The sacrifice precision method

Russell Impagliazzo

January 13, 2004

1 The general method

Sometimes problems are only hard when the numerical quantities are arbitrary, complicated large integers. When these numerical quantities are limited to being small integers, the problems become efficiently solvable. (One way to formalize this is to consider these quantities as being specified in unary, and ask if the problem becomes polynomial-time solveable.) This is called being "hard in the weak sense". In this case, it frequently makes sense to round the input parameters to small integers, and hope (prove) that the solution to this "approximated instance" is an approximate solution to the actual instance. We can frequently view this "sacrifice precision" technique as a special kind of reduction, where a general instance of the problem is reduced to a "small integer" instance.

2 The 2-machine minimum makespan problem

We'll illustrate this method with the *minimum makespan problem*, a form of load-balancing.

Instance: A number of identical machines m, and a list of positive real job sizes $a_1, ..a_n$.

Solution Format: Each job is assigned to one of the m machines

Objective: Minimize the maximum load of a machine, $Cost(sched) = max_{1 \le i \le m} \sum_{jobs\ j\ assigned\ to\ machine\ i} a_j$.

For now, we consider the special case when $m = 2$.

We'll give a FPTAS (fully-polynomial approximation scheme) for this problem using the sacrifice-precision method. First, we'll give an efficient algorithm for the case when all a_i are small integers. Then we'll give an approximation-preserving reduction from the general case to this special case.

2.1 An algorithm for the small integer case

Assume all the a_i are integers in the range 1 to A. Then we can solve the minimum makespan for 2 machines exactly using dynamic programming in time polynomial in n and A.

For $1 \leq i \leq n$ and $\Delta \in \{-(i-1)A, ...(i-1)A\}$, we define the subproblem $MM(i, \Delta)$ as follows:

What is the minimum makespan of a schedule for jobs $i, ..n$ that starts with Δ more work on machine 1 than machine 2? (If $\Delta < 0$, that means machine 2 has more work.) i.e., we're looking for the cost of a schedule for $a_i, ..a_n$ that minimizes $max(\Delta + \sum_{jobs-j-assigned\ to\ machine\ 1} a_j, \sum_{jobs-j-assigned\ to\ machine\ 2} a_j)$.

We have the choice of putting a_i on machine 1 or 2. If we put a_i on M_1, the cost is $MM(i+1, \Delta + a_i)$ If we put it on M_2, the cost is $a_i + MM(i+1, \Delta - a_i)$, because we can subtract a_i from the current load on both machines, and add it in to the final cost.

This gives a dynammic programming algorithm to compute the minimal cost solution as follows:

DPMinMakespan (A: positive integer, $a_1, ..a_n$: array of integers, $1 \leq a_i \leq A$): integer

1. Initialize $Cost(I, \Delta)$, $1 \leq I \leq n$, $-nA \leq \Delta \leq nA$.
2. For $\Delta = -nA$ TO nA do:
3. IF $\Delta \geq 0$ THEN $Cost(n, \Delta) = max(\Delta, a_n)$ ELSE $Cost(n, \Delta) \leftarrow max(\Delta + a_n, 0)$.
4. FOR $I = n - 1$ downto 1 do:
5. FOR $\Delta = -A(I-1)$ to $A(I-1)$ do:
6. $Cost(I, \Delta) \leftarrow min(Cost(I+1, \Delta + a_I), a_I + Cost(I+1, \Delta - a_I)$.
7. Return $Cost(1, 0)$.

The algorithm takes time $O(n^2 A)$. It is easy to put in a final loop that outputs an optimal solution.

2.2 The reduction

Remember from last class that we need to provide 3 ingredients to give an approximation preserving reduction: A map f from instances of the first problem to instances of the second, a map g from solutions of the second problem to solutions of the first, and a map h from solutions of the first to solutions for the second.

Let $\epsilon > 0$ be given. (The smaller ϵ is, the tighter our approximation will be, but the more time the algorithm will take.)

We'll reduce the general problem $a_1, ..a_n$ to an integer valued problem $A, \hat{a_1}, ...\hat{a_n}$ as follows:

f: Let $A = \lceil 2n/\epsilon \rceil$. Let $T = \sum_{1 \leq i \leq n} a_i$, $\tau = \epsilon T/2n$, and let $\hat{a_i} = \lceil \frac{a_i}{\tau} \rceil$.

The idea is that the $\hat{a_i}$'s are in roughly the same proportions as the a_i's so that a load-balancing schedule for the a_i's is roughly load-balancing for the $\hat{a_i}$'s and vice versa. g and h will formalize this. g and h just schedule the corresponding jobs on the same machines. If we didn't round up when computing $\hat{a_i}$, g would multiply the cost by a factor of τ, and h by $1/\tau$, because we'd multiply all the sizes of jobs by these factors. We need to show that this is still approximately the case with rounding.

