Computation in Finance: Potentials and Limitations

Edward Tsang
Centre for Computational Finance and Economic Agents (CCFEA)
University of Essex, UK
3 November 2010
Working Paper WP047-10

Abstract

One major flaw in classical economics is that computational costs are ignored. It is assumed that agents will be able to make the optimal decision given sufficient time. Unfortunately, combinatorial explosion limits the capacity of computation. Its impact is often underestimated. Besides, the cost of computation is often ignored.

1. Classical economics completely ignores computation and computational time

In classical economics, rational decision makers are assumed to be able to make optimal decisions given the available information. This assumption is flawed. From a computational point of view, optimization is not a "solved" problem. There are many problems for which optimal solutions are yet to be found. These will be elaborated below.

A reasonable interpretation of "perfect rationality" is that the decision maker is capable of making all possible inferences given the available information. It is important to note that computation costs time. It is difficult to see how anyone can possibly make all inferences within a reasonable amount of time – if one can, then chess will become an uninteresting game.

It is also important to note that two perfectly rational decision makers do not necessarily take the same amount of time to come up with the optimal decisions. The time difference in arriving at the same decision may determine the outcome of their interaction. For example, the first agent to pick up an investment opportunity may complete a deal before the other agents can act.

Furthermore, even if perfect rationality is assumed, computational costs cannot be ignored. Computation costs may include the costs for collecting data, developing algorithms, implementing the algorithms, purchasing and setting up computer hardware, etc. Such costs should be included in economic models. This will be discussed below.

2. Basic computing helps

The first application of computers in finance was in replacing mundane calculations. Back in the 1970's, computers started to replace manual

bookkeeping. As computers were expensive, only large institutes such as banks could afford them.

In the 1980's, spreadsheet emerged to be a useful tool. It changed many people's ability to handle complex arithmetic models. At the same time, databases gained popularity. The continuous fall of computer storage costs enabled companies to build data warehouse in the 1990's. Large amount of data were stored for sale. Users started to use historical data for research. The more sophisticated researchers would write programs to clean up data, present them in useful forms, or use data for analysis.

Basic computing helps data processing. Basically, it enables people to handle large amount of data, and handle them fast. Having databases and spreadsheets certainly gives one advantage over competitors who handle data manually. Naturally, anyone who knows how to extract information out of data will have a competitive edge over those who do not. In the next section, we shall look at some examples.

3. Advanced Computation could help more, some examples

The availability of data enabled more sophisticated analysis. One of the most relevant branches of computer science is machine learning. Typically, machine learning attempts to find patterns from data. This is often achieved by searching in the space of possible patterns. An example will be given later. The patterns found could be used for understanding the market or forecasting.

Another branch of computer science that can contribute to finance beyond data processing is modelling and simulation. Modelling can help us to simulate individual entities in the system based on the assumption of self-interest. In computational finance, the researcher typically explores possible models, to find models that synthesize data that assimilate data in the real market. (Characteristics of such data are called stylized facts.)

Constraint satisfaction and optimization deal with decision problems. The aim is to satisfy multiple constraints and find the best solutions according to any criterion (objective) given. As rational decision-making entails making the optimal decisions, constraint satisfaction and optimization techniques simulate rational reasoning. In fact, if agent A uses more advanced constraint satisfaction and optimization techniques than agent B, then A could be said to be effectively more rational than B (see Tsang's CIDER Theory).

4. Evaluating a trading rules, a case study

Suppose you want to assess the following trading rule:

Rule I:

(a) Whenever the short-term moving average crosses the long-term moving average from below, buy;

(b) Whenever the short-term moving average crosses the long-term moving average from above, sell.

Our focus here is not whether this is a sensible rule or not. Our focus is on the cost of computation if we want to examine this rule.

One way to assess this rule is to compute the m- and n-days moving average, where m < n, and then check it against historical data to see whether this rule earns or loses money, and how much.

To properly examine this rule, one should try different values for m and n. The performance of a rule is likely to be sensitive to their values. A particular m value may work better with some n value, but not others.

Suppose, for simplicity, you decide to test m with a value between 1 and 20, and n with a value between 21 and 70. In this case, you have 20 possible values for m, and 50 values for n. Therefore, you have (20 × 50 =) 1,000 combinations of (m, n) pairs to evaluate.

Suppose each (m, n) takes 1 second to evaluate. It will take 1,000 seconds, or 17 minutes to examine the above rule. This is probably acceptable if you are dealing with daily closing prices.

