Assessing Computational Cost Using System Context

Models of a computer program that intend to determine runtime but depend only on the structure of the algorithm are limited and when used in isolation can paint an inaccurate model of the runtime cost of the algorithm when the algorithm is considered in the broader context of the system. Actually determining runtime cost requires knowledge of and a model of the input data and/or the number of times the code path will be invoked based on the conditions of the natural world the system integrates into.

The Theory

In essence the necessity for modelling not based solely on the classical application of asymptotics is self-evident if you yield the assumption that the notion of an effective procedure (as it is currently understood) does not capture the entirety of the dynamics of the structures that influence the course of occurrences in nature.

If you believe this then it follows that the challenge of software engineering is, at least in large part, the challenge of interpolating various forms of models to achieve a meaningful result, or in other words integrating the currently understood notion of an effective procedure into a natural world problem space not determined entirely by the properties which determine the behavior of the effective procedure. It follows then that mathematical analysis of the effective procedure alone will be limited.

It's not hard to get a sense of this by surveying open source repositories. TimSort is a great example. I won't list a series of example of effective procedure confined (asymptotic alone) modelling and modelling which accounts for the probable properties of the natural world conditions the computer system operates within here for the sake of time (I'm not actually an academic after all), but it's not hard to find many such examples.

Further, even if I could list N examples of models which account for the probable properties of the natural world conditions the computer system operates with here, a skeptic may still list N + 1 examples of effective procedure confined (asymptotic) models. In other words, the breadth of known models of either sort in practice doesn't validate my theory. In fact my theory is predicated on a qualitative basis.

The notion put forth here is a computing model of a system which is intended to achieve a desired result is most effectively formulated by interpolating models of non-"effective procedure" confined natural world conditions into the model to account for optimization *because* achieving the desired result is based on an outcome beyond the bounds of the effective procedure.

In other words the desired result of the computing system is qualitative and natural and not defined by it's own theoretical basis. This is easily established as true. Consider the notion of "responsiveness". For a system to possess the property of "responsiveness" it must not create a substantial delay in the execution of the operations the operator desires. Or, put conversely, if the system creates a delay in execution that the operator considers unreasonable otherwise creates a negative experience for the operator due to frustration it does not contain the property of "responsiveness". But what does the operator desire? What is the threshold for frustration? What makes an experience negative?

Well, take, for example, one common example of a useful modern computer system is a system designed to connect operators with their friends via the internet, or in other words, social media. A system which is responsive and designed to connect a user with friends via the internet must be designed to account for the likely behavior of the person using the system. How long do they typically pause on a thread of comments to read? What is the typical cadence of their communication amongst one another?

In other words, for a system to be responsive you have to know what delays are appropriate and what delays are not. Can an instant message sit in a queue for 1–2 minutes before being delivered? How about a "like" showing up as associated with a user's post after another user issued it? All of these questions depend on the user. Since there is no known model which proves 1:1 equivalency between humans and effective procedures as they are understood today there is no way to model, prior to building the system, the quality of "responsiveness". You just know it when you see it. And "you", a human, have to see it and consider it responsive, or responsiveness would make no sense.

In other words, it relies on a definition of the properties of the conditions of the natural world that are outside the bounds of the formal model the system. Now we can define some formal bounds for this system in an effort to codify the property of responsiveness based on a statistical analysis of common cases that are ubiquitous amongst operators. Say, "new posts must not take more than .5 seconds to show up in the users feed after they are 'posted'" or "instant messages must not arrive in the corresponding user's message box longer than 50 milliseconds after being 'sent'".

But these are not quantities determined by any known effective procedures. They are also not asymptotic properties. So you must study or at least define the properties of the natural world problem space in which the computer is intended to be used within to properly understand how to design a system that achieves a desired outcome, in this example "responsiveness".

And then, and only then, can you begin to understand where the “hot” code paths are in your system (those that are likely to be called frequently based on the conditions of the natural world problem space your system will be deployed to) and then you can use tightly defined asymptotic models to optimize your system in the context of where asymptotic optimization of tightly defined code paths we typically call “functions” is most useful / meaningful, but only after you’ve found the places in your code base that benefit from such optimization.

Asymptotically optimal algorithms everywhere all the time in a system does not necessarily create an optimal system. Modelling must account for the properties of the conditions of the natural world in which the system will be deployed.

With that said, if you value excessive formality and perfectly defined thesis over reality you can defeat my argument in generality by supposing that the desired result of a computer system is a proper alignment of the effective procedure in such a configuration that it satisfies a formal condition of asymptotic analysis, again confining the computer system to conditions within it’s own foundational model. There is no way to be more convincing in my argument here than to appeal to a priori assumptions that form through repeated attempts to build and deploy computer models in real world scenarios. In other words, experience building systems in reality.

