Relative Performances

From the preceding section it is clear that, while absolute performance guarantees are the most desirable ones, it is quite unlikely that we can give such guarantees for any interesting class of hard optimisation problems.

Therefore it seems reasonable to relax the requirement for a "good approximation algorithm".

We start by examining the problem of multiprocessor scheduling and use it to motivate the definition of relative performance guarantees.

Interestingly enough, the whole field of approximation algorithms has its roots in the work of Graham in 1966 on the problem of scheduling. In fact, scheduling problems probably have the most well-developed body of work from the point of view of approximation algorithms.

Multiprocessor scheduling

The input consists of n jobs. Each job has a corresponding runtime $p_1, \ldots, p_n$, where each p_i is assumed to be rational. The jobs are to be scheduled on m identical machines or processors so as to minimise the finish time. The finish time is defined to be the maximum over all processors of the total run-time of the jobs assigned to that processor. The set of feasible solutions consists of all partitions of the n jobs into m subsets, and the value of a solution is the maximum over all subsets of the total run-time of the subset. The problem is known to be NP-hard even in the case where $m = 2$.

Algorithm LIST SCHEDULING

Consider the following algorithm due to Graham which is called the *list scheduling*(or LS) algorithm. The algorithm considers the n jobs one-by-one, assigning each job to one of the m machines in an online fashion. The rule is to assign the current job to that processor which is (at that point) the least loaded processor. Note that the load on a processor is the total run-time of all the jobs assigned to it.

For all input instances x,

$$\frac{c(x,\mathrm{LS}(x))}{\mathrm{opt}(x)} \leq 2 - \frac{1}{m}$$

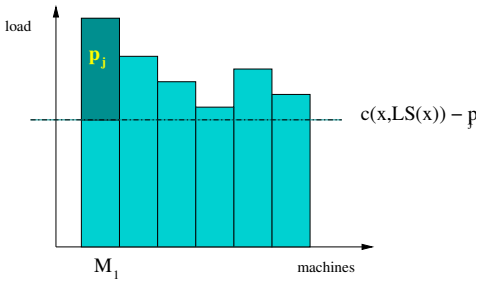
Moreover, this bound is tight in that there exists an input instance x^* such that

$$\frac{c(x^*,\mathrm{LS}(x^*))}{\mathrm{opt}(x^*)} = 2 - \frac{1}{m}$$

Argument

Let us first prove the upper bound on the ratio. Assume, without loss of generality, that after all the jobs have been assigned the machine M_1 has the highest load. This is the cost of the solution returned by the list scheduling algorithm, that is $c(x, \mathrm{LS}(x))$.

Also, let J_j denote the last job assigned to this machine.



We claim that every machine has a total load of at least $c(x, \mathrm{LS}(x)) - p_j$. This is because when J_j was assigned to M_1, M_1 was the least loaded processor (with a load exactly $c(x, \mathrm{LS}(x)) - p_j$).

It then follows that

$$\sum_{i=1}^{n} p_i \geq m(c(x, \mathrm{LS}(x)) - p_j) + p_j$$

But it is also the case that

$$\mathrm{opt}(x) \geq \tfrac{1}{m} \sum_{i=1}^{n} p_i$$

since some processor must have this much load at the end of the scheduling process. Therefore

$$\mathrm{opt}(x) \geq (c(x, \mathrm{LS}(x)) - p_j) + \tfrac{p_j}{m} = c(x, \mathrm{LS}(x)) - \left(1 - \tfrac{1}{m}\right) p_j$$

Observing that $\mathrm{opt}(x) \geq p_j$ since some processor has to execute the job J_j, we obtain the desired result.

To see that the algorithm actually achieves this ratio, consider the following input instance x^*. Let $n = m(m-1)+1$ and let the first $n-1$ jobs have a run-time of 1 each, while the last job has $p_n = m$. It is easy to see that $\mathrm{opt}(x^*) = m$ while $c(x^*, \mathrm{LS}(x^*)) = 2m - 1$. This gives the desired lower bound on the ratio.

Approximation ratio

We have just proved:

$$c(x, \mathrm{LS}(x)) \leq \left(2 - \tfrac{1}{m}\right) \mathrm{opt}(x).$$

The interesting thing to note about this result is that we are measuring the quality of the approximation algorithm in terms of the ratio between the value of its solution and that of the optimal solution. This is exactly what we mean by a relative performance measure. The following definition formalises this notion.

Given an instance x and a feasible solution y of x for a given NPO problem, the *performance ratio* of y with respect to x is

$$R(x, y) = \max\left\{\tfrac{c(x,y)}{\mathrm{opt}(x)}, \tfrac{\mathrm{opt}(x)}{c(x,y)}\right\}$$

"Reasonable" approximation algorithms

We now can define more precisely the type of approximation algorithms that we will consider.

Let an NPO problem be given and let A be an algorithm that, for any given instance x of the problem, returns a feasible solution $A(x)$ of x. Given an arbitrary function $r : \mathbb{N} \to (1, \infty)$ we say that A is an *$r(n)$-approximation algorithm* for the given problem if, for every instance x of order n,

$$R(x, A(x)) \leq r(n).$$

Also we say that the problem *can be approximated with ratio* $r > 1$ if there exists an r-approximation algorithm for it.

Applying these definitions to the list scheduling algorithm, we have that the list scheduling algorithm is a $2 - \frac{1}{m}$ approximation algorithm.

Better results for scheduling

There is an even better approximation algorithm for the scheduling problem called LPT. This algorithm first orders the jobs by decreasing value of their run-times. After this, the algorithm behaves exactly as the list scheduling algorithm.

The performance ratio of the LPT algorithm is at most $\frac{3}{2}$.

Example

Suppose there are four machines ($m = 4$) and the seven jobs to schedule, with processing times:

$$1 \quad 2 \quad 1 \quad 3 \quad 3 \quad 2 \quad 6$$

LS would assign the first and the fifth job to machine one (total load of four), the second job and the last one to machine two (total load of eight), the third and the sixth job to machine three (total load three), and the fourth to machine four (total load of three).

By sorting the jobs first LPT would assign the heavy-weight job seven first, and then proceed like LS. It can be easily checked that the resulting solution is optimal.