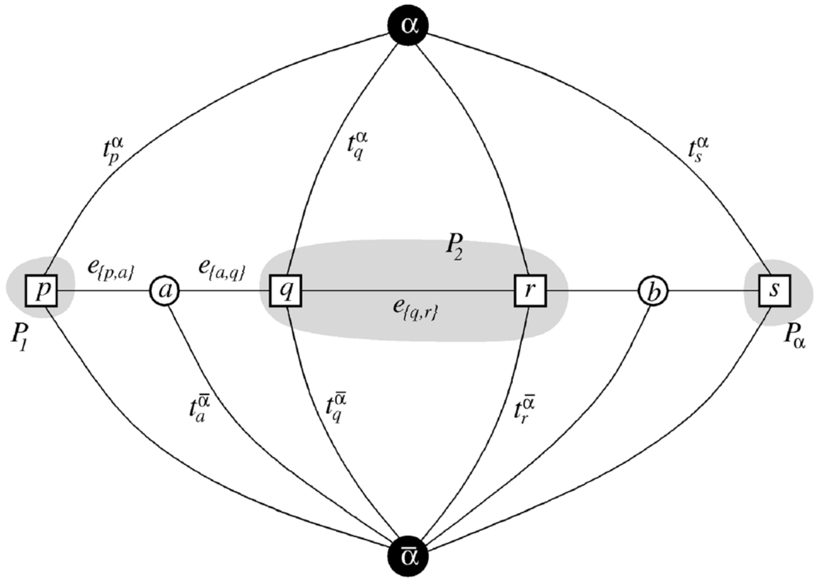
4.2.2. *The $\alpha\beta$ Swap Scheme.* For each pair of labels α, β we compute the best relabeling of vertices whose current label is α or β (relabeling with α and β), and perform the best relabeling. We can do that the same way we did in image segmantation. There is no known bound for the quality of the approximation, and in slide 52 there is a bad example for this scheme.

4.2.3. *The α Expantion Scheme.* For every label α, we find the best relabeling of vertices whose labels where not α, to α. Do this for all the labels and perform the relabeling that improves the most (of course we halt if we reached a local optima).

4.2.4. *Algorithm For Iteration Of α Expansion.* Lets say that the graph of pixels is $G = (V, E)$, and we have a labeling f. For every label α, define a graph $G_\alpha = (V_\alpha, E_\alpha)$,

$V_\alpha = \left\{\alpha, \bar{\alpha}, V\right\} \cup \left\{a_{pq} | (p,q) \in E, f(p) \neq f(q)\right\}$,

$E_\alpha = \left\{(\alpha, v), (v, \bar{\alpha})\right\} \cup \left\{(u,v) \in E | f(u) = f(v)\right\} \cup \left\{(p, a_{pq}), (q, a_{pq}), (\bar{\alpha}, a_{pq}) | (p,q) \in E, f(p) \neq f(q)\right\}$



and the weights are

edge	**weight**	**for**
$(u, \bar{\alpha})$	∞	$f(u) = \alpha$
$(u, \bar{\alpha})$	$c_{f(u)}(u)$	$f(u) \neq \alpha$
(α, u)	$c_\alpha(u)$	$u \in V$
(u, a_{uv})	$p(u,v)$	$(u,v) \in E, f(u) \neq f(v)$
(a_{vu}, v)	$p(u,v)$	
$(a_{uv}, \bar{\alpha})$	$p(u,v)$	
(u, v)	$p(u,v)$	$f(u) = f(v), (u,v) \in E$

This was not in such detail in the slides, maybe it's not for the test , DONT COUNT ON IT!!. If we find the minimum cut here we find the appropriate exapnsion. The analysis we did in class and in homework number 2 shows that $L \leq 2OPT$.