Algorithmic Mechanism Design Tutorial

Brendan Lucier
(Recorded by Geoffrey Peddle)

Nov. 16, 2010

1 Introduction

We are going to consider algorithm design from a game theoretic viewpoint. What changes is that now agents, with vested interests, hold some of the values that are important to the outcome. We want to develop algorithms that are resilient to the possibility that agents will lie in order to manipulate the outcome for their own benefit. Most of todays discussion is based on the paper "Truthful Mechanisms for One-Parameter Agents" by Aaron Archer and Eva Tardos.

2 Makespan problem

Consider the related machine makespan minimization problem.

Definition. *Given n jobs with processing times $\{p_1, p_2, \cdots, p_n\}$ and m machines with speeds $\{s_1, s_2, \cdots, s_m\}$ we want to assign the jobs to machines to minimize the maximum finish time.*

Recall that the load on machine j is $L_j = \sum_{i \in I_j} p_i$ where we are summing over the jobs I_j assigned to machine j. The makespan is $C_{max} = max_j L_j/s_j$.

The game theoretic twist: Machines are rational agents. If machine j runs a load L_j it incurs a cost (negative value) of L_j/s_j. To minimize this cost machines want to minimize the load that they are assigned. The machine speed is private information and they will lie to reduce their load and thus their cost.

We need to convince machines to reveal their true speed, not under report their speed to avoid work. This will allow us to solve our makespan problem. The idea is to pay machines for their time so they will no longer have an incentive to lie.

Definition. *A* mechanism *takes bids (i.e. declared speeds) and returns an assignment of jobs plus payments to the machines $P_1, P_2, \cdots, P_m$.*

The utility of machine j is $P_j - L_j/s_j$ or the payment - load / true speed.

We will write $\{b_1, b_2, \cdots, b_m\}$ for declared speed (bids) and $u_j(\vec{b})$ for the utility of machine j resulting from bids $\vec{b}$ where $\vec{b}$ represents the set of bids by all machines. Note that the utility of machine j also implicitly depends on s_j, its true speed.

We write b_{-j} for the bids of all machines excluding machine j.

Definition. *A mechanism is* truthful *if the utility of a machine is maximized by telling the truth. That is* $\forall j, \forall s_j, \forall b_{-j}$ *and* $\forall b_j$ *its the case that* $u_j(s_j; b_{-j}) \geq u_j(b_j; b_{-j})$.

Assume that each agent only cares about maximizing its own utility so it won't lie unless it will directly benefit. Each agent's utility is only a function of what jobs that agent gets assigned.

Definition. *Let* $L_j(b_j; b_{-j})$ *be the load assigned to machine* j *by an algorithm for our makespan problem. Then that algorithm is* monotone *if* $L_j(b_j; b_{-j})$ *is a monotone non-decreasing function* $(\forall j, \forall b_{-j})$.

Another way of saying this is that the allocation to machine j is non-decreasing as the declared speed increases.

Theorem 1. *Given an algorithm A, there exists a payment scheme that makes A truthful if and only if A is monotone.*

The problem of designing a truthful mechanism therefore reduces to designing a monotone algorithm. Intuitively it would seem easy to come up with monotone algorithms but this is often not the case.

2.1 Non-monotone algorithm

Recall the greedy longest-processing-time algorithm: order the jobs from longest to shortest processing time and then greedily assign each to the least-loaded machine. This algorithm is a 2-approximation, but it is not monotone. Consider an example with 3 tasks and 2 machines.

Let the tasks be $p_1 = 2$ and $p_2 = p_3 = 1 + \epsilon$

Let the machine speeds be $s_1 = 1$ and $s_2 = 1 - \epsilon$

Our greedy algorithm assigns: $p_1 \to s_1$, $p_2 \to s_2$ and $p_2 \to s_2$ resulting in loads $L_1 = 2$ and $L_2 = 2(1 + \epsilon)$

But if we increase the speed of machine 2 to $s_2' = (1 + \epsilon)$ then its load decreases from $2(1 + \epsilon)$ to 2. So this algorithm is not monotone.