5. Considering a more complex rules

Suppose you suspect that different (m, n) values would work for buying and selling rules. To assess such hypothesis, you need to find a (m1, n1) pair for a buying rule, and a (m2, n2) pair for a selling rule. It is not possible to evaluate a buying rule on its own, because whether it makes money or not depends on when the holdings are sold. As different buying rules work better with different selling rules, you need to evaluate every combination of (m1, n1) and (m2, n2). Since there are 1,000 combinations for (m1, n1), and 1,000 combinations for (m2, n2), you need to evaluate (1,000 × 1,000 =) 1,000,000 combinations.

As before, if we assume that each evaluation takes 1 second, then it will take 1,000,000 seconds, or 115 days to complete the evaluation. This can be reduced to approximately one day, if one uses 115 computers.

6. Pushing the boundary forward in rules examination

Simple rules could be found by others. In order to beat your competitors, it is desirable to find more complex rules. Suppose you decide to look for rules that relate stock with index prices. Suppose you will only buy (or sell) if you see crossing in both the stock's moving averages and the index's moving averages. Suppose you allow the crosses to take place within a few days. Without this allowance, there will be too few opportunities to buy or sell.

Following is one way to formalize the above idea. Let k-MA_s be the k-days moving average for stock s, and k-MA_I be the k-days moving average for index I. We could examine the following buying rule. The selling rule can be defined similarly.

Rule II for buying:

(a) m-$MA_s \leq$ n-MA_s on day d, but m-$MA_s >$ n-MA_s on day d+1

(b) m-$MA_I \leq$ n-MA_I on day d', but m-$MA_I >$ n-MA_I on day d'+1

(c) $| d - d' | \leq D$

Part (c) of this rule allows the crosses to take place within D days. Suppose you allow D to take a value between 0 and 9.

To evaluate all possible buying rules, you need to evaluate 1,000 combinations of (m, n) pairs. This means you need to evaluate (1,000 × 10 =) 10,000 combinations of (m, n, D). The same number applies to the selling rules. Suppose we need 2 seconds per evaluation, as we are now dealing with two series instead of one. Therefore, time required to evaluate all the combinations is (10,000 × 10,000 =) 200,000,000 seconds. That is approximately 63 years. Unless one has super-fast computers, this is clearly unaffordable.

7. Combinatorial explosion haunts – introduction to complexity

The above example is artificial. But it demonstrates a fundamental phenomenon in computation. Every extra factor we consider, the number of combinations grows significantly. In fact, it grows exponentially. This is called "combinatorial explosion" in computer science.

Combinatorial explosion is the reason why computers cannot find the optimal moves in chess yet. That is also the reason why passwords work: if a password has 6 characters, and each character can take one of 62 values (A to Z, a to z or 0 to 9), say, there are 62^6 = 56,800 million possible combinations.

A significant part of computer science is about how to contain combinatorial explosion, or to extend our ability to handle more combinations. Some algorithms have been studied for their complexities. Here we shall limit our discussion to the Big O notation, which expresses the worst-time complexity. An algorithm that explores all the possible combinations of m and n in Rule I in a brute force manner will have a complexity proportional to m×n. In Big O notation, we say that the algorithm that evaluates all the combinations has a worst-time complexity of O(m×n). Or, if n is always a multiply of m, then we can say that the worst-time complexities for this algorithm is roughly (the technical term is "asymptotically") and $O(m^2)$. When m increases linearly, the increase in run-time is m^2 for this algorithm. We call this algorithm has polynomial run-time.

A polynomial time algorithm may still take too long to run, as we have demonstrated in Rule II above. However, it is not as bad as an algorithm that attempts to crack a password by brute force. If a password has r characters, and each character can take a value from d values, (we used r=6 and d=62 in the above example), then an algorithm that evaluates all possible combinations has a worst-time complexity of $O(d^r)$. As r grows, the run-time of this algorithm grows exponentially. An algorithm that has polynomial run-time is considered to be "tractable". An algorithm that has exponential run-time is considered to be "intractable".

For some problems, no tractable algorithms have been found; chess for example. By exploiting characteristics of a problem, tractable algorithms could be found for non-trivial problems. Finding a shortest path from A to B is a good example. (That is why satellite navigators can update routes in real time.) However, a slight change in the specification of a problem could completely change the complexity of an algorithm. Sometimes, it is not obvious whether a problem has a tractable solution not.

