Brief Note on Competitive Analysis

Jasper Lee

(Updated: October 4, 2018)

This note aims to complement Paul's note on *competitive analysis*, motivating the definition of *competitive ratio* and explaining how an algorithm can be shown to have a certain competitive ratio, through a running example.

There is a small number of exercises throughout the note. Solutions are included in the appendix.

1 What's the context?

Algorithmic tasks we have seen so far in class all follow a same framework: given the entire description of a problem instance, solve it in some efficient manner. Examples:

- From the team presentation problems, a singer wishes to plan the days she wishes to perform, given the different amounts of money she would earn each day if she sang, subject to constraints that she requires rest.
- More in line with "Map of the Computer" section of the course, given a sequence of cache line requests, compute the most efficient plan of evicting cache lines to minimise the number of evictions.

Both problems are instances of planning problems, where the concrete parameters of the planning scenario is given, and we wish to compute the optimal plan given these parameters.

Planning scenarios exemplify how the above framework does not always suit practical needs. In the singer's planning example, she might only find out on the day how much she could earn, instead of ahead of time. Similarly, a computer does not know future cache requests when a cache eviction decision has to be made. In both situations, a part of the solution has to be computed and committed to before the full problem instance is made available, yet the commitment may restrict the choices available in other parts of the solution.

Clearly, if not all information is present, for general problems there is no way to make a decision that guarantees optimality no matter what the rest of the information is. A natural question then is, given a strategy or an algorithm for making these decisions, is there some way to measure its performance that captures the fact that the algorithm makes decisions when lacking information? If so, can we design algorithms that work well under this performance measure?

The answers to the two questions are both *yes*. This note introduces the *competitive analysis* framework for analysing algorithms that compute with incomplete information. We shall define how an algorithm's performance can be quantified, namely the notion of *competitive ratio* of an algorithm, and discuss how such ratio can be calculated and proven.

Before we proceed to the details of the framework, a brief comment on the applicability of the analysis framework is warranted. Both the performance planning and cache line eviction scenarios considered above are examples of *online* optimisation tasks. That is, the problem instance is made available as a long sequence of parts, in some serial order (respecting time, for example), and the algorithm needs to make a decision before the next part becomes known. However, online algorithmic tasks are not the only situations where an algorithm needs to make a decision before all relevant information is known. Another common class of algorithmic task simply hides an important part of the instance, and waits for the algorithm to make all the decisions in "one shot", before revealing the entire instance. Competitive analysis is also a suitable framework for these scenarios, and in general applicable whenever the algorithm is required to compute (parts of) its output without knowing the entire input instance.

A related area is the study of *approximation algorithms*, for which the obstacle to computing optimal solutions is not the missing information, but simply that the optimisation problems are too difficult to solve. You should keep the competitive analysis topic in mind when we start talking about "NP-hardness" in class, and think about the similarities and differences between the topics.

2 Defining the competitive ratio

To make the exposition clearer, let us use the following problem from a past midterm as a running problem throughout the note. For more examples, refer to Paul's lectures and note on the topic.

Problem 1. Piglet is going to Eeyore's birthday party, but realizes he forgot to bring presents. Luckily, on the path to Eeyore's house, there are lots of acorns and pine cones; Piglet's bag can hold 100 acorns, or 10 pine cones, or any combination of the two (pine cones are 10 times as big as acorns). Piglet, unfortunately, doesnt know whether Eeyore likes acorns or pine cones; for each acorn Piglet brings, Eeyore will gain $A \geq 0$ happiness; and for each pine cone Piglet brings Eeyore will gain $P \geq 0$ happiness; however Piglet does not know A or P (and neither do you).

1. With competitive analysis in mind, design a strategy for Piglet to choose how many acorns and pine cones to bring to Eeyore's birthday.

2. What is the competitive ratio of your strategy against a wizard who brings the optimal presents? (Justify your answer)

Each algorithmic task has its own way to measuring the performance of an algorithm. Here, we measure a strategy by the *happiness* it generates, which is a function of A and P, the two parameters defining the instance of the problem. Suppose we have a strategy $\mathsf{Alg} = (n_A, n_P)$, where n_A and n_P are the numbers of acorns and pine cones respectively. How do we quantify how optimal Alg is?

