Analysis of the Limits of Computation through Unconventional Models of Computation

Vivek Sharma Advisor: Richard Molnar

May 11, 2000

Abstract

Our current paradigm, a combination of the Universal Turing Machine and the Von Neumann architecture has allowed us to define certain "limits" in computing. In this paper, we examine the bio-molecular and quantum computation models as possible alternatives to our dominant way of thinking.

Are our current notions of "limits" universal across models? Is our current view of computing ideal for solving "hard" problems? Are problem classes, as we understand them today, universal across other models of computation?

We suggest answers for these questions and use the insights gained to build a more robust understanding of the concept of "computation."

Contents

1 Introduction

It is widely accepted that there are certain problems that are "easy" to solve and there are certain problems that are "hard" to solve. These problems classes are loosely based on the ability of modern computers to provide solutions in a given time and space. We are also faced with limitations in modern computation on a physical level. Our current dominant paradigm, the universal Turing machine and the Von Neumann architecture, has allowed us to build faster and more powerful computers. It is becoming quickly obvious, however, that there are physical and theoretical limitations inherent in our current paradigm.

Is our current view computing ideal for solving "hard" problems? Are problem classes, as we understand them today, universal across other models of computation? In this paper, we will investigate two major alternative "unconventional" models of computation to analyze these issues. We will examine the bio-molecular computation (BMC) and quantum computation (QC) models and show the natural division into problem classes that occurs. Using the differences and similarities in the models discussed, we suggest a more robust understanding of the "limits" of computation.

2 Turing's mathematician-clerk

The major fall-back of the Universal Turing Machine (UTM) is that it describes a process that is inherently sequential, static and linear. The clerk who silently "computes" on a piece of paper is hardly impressive, yet this is the basic notion of computation as we know it. A computer is nothing more than a mechanized version of Turing's clerk which ponders over digits represented digitally in random access memory.

A UTM's basic sequential abilities are inadequate in dealing with certain types of problems. When it comes to large data sets or problems that have a large solution space, a UTM suffers from inefficiency [7]. We also see that this deficiency in the model to handle large solution spaces is reflected in the complexity classes.

A logical improvement over the classic UTM is one that incorporates multiple tapes or parallelism. This however, only improves the time-scale of the UTM in a linear fashion. As the problem space increases in size, the amount of parallelism required to efficiently determine an answer also increases. Furthermore, this enhanced UTM still does not provide a solution to the problem of increased memory space. Even if we were able to build super-parallel classical computer, unrestricted by hardware bottlenecks, it would soon become obvious that space and time constraints limit which problems we can solve.

The UTM does, however, provide us with the notion of *computability*. It is clear that these alternative models cannot violate the Church-Turing thesis. Any basic facility made possible in QC and BMC can be simulated with many parallel Turing machines. But as soon as we start asking questions of *efficiency*, our notion of complexity changes completely as we move from model to model.

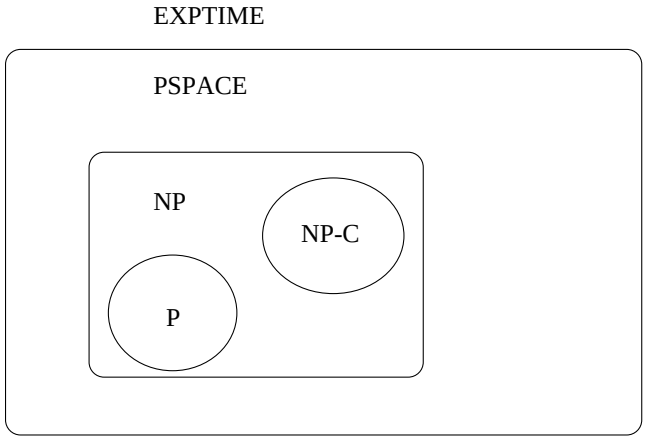

Figure 1: Complexity classes

Later, we will try and compare the limits of this dominant paradigm to those of BMC and QC. This will be crucial in determining a more 'universal' view of feasible computation.

3 Complexity and classes of problems

Here we refer to the classical notion of complexity. Our modern view of "easy" and "hard" problems is rooted in the Turing Machine's ability to determine solutions in a certain time and space [13].

In Figure 1, we see the hierarchy of classes as they are understood today. The UTM model allows us to work within the polynomial time class (P). Even super-parallel classical machines, however, cannot solve problems higher than this class in an efficient manner. We will use the framework of this hierarchy, which tells us what is "hard" and "easy", in our analysis of the alternative models. A complete background on complexity can be found in [7] and [13].

