Using reductions to give approximation algorithms

Russell Impagliazzo

January 6, 2004

1 Defining reductions

An optimization problem Π has four ingredients:

1. An *instance format*, what types of objects I are inputs to the problem?
2. A *solution format*, what types of objects S are outputs to the problem?
3. *Constraints*, a property that should hold between the input and output $R(I, S) \in \{True, False\}$.
4. An *objective*, a function mapping instances and solutions to real numbers, which we either want to maximize (call the function $Value(I, S)$ in this case) or minimize ($Cost(I, S)$).

Let Π_1 be a minimization problem with constraints $R_1(I, S)$ and objective $Cost_1(I, S)$. Let Π_2 be a minimization problem with constraints $R_2(I, S)$ and objective $Cost_2(I, S)$.

A *reduction* from Π_1 to Π_2 consists of three functions, $f(I_1)$, $g(I_1, S_2)$, and $h(I_1, S_1)$ so that:

1. If I_1 is an instance of Π_1, then $f(I_1)$ is an instance of Π_2.
2. If S_2 is a legal solution for $f(I_1)$, i.e., if $R_2(f(I_1), S_1)$, then $g(I_1, S_2)$ is a legal solution for I_1, i.e., $R_1(I_1, g(I_1, S_2))$.
3. If S_1 is a legal solution for I_1, i.e., if $R_1(I_1, S_1)$, then $h(I_1, S_1)$ is a legal solution for $f(I_1)$ $R_2(f(I_1), h(I_1, S_1))$.

To save space, and make things less cluttered, we'll use the following abbreviations and assumptions in what follows:

1. I_1 is an instance of Π_1.
2. $I_2 = f(I_1)$ is the instance of Π_2 obtained by applying the reduciton to I_1.

3. S_1 is a legal solution to I_1 in Π_1 sense, i.e., $R_1(I_1, S_1)$. S_2 is a legal solution to I_2 in Π_2 sense, i.e., $R_2(I_2, S_2)$.

4. We'll drop the first parameter from $g, h, Cost_1, Cost_2$, so e.g., $g(S_2)$ denotes $g(I_1, S_2)$

Any such reduction converts an algorithm for the *search* version of Π_2 to the *search* version of Π_1. The search problem is to find any legal solution, if one exists. Let A be an algorithm that solves the search version of Π_2. Let $\overline{A}(I_1) = g(A_{search}(f(I_1)))$. Note that, paradoxically, we don't actually use the function h to get $\overline{A}$. But if such a function didn't exist, there would be no guarantee that $\overline{A}$ didn't send every instance of Π_1 to instances of Π_2 with NO solutions. Since they would have no solutions, A would not be expected to find solutions. So $\overline{A}$ would not find solutions, either, even for instances of Π_1 that had solutions.

To see that, if we have all three ingredients, $\overline{A}$ solves the search problem, assume S_1 is any solution to I_1. Then $h(S_1)$ is a solution to I_2, so I_2 has solutions. Therefore, $A(I_2)$ is a solution to I_2 (because A solves the search problem for Π_2). Hence, $\overline{A}(I_1) = g(A(I_2))$ is a solution to I_1. So whenever I_1 has a solution, $\overline{A}$ finds such a solution.

So far we have no conditions that link the *quality* of the solution A finds to that found by $\overline{A}$.

I'll give such a condition that will allow the reduction to be composed with any approximation algorithm, to get a related approximation algorithm. α and β will measure how much error is due to the reduction itself, as opposed to the approximation ratio of the algorithm we compose it with.

Say that f, g, h is an *α, β-reduction* if

1. For all legal solutions S_2 for I_2, $Cost_1(g(S_2)) \leq \alpha Cost_2(S_2)$

2. For all legal solutions S_1 for I_1, $Cost_2(h(S_1)) \leq \beta Cost_1(S_1)$

Theorem: If f, g, h is an $\alpha - beta$ reduction, from Π_1 to Π_2, and A approximately solves the optimization problem for Π_2 with approximation ratio γ, then $\overline{A}$ approximately solves the optimization problem for Π_1 with approximation ratio $\alpha\beta\gamma$.

Proof: If I_1 has no legal solutions, there is no requirement for $\overline{A}$. Otherwise, let S_1 be any solution for I_1, and let $S_2 = h(S_1)$. S_2 is a solution for I_2, so I_2 has solutions. Then $A(I_2)$ is a solution, and $Cost(A(I_2)) \leq \gamma Cost(S_2)$, Then $Cost_1(\overline{A}(I_1)) = Cost_1(g(A(I_2)) \leq \alpha Cost_2(A(I_2)) \leq \alpha\gamma Cost_2(S_2) = \alpha\gamma Cost_2(h(S_1)) \leq \alpha\beta\gamma Cost_1(S_1)$. Since S_1 is an arbitrary solution, this means $\overline{A}(I_1)$ is within an $\alpha\beta\gamma$ factor of the optimal solution for I_1.

2 Metric Traveling Salesman

As an example, we give a $(2,1)$-reduction from Metric TSP to minimum spanning tree. Since there are many efficient exact algorithms for MST, this gives an efficient 2-approximation algorithm for MetricTSP.

In the metric TSP, the instance is a set of vertices and distances between each vertex satisfying the triangle inequality. The f in the reduction is the identity function, i.e., we simply view the distances as edge weights in an undirected graph.

g is as follows: Given a solution T to the MST problem, we pick one node arbitrarily (say, the one whose name is first lexicographically.) We perform a DFS at this node, and output the tour which visits nodes in the order the DFS discovers them, returning to the root at the end. Since the DFS traverses each edge twice (once going away from the root, adn the second time coming back), and since the triangle inequality guarantees that going directly to the next visited node is only shorter than going via the DFS path, the constructed tour has length at most twice the cost of T. Thus, $\alpha = 2$.

h is as follows: Given a tour, pick a node arbitrarily, and construct a spanning tree that simply contains a path from that node to all the nodes visited, in order of the tour. Since we only decrease the total length (by failing to return to the first node), the total cost of the resulting spanning tree is less than the cost of the tour. Thus $\beta = 1$.

Since the MST problem is solved exactly, $\gamma = 1$, which, by the theorem, gives us a $2 * 1 * 1 = 2$ approximation algorithm for Metric TSP.