A common (but not the only reasonable) way is to consider the *competitive ratio* of strategy Alg. Let us try deriving the definition of this notion. Denote by $h_{\mathsf{Alg}}(A,P)$ and $h_{\mathsf{Opt}}(A,P)$, as functions of A and P, the happiness generated by Alg and Opt, where Opt is the optimal offline strategy that is computed knowing A and P. Since we are trying to define the competitive *ratio* of Alg, the definition must contain the following expression[1]:

$$\frac{h_{\mathsf{Opt}}(A,P)}{h_{\mathsf{Alg}}(A,P)}$$

Are we done? Unfortunately not, because this is not a well-formed expression, as A and P are neither defined in the context, nor properly quantified. Each instance of (A,P) gives a different ratio, so which one of these ratios would we want? Following the usual "worst case" flavour of complexity analysis, and recalling that the happiness measure in this problem is one we want to maximise rather than minimise, we take the maximum over all these ratios to be the competitive ratio of Alg, thus giving us the following definition:

$$\max_{A\geq 0, P\geq 0} \frac{h_{\mathsf{Opt}}(A,P)}{h_{\mathsf{Alg}}(A,P)}$$

An algorithm is ρ-*competitive* if its competitive ratio is at most ρ.

What if we had a different problem, and the performance measure were one that we instead wish to minimise, for example the cost of implementing some plan? We would invert the ratio to keep it at least 1, since the cost of our algorithm would be at least as great as that of Opt, and we would still maximise the ratio over all possible instances in the spirit of worst case analysis.

Having formally defined the notion of competitive ratio, it is important to emphasise again that this is only one way, albeit a very widely used way, to evaluate algorithms. In some contexts it may also be reasonable to consider the additive difference between the performance of Alg and Opt instead of their ratio.

Before we move onto the next sections to discuss how the competitive ratio is analysed, we need to write down an algorithm for the analysis. The following exercise asks you to come up with a 2-competitive algorithm.

Exercise 1. Write down an algorithm $\mathsf{Alg} = (n_A, n_P)$ that is 2-competitive for Problem 1.

The answer to the exercise is included in the appendix, and also stated in the next section for analysis. You should of course attempt to solve the problem before turning to the next page to look at the solution.

[1]Here we take the convention that the ratio should be at least 1, but in the literature some authors consider the reciprocal and use ratios that are at most 1.

3 “The worst case is ...”: How to write an invalid proof

After thinking about Exercise 1, you should hopefully arrive at the answer ($n_A = 50, n_P = 5$). The intuition is that space in Piglet’s bag is the “resource” to be allocated to each option, and a common technique in designing competitive algorithms is to commit to each option by the same amount. Here, we are committing an equal fraction (namely, half) of the bag’s space to each of the two possibilities.

At this point, you may be *very* tempted to write something like the following as the analysis:

> Denote our strategy by $\mathsf{Alg} = (n_A = 50, n_P = 5)$. The worst cases are when 1) $A = 0$ and $P > 0$, and similarly 2) $A > 0$ and $P = 0$. In case 1), Opt would take 0 acorns and 10 pine cones, yielding a happiness of $10P$, whereas Alg yields a happiness of $5P$, thus the ratio is 2 in this case. Similarly, in case 2), Opt would take 100 acorns and 0 pine cones, leading to a happiness of $100A$, which again twice the happiness Alg gives.
>
> Thus we conclude that the competitive ratio is 2.

There are unfortunately significant issues with the above argument. The phrase “the worst cases” immediately raises the question of “worst cases for what?” Two interpretations are relevant in the context, namely 1) the worst cases for the problem, and 2) the worst cases for Alg. However, the first interpretation is in general nonsensical. The reason is that whether an instance is “bad” depends on the algorithm. For example, the case $A = 0$ and $P > 0$ might be bad for Alg, but it is a good instance for the strategy ($n_A = 0, n_P = 10$). That leaves us the second interpretation, namely that the two cases mentioned are supposedly the worst cases for Alg.

Given this interpretation, let us then unpack the (very strong) claim made just in the second sentence, that the worst cases (for Alg) are the two cases stated. What does it mean for an instance to be a worst case? In the context of the conclusion that follows the claim, it must be interpreted to mean that these are the instances (A, P) that have the greatest ratio of $h_{\mathsf{Opt}}(A, P)/h_{\mathsf{Alg}}(A, P)$. That is, the claim is formally[2]:

$$(A, P) = \arg\max_{A \geq 0, P \geq 0} \frac{h_{\mathsf{Opt}}(A, P)}{h_{\mathsf{Alg}}(A, P)}$$

To prove this claim is at least as hard as proving that the competitive ratio is 2! Therefore the argument above is essentially a circular argument, and does not prove that the competitive ratio of Alg is 2.

The moral of this section[3] is to avoid the phrase “the worst case(s)”, unless you really understand what it means, and are prepared to justify the claim that the case(s) you give are indeed the worst. In the next section, we shall see how we can do the analysis properly.