2.2 Monotone algorithm

After much work people have a monotone PTAS solution that is very complex. Instead of looking at this we are going to look at a randomized 3-approximation that is truthful in expectation (i.e. agents maximize their expected utility by declaring truthfully). Note that with random algorithms the definition of monotonicity changes slightly but has the same form. Now $L_j(b_j; b_{-j})$ becomes the expected load. Our random monotone greedy algorithm for a 3-approximation is algorithm 1.

We will now describe steps 2-4 in more detail, starting with **step 2**.

```
Algorithm 1: Input: declared speeds $s_1, s_2, \cdots, s_m$ and job times $p_1, p_2, \cdots, p_n$
1 Sort input so $s_1 \geq s_2 \geq \cdots \geq s_m$ and $p_1 \geq p_2 \geq \cdots \geq p_n$.
2 Compute a lower bound T on the optimal makespan.
3 Greedily make a fractional assignment with makespan T.
4 Round to an integral assignment with makespan $\leq 3T$.
```

Definition. *A fractional assignment is* valid *for makespan T if whenever job j is partially assigned to machine i, $p_j/s_i \leq T$.*

This is saying that if job j was the only job on machine i, then it would complete in less time than T. Now we give the algorithm that will attempt to do a fractional assignment with makespan T.

```
Algorithm 2: Fractional assignments
1  $i \leftarrow 1$;
2  $L_1, L_2, \cdots, L_m \leftarrow 0$;
3  for $j = 1, \cdots, n$ do
4      Compute what fraction of job j to put on machine i.
5      if $L_i + p_j \leq T \times s_i$ then
6          $p_j$ assigned to machine i
7          $L_i \leftarrow L_i + p_j$
8      end
9      else
10         place fraction of $p_j$ onto machine i such that
           $L_i + p_j \times fraction = T \times s_i$
11         place the rest of the job on machine i+1.
12         $i \leftarrow i + 1$ (switch to machine $i + 1$)
13     end
14 end
```

Details: Step 3

Note: this greedy algorithm creates a fractional assignment. Is this assignment valid? Write $i(j)$ for the slowest machine onto which job p_j fits (in time T). Then we find a valid assignment if job j is placed on or before machine $i(j)$ for all j. So, if the greedy algorithm creats a valid assignment where job j is placed on machine i, we must have

$$T \geq \max\{p_j/s_i, \sum_{k=1}^{j} p_k / \sum_{l=1}^{i} s_l\}.$$

Since every job must be placed on some machine, the above equation must hold for some i, for every j. So it must be that

$$T \geq \max_j \min_i \max\{p_j/s_i, \sum_{k=1}^{j} p_k / \sum_{l=1}^{i} s_l\}.$$

Call the right-hand side of the above inequality $T_{L.B.}$; this will be our lower bound for step 2.

One can show that the greedy algorithm does actually find a valid assignment when $T = T_{L.B.}$.

Lemma: There exists a fractional assignment meeting lower bound T.

Details: Step 4.

Our rounding is as follows. If some fraction α of job p_j is on machine i then assign p_j to machine i with probability α. Note that the expected load will be equal to the fractional load with this assignment so it is enough to show that the fractional load is monotone.

Claim: Fractional loads are monotone.

Suppose that machine i lowers its declared speed from s_i to $\acute{s_i} = s_i/\alpha, \alpha > 1$.

Let $T^{'}_{L.B.}$ be the new lower bound with declared $\acute{s_i}$ and $T_{L.B.}$ be the lower bound with declared s_i.

Note: $T^{'}_{L.B.} \geq T_{L.B.}$ and $T^{'}_{L.B.} \leq \alpha \times T_{L.B.}$

If machine i that we are considering is the slowest then it was allocated the remainder when all other machines had finished with time $T_{L.B.}$. Since $T^{'}_{L.B.}$ is at least as large as $T_{L.B.}$ this left over portion will only get smaller when bid changes from s_i to $\acute{s_i}$.

Otherwise if machine is not the slowest then the load will change from $T_{L.B.} \times s_i$ to $T^{'}_{L.B.} \times \acute{s_i}$. But $T^{'}_{L.B.} \times \acute{s_i} = T^{'}_{L.B.} \times s_i/\alpha \leq T_{L.B.} \times s_i$ so the load decreases.

We conclude that our algorithm is monotone, so we could implement this algorithm and calculate payments so that no machine would have an incentive to lie about its speed.