The transformation method

Russell Impagliazzo

January 29, 2004

1 The general method

Say that an algorithm is picking a solution one element at a time, and each element has a value. We want to compare the value of the element picked to the "opportunity cost" of other elements that could have been chosen instead. One way of visualizing this opportunity cost is to imagine an optimal solution being transformed into the algorithm's solution element by element. When we add the first element to the optimal solution, we might have to remove other elements or make other adjustments. The difference between the value of the added element and the costs of adjustments is the cost of adding the element. We need to show that cost is not too large compared to the value of the element.

More precisely, say that on a certain instance, the algorithm has t iterations, and each iteration picks a set of elements m_i in iteration i. So the algorithm's solution $S = m_1 \cap m_2 \cap ... m_t$, Assume we can write $Value(S) = \sum_{i=1}^{i=t} Value(m_i)$, e.g., if we are trying to maximize the sum of weights of all elements in S, $Value(m_i)$ would be the sum of weights of all elements in m_i. Let's also assume that there is no other solution but S that contains all the algorithm's moves, i.e., the solution is *maximal.* Formally, there is no legal solution S' with $S \subseteq S'$.

To use the transformation method (for maximization problems), we prove the following, for some $\alpha > 0$:

Lemma: Let S_i be any solution that contains the first i moves, $m_1 \subseteq S_i, m_2 \subseteq S_i, .., m_i \subseteq S_i$. Then there is a solution S_{i+1} with $m_1 \subseteq S_{i+1}, m_2 \subseteq S_{i+1}, .., m_{i+1} \subseteq S_{i+1}$ so that $Value(S_{i+1}) \geq Value(S_i) - \alpha Value(m_{i+1})$.

We then reason as follows: Let S_0 be an optimal solution. Apply the lemma to S_0 to get S_1 with $m_1 \subseteq S_1$, and $Value(S_1) \geq Value(S_0) - \alpha Value(m_1)$. Then apply it to S_1 to get S_2 with $m_1 \cup m_2 \subseteq S_2$ and $Value(S_2) \geq Value(S_1) - \alpha Value(m_2)$, and so on, until we finally reach S_t with $m_1 \cup m_2 .. \cup m_t \subseteq S_t$. Now since S_t contains our algorithm's solution S. $S_t = S$ by maximality. Furthermore, we have $Value(S) = Value(S_t) \geq Value(S_0) - \alpha(Value(m_1) + Value(m_2) + ... Value(m_t)) = Value(S_0) - \alpha Value(S)$. Thus, $Value(S) \geq \frac{Value(S_0)}{\alpha+1}$. Thus, our algorithm achieves approximation ratio $\rho = \alpha + 1$.

For a minimization problem, the lemma we prove is Lemma: Let S_i be any solution that contains the first i moves, $m_1 \subseteq S_i, m_2 \subseteq S_i, .., m_i \subseteq S_i$. Then there is a solution S_{i+1} with $m_1 \subseteq S_{i+1}, m_2 \subseteq S_{i+1}, .., m_{i+1} \subseteq S_{i+1}$ so that $Cost(S_{i+1}) \leq Cost(S_i) + \alpha Cost(m_{i+1})$

Here, we must have $\alpha < 1$. The conclusion is that $Cost(S) = Cost(S_t) \leq Cost(S_0) + \alpha Cost(S)$, so $Cost(S) \leq \frac{Cost(S_0)}{1-\alpha}$, so we achieve ratio $\rho = 1/1 - \alpha$.

This method is also used for exact algorithms, when we can use $\alpha = 0$.

2 Unit Cost Interval Scheduling

Let the unit cost interval scheduling problem be as follows:

Instance: A positive integer number of machines , m; A list of n jobs $Job_1, ..Job_n$, each with a *start time* s_i, and a *finish time* f_i with $s_i < f_i$.

Solution format Assign each job Job_i either a machine $1 \leq m_i \leq m$ or "Not done"

Constraint We can't assign too overlapping jobs to the same machine. I.e., if $s_i \leq s_j < f_i$, and Job_i and Job_j are both done, $m_i \neq m_j$.

Objective Maximize $Value(S)$, the number of jobs performed.

2.1 A greedy algorithm

In class, I gave two approximation algorithms for this problem, Smallest Duration and First Finish. I'll only give the better algorithm, First Finish, here. (Thanks, Sashka, for pointing this out.)

The First Finish algorithm is:

1. Order the jobs by f_i from smallest to largest.
2. For each job Job_i, if there is a machine with no previously scheduled conflicting jobs, schedule Job_i on the smallest (in index) such machine. Else, it is "Not Done".

Let the jobs performed by the above algorithm be numbered $i_1, ..i_t$, and the machines they are assigned to be $m_{i_1}, ..., m_{i_t}$.