We'll bound the amount the load of any one machine changes as a fraction of its load in the other schedule. Then the maximum load is at most this fraction of the maximum load of the other schedule.

Let *sched* be a schedule of jobs to machines. Let *cost* be the makespan of *sched* for $a_1, ..a_n$, and $\hat{cost}$ be the same for $\hat{a_1}, ..\hat{a_n}$. Look at the load of a machine M in the instance $a_1, ..a_n$ $\sum_{i \ assigned \ to \ M} a_i \leq \sum \tau \lceil \frac{a_i}{\tau} \rceil = \tau \sum_{i \ assigned \ to \ M} \hat{a_i}$. So the cost of *sched* for the a_i is at most a factor τ of its cost under $\hat{a_i}$, $cost \leq \tau \hat{cost}$.

In the other direction, the $\sum_{i \ assigned \ to \ M} \hat{a_i} = \sum \lceil \frac{a_i}{\tau} \rceil \leq \sum_{i \ assigned \ to \ M} (\frac{a_i}{tau} + 1) \leq \frac{1}{\tau}((\sum_{i \ assigned \ to \ M} a_i) + \tau n) = 1/\tau((\sum_{i \ assigned \ to \ M} a_i) + \epsilon T/2)$. Thus, $\hat{cost} \leq 1/\tau(cost + \epsilon T/2)$. As mentioned before, any schedule has cost at least $T/2$, so this means $\hat{cost} \leq \frac{1}{\tau}(cost + \epsilon cost) = \frac{1+\epsilon}{tau}(cost)$.

Thus, we have an α, β reduction with $\alpha = \tau$, and $\beta = \frac{1+\epsilon}{tau}$. Since we solve the problem we reduce to optimally , this means the approximation factor for our algorithm is $\tau \frac{1+\epsilon}{\tau} = 1 + \epsilon$.

The total time for the algorithm is dominated by the DP algorithm, which we run with $A = 2n/\epsilon$. Since the time of that algorithm is $O(n^2 A)$, this gives a total time of $O(n^3/\epsilon)$. So we can pick ϵ arbitrarily small and still have an $O(n^3)$ algorithm, or pick $\epsilon = 1/n^c$ and have a polynomial time algorithm.

3 A PTAS for the general problem

The above FPTAS used an arithmetic rounding function: we re-scaled and rounded to the nearest integer, guaranteeing that $\hat{a} - a$ was bounded. We'll get a PTAS for the general problem with m machines by using a geometric rounding function, that will insure $a/\hat{a}$ is bounded. The geometric rounding function won't make all $\hat{a}$'s small integers, but it will ensure that there aren't many possible values for $\hat{a}$.

3.1 The reduction

The rounding procedure is as follows. We'll use a "guess-and-verify" approach. Let t be a guess that approximates the real optimal makespan. Let $a_1, ..a_n$ be the sizes of the jobs to be scheduled, let $T = \sum_{1 \leq i \leq n} a_i$ be the total size of all jobs, and let $\delta > 0$ be given.

Define $l = log_{1+\delta} 1/\delta$. Let $\beta_0 = \delta t$, and let $\beta_i = \beta_0(1+\delta)^i$, for $i = 1, ..l$. (Note that $\beta_l > t$).

If any $a_i > t$, our guess is clearly incorrect and we can go on to another t. Otherwise, define a new instance $\hat{a_1}, ..\hat{a_n}$ by: $\hat{a_i} = 0$ if $a_i < \beta_0$, and $\hat{a_i} = \beta_j$ if $\beta_j \leq a_i < \beta_{j+1}$.

We need to relate the optimal solutions for the original a_i and the rounded $\hat{a_i}'s$.

Claim 1: If $a_1, ..a_n$ has makespan at most t on m machines, then so does $\hat{a_1}, ..\hat{a_n}$.

Proof: $hata_i \leq a_i$, so the same schedule has only smaller makespan for the input $\hat{a_i}$.

Less trivially: Claim 2: If $\hat{a_1}, ..\hat{a_n}$ has makespan at most t on m machines, and $t \geq T/m$ then $a_1, ..a_n$ has a schedule with makespan at most $t(1+\delta)$. Moreover, from a schedule for the $\hat{a_i}$ of makespan t, we can construct such a schedule for the a_i in time polynomial in n.

Let *sched* be a schedule for $\hat{a_1}, ..\hat{a_n}$ with makespan at most t. The construction goes in two steps: First, we'll use *sched* as a schedule for all jobs a_i with $a_i \geq \beta_0$. Then we'll add in all the small jobs, with $a_i < \beta_0$, placing each in the machine whose load is currently smallest.

Note that, if $a_i \geq \beta_0$, then $\hat{a_i} = \beta_j$ where $\beta_j \leq a_i < \beta_{j+1} = \beta_j(1+\delta)$, so $a_i < (1+\delta)\hat{a_i}$.