Some people may find the above disappointing as what I put forth, if accepted, limits the utility of theoretical breakthroughs. But I’m not sure the limitations that one would imagine would be imposed by my theory here being meaningful would invalidate or delimit novel theoretical insights. It simply expands the bounds of complexity theory and effective runtime cost modelling beyond the foundational model of the computer and necessitates some interpolation between models of “effective procedures” and models of nature that aren’t perfectly aligned with the mechanisms of the systems we use today. Which is more interesting anyways and opens up new theoretical ground that is rich with potential for exploration.

Here’s an example another example from Python’s source code — Objects\setobject.c

Here’s Richard Feynman saying things I agree with that are similar to my arguments here — https://www.youtube.com/watch?v=B-eh2SD54fM

A Formal Example

Let me craft a formal example and make up some notation.

Consider the following tightly defined problem ...

Write a function that takes an input that is a string and returns the length of the longest substring without repeating characters.

Suppose that, for some reason, I want to invoke this algorithm one time for all the sentences in a series of books what I am provided as large .txt files from various publishers. Perhaps to be able to do some statistical analysis on the author's writing style.

Let's just start with the trivial $O(n^3)$ case, which you can read about on Leetcode above. I solved the problem myself and included the Python below, for fun, even though this is unrelated to this article.

```
class Solution:

  def longestSubString(self, s: str):
    if (len(s) == 1):
      return 1
  length = 0
    stack = set()
    for i in range(len(s)):
      if not (s[i] in stack):
        stack.add(s[i])
      else:
        if (len(stack) > length):
          length = len(stack)
        stack = set()

    if (len(stack) > length):
      length = len(stack)

    return length

  def lengthOfLongestSubstring(self, s: str) -> int:

    longest = 0
    for i in range(len(s)):
      curr_l = self.longestSubString(s[i:])
      if (curr_l > longest):
        longest = curr_l
    return longest
```

Ok cool. So there will be about $O(n^3)$ “operations” necessary to complete this effective procedure. If I was studying for my algorithms 101 class, great, I’m done. But suppose this effective procedure is one small part of a large system that will be deployed to the real world. Then… do I want to try and optimize it? I mean… what if it’s only called once or twice a day when the system hits the real world? Is that worth it?

Of course I understand this is an articulated example as it is not hard to see there is an asymptotically superior solution to this problem, but rather what I’m getting at here is that there are conditions you will find in practice where it’s not clear when optimization is useful, and figuring that out is important.

From that lens, the broader picture, $O(n^3)$ tells us very little. This bound tells us almost nothing about what will happen in the real world when we deploy the algorithm in the context of a system.

Now… suppose I want to use the insight I pointed out above. What I’m after is a more complete model of runtime cost associated with the algorithm based on a model of the natural world domain that shapes the data input into my program in the context of the system. For this I need to model the number of invocations of the algorithm I expect to occur over the course of it’s lifetime while it is deployed in my system based on the nature of the data coming into the system.

This is entirely new. So I need new notation. In this case let’s suppose the number of invocations depends on the number of books the system is intended to process over it’s lifetime (let’s just ignore the correlation between books and sentences since this should be linear). So I build a model that says something like $I(b) — O(n^3)$ (I just made up some entirely new notation here to denote a model of the number of Invocations (I) of the algorithm that performs linearly asymptotically against the size of the input each time it is invoked). In a sense I’ve extended asymptotics and tied it into a variable that models expectations of invocations based on domain.

Suppose in this case, the articular example above, we don’t have a fixed quantity for the number of books our publishers intends to submit to us which we need to process (say this depends on many variables they have trouble forecasting themselves) and we are simply exposing an API to them to hit at their will with no rate limit. They themselves simply don’t know how many books they will have to process. It’s one of the unknown human variables we have tons of trouble effectively forecasting.

But since we need a model (we have no certain limit from our publishers and we don’t want to delimit them but we want to know approximately how “hot” this part of the system will be to know how much time we should spend thinking about how to optimize it) let’s suppose

we can just use some other models from industry that have been demonstrated to be accurate enough based on historical data. These models show we can roughly anticipate the number of books we will process as a function of employed writers of our publishers. In practice you might just use your gut to come up with this.

So we can easily figure out the number of writers employed by all the publishers we have agreed to work with. And we have at least a model of the number of books a publisher produces based on the number of writers employed. Because of network effects and good old teamwork the relationship is exponential

b = books produced over a given time frame
w = number of writers employed by the publisher
b = 3^w

This is a bit of a silly example, but again, it's just for the sake of exposition.

So here things could get interesting. Because I just sucked in some additional contextual information outside the immediate context of the structure of the algorithm I have a much better model and it shows the runtime cost, when considered in the context of the nature of the problem space, is actually completely different than I expected when I was only able to observe the asymptotic analysis (because of the locality with which it was constrained to and because it didn't model the size of the input).

The asymptotics only painted a small part of the picture, but the model of asymptotic runtime in conjunction with the model of number of invocations based on the nature of the natural world domain producing the requests or data paints a completely different picture of the actual runtime cost of this portion of the system.