[2]The operation $\arg\max_x f(x)$ gives x^* such that $f(x^*) = \max_x f(x)$

[3]This moral in fact applies to essentially every part of the course.

4 Analysing the competitive ratio correctly

Let us first formally state the claim.

Theorem 1. *The strategy* $\mathsf{Alg} = (n_A = 50, n_P = 5)$ *for Problem 1 has a competitive ratio equal to 2.*

In order to reason about the competitive ratio, it is important to understand what Opt does. The following exercise asks you to state and prove the behaviour of Opt.

Exercise 2. For Problem 1, write down the optimal offline strategy Opt (which can be computed depending on A and P), and show that it yields happiness $\max(100A, 10P)$. Show that the strategy you wrote down is optimal.

Equipped with an understanding of Opt, we now move on to the analysis structure. Recall that the competitive ratio of Alg is defined as the maximum over a set of ratios, and we want to show that the maximum is equal to some concrete number (2 in this case). How do we rigorously show this equality, given that we cannot just use simple equalities since the competitive ratio is the maximum of a set of numbers? Think about the following exercise for a bit before reading on.

Exercise 3. Consider arbitrary quantities a and b, and the logical statement asserting that $a = b$. Express the equality in terms of two less restrictive statements. This is an informal and vaguely phrased exercise, but thinking about decomposing a complicated statement into statements that are easier to prove is a common technique and a useful skill to have.

Think about Exercise 3

An answer to Exercise 3 that is useful to the analysis is that $a = b$ is true if and only if both $a \leq b$ and $a \geq b$ are true. For now, this may seem like a "come on, that's dumb" answer, but it turns out it does give us a handle on proving the result.

There are two statements we need to prove, so let us first consider the easier one, namely to show that the competitive ratio is at least 2. Again, how can we show that the maximum of a set is lower bounded by a number? Try re-expressing the statement into a form that is more amenable to proving. The answer is on the next page.

Exercise 4. Given a set of numbers $R = \{f(x) \,|\, x \in X\}$, show that $\max\{r \in R\} \geq b$ if and only if there exists an $x \in X$ such that $f(x) \geq b$.

From the exercise, all we have to do is exhibit a case, namely a pair of values (A, P), such that $h_{\mathsf{Opt}}(A, P)/h_{\mathsf{Alg}}(A, P) \geq 2$. Do we already know of such a case, or rather even two of them? The invalid argument in Section 3 gave us two possibilities, namely that one of A and P is 0 and the other one strictly positive, and so we have shown that the competitive ratio is at least 2.

Now comes the difficult part of the analysis, that is to upper bound the competitive ratio of Alg. Similar to the lower bound, we need to re-express the assertion into a form that is simpler to reason about. You should again think about this for a moment before you read the answer in the next exercise. As a hint, you should expect some notion of duality between what we need here, and what we have in Exercise 4.

Think about what it means to upper bound the competitive ratio

Exercise 5. Given a set of numbers $R = \{f(x) \,|\, x \in X\}$, show that $\max\{r \in R\} \leq b$ if and only if for all $x \in X$, we have $f(x) \leq b$.

Therefore, to upper bound the competitive ratio, we need to show that the ratio is upper bounded for *every* possible instance of $(A \geq 0, P \geq 0)$. The most direct way to tackle this is to use *inequality* reasoning, as in the following proof.

Consider an arbitrary instance (A, P) such that $A, P \geq 0$. In the special (boring) case that both are 0, clearly both Alg and Opt produce happiness of 0, and so the ratio is bounded by 2 (here, we completely ignore the fact that 0/0 is undefined). Now, for the general case that at least one is strictly positive, the ratio

$$\frac{h_{\mathsf{Opt}}(A,P)}{h_{\mathsf{Alg}}(A,P)} = \frac{\max(100A, 10P)}{50A + 5P} \quad \text{by Exercise 2 and by the definition of } \mathsf{Alg}$$

To show that this fraction is upper bounded by 2, we consider two subcases, where 1) $100A > 10P$ (and hence $50A > 5P$) and 2) $100A \leq 10P$. In case 1),

$$\begin{aligned}\frac{\max(100A, 10P)}{50A + 5P} &= \frac{100A}{50A + 5P} \\ &\leq \frac{100A}{50A} \text{ since } 5P > 0 \\ &= 2\end{aligned}$$

Similarly, in case 2),

$$\begin{aligned}\frac{\max(100A, 10P)}{50A + 5P} &= \frac{10P}{50A + 5P} \\ &\leq \frac{10P}{5P} \text{ since } 50A > 0 \\ &= 2\end{aligned}$$

Thus we have shown that the competitive ratio is upper bounded by 2, since the ratio is bounded for all instances (A, P).