Thus, for any machine M, under sched, $\sum_{i \ assigned \ to \ M, a_i \geq \beta_0} a_i \leq \sum_{i \ assinged \ to \ M, a_i \geq \beta_0} ((1+\delta)\hat{a_i}) \leq (1+\delta)t$, since by assumption *sched* has makespan at most t.

So after the first stage, our schedule has makespan $< (1+\delta)t$. We claim that this stays true throughout the second stage, adding each small job to the schedule. Each time we add a small job, we only increase the load of the currently least loaded machine. Thus, we could only exceed makespan $(1+\delta)t$ after this step if the load of that machine became more than $(1+\delta)t$. Before we added this small job, the sum of all loads of all m machines is $< T$, so the average load is at most T/m, and the least load is at most the average. Thus, before this step, the least loaded machine had load less than T/m. After, it has load less than $T/m + a_i \leq T/m + \beta_0 \leq t + \delta t = (1+\delta)t$. Thus, no one addition of a small job can increase the makespan to more than $(1+\delta)t$, and so the final makespan is at most $(1+\delta)t$.

3.2 Solving the reduced, succinct instance

The reduced instance $\hat{a_i}$ has a special property that will allow us to solve it quickly. Each $\hat{a_i} > 0$ is in the set $\beta_0, ..\beta_l$, so there are at most $l+1$ *distinct* sizes of jobs. Since l is a constant for any fixed δ, this will allow us to solve the problem in polynomial time in n.

In general, we'll give a dynamic programming algorithm for the following problem:

Succinct Min Makespan:

Instance: An array of non-negative integers $n_1, ..n_l$, and positive real numbers $\beta_1, ..\beta_l$, and a number of machines m.

Problem: Find a schedule on m machines of minimum makespan for n_1 jobs of size β_1, n_2 jobs of size β_2, $...n_l$ jobs of size β_l.

Let $n = \sum n_i$. We'll sketch an algorithm for this problem that runs in time $O(n^{2l}m)$. This is not polynomial in the length of the succinct problem instance, but we'll be using it on the reduced instance, where l is a fixed constant that depends only on δ.

The idea is to branch on all ways of assigning jobs to machine 1. Since we have to pick a number $0 \leq m_1 \leq n_1$ jobs of size β_1, $...0 \leq m_l \leq n_l$ jobs of size β_l there are at most $(n+1)^l$ such choices. Each gives us a sub-problem of the form: Find the min makespan schedule of $n_1 - m_1$ jobs of size β_1,$...n_l - m_l$ jobs of size β_l on $m-1$ machines. Doing this recursively, we'd have a family of possible sub-problems that might come up: Find the min makespan schedule on m' machines of r_1 jobs of size β_1,$..r_l$ jobs of size β_l, where $0 \leq r_i \leq n_i$ are integers. Thus, there are also at most $m(n+1)^l$ such sub-problems, so by using dynamic programming, we can solve and store each possible sub-problem.

The time per sub-problem is $O(n^l)$, and the number of sub-problems is $O(m(n^l))$, giving us the claimed time bound.

3.3 The final PTAS

Here's how we put it together to get the PTAS. Let $\epsilon > 0$ be given, and pick $\delta > 0$ so that $(1+\delta)^2 \leq 1+\epsilon$.

Initialize $t = T/m$.

Repeat until we find a schedule of makespan $(1+\delta)t$:

Compute the reduced instance $\hat{a_1}, ..\hat{a_n}$ for this t. Use the above DP algorithm to find the optimal schedule for $\hat{a_i}$'s, in time $O(n^{2l+2}m)$, where $l = \log_{1+\delta} 1/\delta$.

If the makespan found is at most t, use the construction in claim 2 to find a schedule for $a_1, ..a_n$ of makespan $(1+\delta)t$ and halt.

Otherwise, $t \leftarrow t(1+\delta)$ and repeat.

Let t_{opt} be the makespan of the optimal schedule for $a_1, ..a_n$. First, consider the first time when $t \geq t_{opt}$. Since the previous t was less than t_{opt}, $t \leq (1+\delta)t_{opt}$. Since there is a schedule for the a_i's of makespan t, by claim 1 there is such a schedule for the $\hat{a_i}$'s. By the correctness of our DP algorithm, the DP algorithm finds such a schedule. Then by Claim 2, the procedure to construct a schedule for the a_i's has makespan at most $(1+\delta)t \leq (1+\delta)^2 t_{opt} = (1+\epsilon)t_{opt}$. So the algorithm has approximation ratio $1+\epsilon$. Second, since $t_{opt} \leq T$, the maximum number of loops before this occurs is $\log_{1+\delta} m$, so the total running time is at most $O(n^{2l+2} m \log m)$, which is polynomial for any fixed $\delta > 0$.

(Note: But it's a very BIG polynomial for any reasonable value of δ!!!)