Remember my model from above? I(b) — $O(n^3)$. b is initially an unknown, and something that is easy to forget to consider while developing a system in reality, (b is the number of books, or invocations of the algorithm, that we expect to be requested over the lifetime of the systems deployment).

But, recall what we just modelled above? Now we can actually determine b as well with a function based on a known quantity, w, (the number of writers employed by all of our publishers).

b = 3^w

So the model becomes I(3^w) — $O(n^3)$ and over the lifetime of the system the actual in aggregate cost of the operations of the algorithm scale exponentially based on the number of writers employed by our publisher. So we have an exponential invocation asymptotic hot spot in our system.

Hmm, now it makes a lot more sense to spend some time trying to drive down that $O(n^3)$ cost. In fact it's super important. We can't really and don't want to change the I(3^w) cost, presumably, since it's based on our customers and likely tied to our revenue, but we do want to drive down the $O(n^3)$ cost, since, if that were better, it would just be better, period. There is no downside to having a more effective $O(n^3)$ algorithm doing that processing. In fact the system might not functional at all during times in which high volumes of books are being processes if we don't figure out a better algorithm.

Lucky for us there a O(n) algorithm to solve this problem. That's a huge improvement. And if you modeled it the way I did above you can demonstrate why that improvement actually mattered for the company and was necessary and important.

In this case the asymptotics of that particular algorithm alone is of limited relevancy in terms of it's actual runtime cost for the system. Nonetheless, that algorithm is embedded in a potential hot spot in this system — it will be invoked based on an exponential relationship with a variable we know, and we might even know will increase rapidly in certain cases! Even though the algorithm is $O(n^3)$, there's something else we can look at that shows that it will actually be invoked on an exponential scale based on a condition of the natural world the system will be deployed within that we couldn't, at the time, effectively model with another algorithm.

The exponential growth relation was outside the context of the algorithm, but it doesn't matter to the system when that's what determines the number of operations the system performs. The runtime model had to account for something that wasn't the structure of the algorithm itself that changed the runtime drastically, in this case into a completely different "asymptotic" class.

How much of a difference could I make if I accounted for this in my system design?

Let's wave our hands and say the average number of characters in a sentence is 88 and the average number of sentences in a book is 5000.

8^{83} x 5,000 = 681,472 x 5000 = ***3,407,360,000 operations*** *per book for this algorithm alone*

Let's say for the duration of the deployment of the system containing the algorithm that determines the length of the longest substring without repeating characters in each sentence, we know there are a total of 10 writers employed by all of our partner publishers (so w = 10).

I(3^w) — $O(n^3)$: precisely ^{310} x 3,407,360,000 = 59,049 x 25,981,560,000
= ***201,201,200,640,000 operations***

Now let's suppose I had done this sort of "invocation asymptotics" modelling, discovered the hotspot, and found the linear time algorithm for the longest substring without repeating characters problem.

88 x 5,000 = ***440,000 operations*** *per book for this algorithm alone*

I(3^w) — O(n): precisely ^{310} *x 440,000 =* ***259,621,560,000 operations***

In this case we saved ~200 trillion operations over the life time of the system if we were able to reduce the asymptotic runtime of the function from $O(n^3)$ to O(n). I know this is an arbitrarily large number, but the point is the pay off of this asymptotic improvement can be modelled and justified against the backdrop of the conditions of the natural world problem space in which the system was deployed.

In Summary

This style of modelling accounts for a completely different type of complexity analysis that could be useful in practice: algorithm development in the context of the broader system considered by a model in conjunction with domains of the natural world that don't have tightly constrained input spaces.

This style of modelling, which I suppose I'll call "invocation asymptotics", or whatever, allows practitioners to actually determine aggregate runtime cost at design time in a way that could change the design of the system significantly. If you know what parts of the system are worth applying asymptotic analysis to, you know how to create a more effective system based on it's intended use case. You can ignore optimization problems that don't really need to be solved and focus on the ones that do.

Thus you have a new model of runtime cost based on the nature of the domain in which you deploy the system and the duration in which you expect the system to live in that domain (the invocation asymptotic model) that empowers you to optimize spend on ingenuity and apply critical engineering resources only to specific hot spots that are likely to be problem areas for the system, and ignore ones that won't.

If you think this way you also encourage your engineers to consider the lifetime of the system and the customer more deeply so they can build their invocation asymptotic model in the first place.

Probably this is already done commonly in practice. In the pre-amble I cited some examples of algorithms which have been shown to perform optimally based on the characteristics of the input data (Timsort is the most common I can think of, and Objects\setobject.c from the cpython source), which I'm considering as a variation of the type of modelling I'm describing here not based on anticipated invocations but qualities of

anticipated data. I expect these types of modelling are equivalent as you can simply describe a model of the conditions of the natural world that produce the data as an invocation model and use that to shape the analysis of the system and identification of hot spots.

Of course, maybe someone will describe my ideas in here in more academic / theoretical clarity at some point, I simply can't for lack of time and because I face the pressures of industry. Nonetheless it's a very interesting idea, no?