5 Summary and tips

To summarise, when performing competitive analysis to show that the competitive ratio of an algorithm is equal to a number b, you need to:

- Prove that the competitive ratio is lower bounded by b
  - Relatively easy direction, essentially a "there exists" proof
  - Simply provide an instance that gives a ratio of *at least* b
- Prove that the competitive ratio is upper bounded by b
  - Trickier direction, a "for all" proof
  - Show that *all* instances give a ratio of *at most* b
  - Use inequalities and case analysis for reasoning that applies to *all* instances

These are two subproofs of completely different flavours, so make sure that you are comfortable with both, and that your homework submission and exam attempts follow the same rigorous approach.

We conclude with some concrete tips on proving inequalities, as part of the subproofs. The arguments presented for the running example are simpler than those required for the homework. To do the homework problems, it is important to remember the following basic inequalities:

- If $a_i \leq b_i$ for all i, then $\sum_i a_i \leq \sum_i b_i$.
- If $0 \leq a \leq c$ and $0 < d \leq b$, then $a/b \leq c/d$.

The second inequality tells us that in order to upper bound a ratio, we should upper bound the numerator and lower bound the denominator. Similarly, to lower bound a ratio, we should lower bound the numerator and upper bound the denominator. This will help you fill out the contents of the two subproofs required for a complete analysis.

A Solutions to exercises

Exercise 1: We define Alg to be $(n_A = 50, n_P = 5)$.

Exercise 2 : Given values of A and P, we are trying to maximise $n_A A + n_P P$ subject to the constraint that $n_A + 10n_P \leq 100$ (and implicitly, n_A and n_P are both non-negative integers). Our strategy is as follows, if $100A \geq 10P$, then take $(n_A = 100, n_P = 0)$, otherwise take $(n_A = 0, n_P = 10)$. The happiness generated by this algorithm is by construction $\max(100A, 10P)$.

To show that this is optimal, we proceed by doing a case analysis, depending on how $100A$ compare with $10P$. 1) Suppose $100A \geq 10P$, and consider an arbitrary *optimal* solution (n'_A, n'_P). Since $10A \geq P$, the solution $(n'_A + 10n'_P, 0)$ has at least as much happiness value as (n'_A, n'_P), and furthermore still satisfies the bag weight constraint. The solution $(n_A = 100, n_P = 0)$ is clearly at least as good as $(n'_A + 10n'_P, 0)$. 2) Suppose $100A < 10P$, and consider an arbitrary *optimal* solution (n'_A, n'_P). Since $P > 10A$, if $n'_A > 10$, then the solution $(n'_A \bmod 10, n'_P + n'_A \operatorname{div} 10)$ is strictly better than (n'_A, n'_P), which is a contradiction. Thus it must be that $n'_A < 10$. If $10 > n'_A > 0$, then $n'_P < 10$ by the weight constraint. Then $(0, n'_P + 1)$ is a feasible solution, which is strictly better than (n'_A, n'_P), resulting in another contradiction. Therefore, the optimal solution must have the form $n'_A = 0$, and clearly the best solution of that form is $(0, n_P = 10)$.

There is a slightly cleaner proof if we considered also non-integer values of n_A and n_P in the optimisation, and show that the optimal solution must still be $(n_A = 100, n_P = 0)$ and $(n_A = 0, n_P = 10)$ in the respective cases. The relaxed optimisation problem is an instance of a *linear program*, the theory of which will be introduced later in the course.

Exercise 3 : $a = b$ is true if and only if both $a \geq b$ and $a \leq b$.

Exercise 4 : "$\Rightarrow$": If $\max\{r \in R\} \geq b$, then let $x^* = \arg\max_x\{f(x)\}$, and so $f(x^*) = \max\{r \in R\}$. By definition, we have $f(x^*) \geq b$, and hence there exists an $x \in X$ such that $f(x) \geq b$.
"$\Leftarrow$": If there exists $x \in X$ such that $f(x) \geq b$, then $\max\{r \in R\} \geq f(x) \geq b$.

Exercise 5 : "$\Rightarrow$": Suppose $\max\{r \in R\} \leq b$, then consider an arbitrary $x \in X$, and we have $f(x) \leq \max\{r \in R\} \leq b$.
"$\Leftarrow$": Suppose for all $x \in X$, we have $f(x) \leq b$. Let $r^* = \max\{r \in R\}$, and there must be some x^* such that $f(x^*) = r^*$. Therefore $\max\{r \in R\} = f(x^*) \leq b$.