Lemma: Let S_l be any schedule that assigns Job_{i_1} to $m_{i_1} ..., Job_{i_l}$ to m_{i_l}. Then there exists a schedule S_{l+1} that assigns Job_{i_1} to $m_{i_1} ... Job_{i_{l+1}}$ to $m_{i_{l+1}}$, with $Value(S_{i+1}) \geq Value(S_i) - 1$.

Proof: If S_l schedules $Job_{i_{l+1}}$ on $m_{i_{l+1}}$ we are done, letting $S_{l+1} = S_l$. Otherwise, look at all jobs that intersect with $Job_{i_{l+1}}$ and are scheduled on $m_{i_{l+1}}$ in S_l. We claim there is at most one such job. Any such job Job_k can be consistently scheduled with $Job_{i_1}, ..Job_{i_l}$ on $m_{i_1}, ..m_{i_l}$, since it is so scheduled in S_l. Thus, it must finish after f_{i_l}, i.e., $f_{i_l} \leq f_k$, since otherwise it, not $Job_{i_{l+1}}$ would be the next scheduled. To intersect, it must also have $s_k < f_{i_l} \leq f_k$.

Thus, all such job must be in progress at time f. Since at most one job can be in progress on any one machine at any time in a legal schedule, at most one such job can be in S_l. Thus, we can construct S_{l+1} by deleting this one job (if there is one) and either moving $Job_{i_{l+1}}$ from its current machine to $m_{i_{l+1}}$ if it is scheduled in S_l or adding it if it is unscheduled. In any case, we delete at most one job, so $Value(S_{l+1}) \geq Value(S_l) - 1$.

Theorem: First finish time achieves approximation ratio 2.

Proof: Let S_0 be any schedule, and let S_{FF} be the schedule produced by first finish time. Assume S_{FF} schedules t jobs, $i_1, ..i_t$. Applying the Lemma repeatedly, we get S_1, S_2, $...S_t$ with each S_k scheduling jobs $i_1, ..i_k$ on the same machines as FF and with $Value(S_k) \geq Value(S_0) - k$. Since at the end of First Finish, no more jobs can be scheduled consistently, we must have $S_t = S_{FF}$, and since $t = Value(S_{FF}) = Value(S_t) \geq Value(S_0) - t$, $2t \geq Value(S_0)$, and so $Value(S_{FF}) = t \geq 1/2 Value(S_0)$, as claimed.

3 Weighted Independent Set

An *independent set* in a graph is a set of vertices I so that no two elements of I are directly connected by an edge in the graph. Look at the problem of, given a graph with positive node weights $w(u)$, finding an independent set of highest total weight, $Value(I) = \sum_{u \in I} w(u)$.

Consider a simple greedy algorithm that repeatedly chooses the node of highest weight, puts it in the independent set, and deletes the node and its neighbors. This algorithm will not be close to optimal for high-degree graphs, but we can use a transformation argument to show that in graphs of degree at most Δ, it achieves approximation ratio Δ.

Let $S_{greedy} = \{u_1, u_2, ..u_t\}$ be the greedy algorithm's independent set in the order chosen.

Lemma: Let S_i be any independent set containing $u_1, ..u_i$. Then there is an independent set S_{i+1} containing $u_1, ..u_{i+1}$ so that $Value(S_{i+1}) \geq Value(S_i) - (\Delta - 1) w(u_{i+1})$.

Proof: If $u_{i+1} \in S_i$, we are done. Otherwise, delete all at most Δ neighbors of u_{i+1} from S_i and add u_{i+1} to get S_{i+1}. No node in S_i can be a neighbor of $u_1, ..u_i$, and by definition u_{i+1} is not a neighbor of $u_1, ..u_i$. So any deleted node was in the graph at the time we chose u_{i+1}, and hence all deleted nodes u have $w(u) \leq w(u_{i+1})$. Thus, $Value(S_{i+1}) \geq Value(S_i) + w(u_{i+1}) - \Delta w(u_{i+1})$, as desired.

Theorem: The greedy algorithm acheives approximation ratio Δ on any graph of degree Δ.

Proof: Let S_0 be any independent set. Applying the lemma repeatedly, we get $S_1, ..S_t$ where S_i contains $u_1, ..u_i$ and $Value(S_i) \geq Value(S_0) - (\Delta - 1) \sum_{j=1}^{j=i} w(u_j)$. Now, the algorithm ends when we've deleted all nodes of the graph, so all nodes must either be in $u_1, ..u_t$ or neighbors of such nodes. Thus,

$u_1, ..u_t$ is a maximal independent set so any independent set containing them must be exactly this set, and $S_t = S_{greedy}$. Thus, $Value(S_{greedy}) \geq Value(S_0) - (\Delta - 1)\sum_{j=1}^{j=t} w(u_j) = Value(S_0) - (\Delta - 1)Value(S_{greedy})$. Hence, $\Delta Value(S_{greedy}) \geq Value(S_0)$, as claimed.