The Big O notation only refers to the worst situation. Another widely used complexity notation measure is the average-time complexity, which uses the Big Ω notation. The Simplex method in linear programming is a good example: although it has exponential time complexity, it finds the solution in linear time in most cases.

8. How to handle combinatorial explosion?

When a problem is intractable, compromises have to be made. Often, one goes for approximations. Approximations are no good for the password problem, but it is good for chess, for example. Approximations are often found by heuristics. Heuristics are rules that work intuitively or statistically, but they are not guaranteed to find optimal solutions. Neither do they guarantee to find good approximations every time.

For an optimization problem, a search that guarantees to find the optimal solution is called a complete search. Naturally, an algorithm that exhausts all possibilities is complete. Some complete algorithms could use heuristics to avoid searching parts of the space that guarantee to contain no solutions. This could significantly save run-time, though normally it does not change the worst-time complexity of an algorithm.

When no tractable complete search can be found in practice, stochastic search is often used. Stochastic search refers to algorithms that rely on chance to find solutions. Most of these algorithms use heuristics, or use feedback from the search history. These algorithms will be elaborated later in the module.

9. Knowledge representation – a deeper issue in computation

It is worth noting that combinatorial explosion is not the only difficulties in computing. Knowledge representation is another source of difficulties. In the above example, we have demonstrated how an idea can be written down as rules. Writing down the rules is one way to channel human knowledge into computer programs. Computation cannot start before ideas are properly formulated. Some ideas are more difficult to formulate than others. For example, representing the head and shoulder chart pattern is harder than representing the rules above. Modelling the decision making process of a trader is extremely difficult, even with the help of the trader. Knowledge representation is a fundamental issue in artificial intelligence.

10. Including information costs in economic models

Perfect rationality is a basic assumption in classical economics. But what exactly does perfect rationality mean? One might attempt to define perfect rationality as the ability to make the optimal decision under the information available. However, this definition assumes that the amount of information is given fixed. In reality, information can be gathered. By having more relevant information, one could be expected to make better decisions. Note that financial data cost money to collect. Data processing (in order to extract information) costs money; this cost could be huge if research is required. Therefore, a decision maker (with or without perfect rationality) must be able to decide whether and how much to invest in data collection and how much to invest in data processing before deciding what decision is optimal under the information available.

To model economic decision-making properly, one should include the cost of data collection and information processing. Unfortunately, it is very difficult for a decision maker to know how much he/she can benefit from the information that he/she is yet to gather. Therefore, it is very difficult to model an agent's decision on information investment.

11. Concluding summary

In classical economics, agents are assumed to be able to make optimal decisions. This assumption is flawed, at least from the computation point of view. Due to combinational explosion, finding the optimal solution requires more time than anyone can realistically afford. If you have better algorithms and better heuristics, then you can find better solutions than your competitors (Tsang 2008). Besides, information costs (which include the cost of computation) should be part of economic models. It is difficult to do so, but we should be aware of what we are leaving out.

Bibliographical remarks

Above, we referred to machine learning, modelling and simulation and constraint satisfaction and optimization as advanced computing. Some people would refer to them as techniques in artificial intelligence (Barr et al 1981) (Russell & Norvig 1995) or computational intelligence (Kordon 2010). Artificial intelligence and computational intelligence are basically different terms given to two vague, overlapping sets of computational techniques. The boundaries of both sets are vague and evolving. The two terms sometime used by different communities. There are attempts to define and differentiating these two terms, but debates on definitions are often unconvincing or not productive in this case.

References

[1] Barr, A., Feigenbaum, E. & Cohen, P.R., The handbook of artificial intelligence, Vols.1&2, Morgan Kaufmann, 1981

[2] Head and shoulders chart pattern, http://en.wikipedia.org/wiki/Head_and_shoulders_(chart_pattern)

[3] Kordon, A., Applying computational intelligence: how to create value, Springer, 2010

[4] Russell, S. & Norvig, P., Artificial intelligence, a modern approach, Prentice Hall, 1995

[5] Tsang, E.P.K., Computational intelligence determines effective rationality, International Journal on Automation and Control, Vol.5, No.1, January 2008, 63-66

[6] Tsang, E.P.K. Book Review, on "A.K. Kordon, Applying Computational Intelligence, Springer 2010", IEEE Magazine, May 2010, 108-109