An important questions arises: is this discussion of complexity really appropriate? This theory might not make sense under the realms of "unconventional" models. This is partly the point, however, as we are trying to develop a more robust way to characterize solvable problems. An investigation of complexity is essential as this is the best-defined way of viewing modern "limits" of feasible computation. There are certain classes of problems that cannot be solved in a reasonable amount of time under our dominant paradigm. However, this same imposing "limit" could potentially be bypassed under a different model of computation.

A discussion of complexity leads also nicely into an analysis of problem classes. It may not be always appropriate to discuss complexity for certain "unconventional" models, but it may be the case that there are certain sets of problems that are better viewed under a non-classical paradigm.

Furthermore, complexity theory is independent of the physics of a given model. That means that we do not necessarily classify problems based on implementation-level tendencies of models. A problem that is inefficient for a classical Turing machine remains inefficient for all classical Turing machines regardless of hardware improvements. Similarly, our notions of complexity for alternative models remains the same regardless of technological advances.

The other compelling reason for an inclusion of complexity theory is that time and space will always be factors in our understanding of "computation." Unless there is way to bypass time and space completely, there will always be motivation to analyze problems in terms of how *fast* they are and how much *space* they require.

There are certain exploitable patterns, however, in the classification scheme of efficiently and non-efficiently solvable problems that become evident. These patterns can be used to define appropriate models for certain types of problems. In the next section, we discuss a complexity class of particular interest; the NP-complete class of problems.

We will also see later that these problem spaces seem different for BMC and QC. Our current way of visualizing spaces might be correct, but the actual limits and bounds change as we move from model to model.

3.1 The NP-complete class of problems

Our paper will mainly deal with the NP-Complete class of problems. This area is of particular interest to Computer Scientists as it allows us to ask crucial questions about the relationship between P and NP classes. Most theorists will claim that the existence of the NP-complete class is enough evidence that $P \neq NP$ [13] [7]. This arises from the notion that inherently, all NP-complete problems are reducible. That is, when we solve one problem from this space in efficient time, we can solve all problems in the space in a similar manner.

This tendency is also due to the similar construction of the problems. There is a solveable, definable methodology for every problem in this space, but none that solves every problem efficiently. Since NP-complete problems can all be solved through an exhaustive trial (explained further below), once we solve one problem efficiently, we can reduce other problems in the class to the efficient one and solve them in reasonable time as well.

From our current paradigm, it is seen that anything short of an "oracle" will not solve every given problem efficiently from NP-complete space. Our best method in approaching these problems generally is an exhaustive search. It is possible to find shortcuts to solutions for some problems but there is no *fast* method for efficiently solving every problem.

An exhaustive search, however, is not so undesirable if we are given powerful enough tools. We do not necessarily have to explore an entire solution space for a given problem. If unreasonable solutions can be eliminated right away or are never fully considered, our exhaustive search becomes manageable.

This is exactly why this problem space is relevant to the context of alternative models Simply put, BMC and QC offer us an incredibly powerful tool to

solve NP-complete problems. Their ability to massively parallelize[1] searches through a solution set provokes a new analysis of complexity theory and specifically the NP-complete class of problems.

We briefly referred to an improvement over the classical UTM. Parallelization can drastically improve the time-scale of a problem. It can be seen that models that exploit parallelism are inherently better suited to solve exhaustive search problems.

The NP-complete class of problems are also of interest to us because a subtle range of problems can be identified within this class. Our main area of focus in the context of BMC will be combinatorial problems such as those that employ graph traversal and decision trees. For QC, we will be mostly interested in problems where *probabilistic* and randomized techniques work best. These include factoring and function solving problems.

It can be noted that both models offer one similar strength—mainly, the ability to massively traverse a solution set. After a discussion of the two models, we will examine whether this alone guarantees that the entire NP-complete class of problems can be 'captured' into the same class by these models.

In summary, there is no general algorithmic method for efficiently solving all given problems in NP-space. There are underlying models, however, which can exploit the "reasonable" exhaustive nature of certain problems. These models are not only able to redefine the boundaries of complexity classes, but also extend our notion of "computation".

This will become a key concept when we examine BMC and QC.

4 Bimolecular computation

First demonstrated by Adleman [2], bio-molecular computation promises two powerful advantages; efficient data storage along with massive parallelism and DNA molecule complementarity. We will see how bio-molecular computation (BMC) can be used to solve combinatorial graph path and coloring problems. We will also see that these results can be extended to other problems in NP-complete space. Finally, there will be a discussion of the types of problems that this model is best suited for.

4.1 Basic ideas and methods

The nature of bio-molecular computation arises from a Deoxyribonucleic Acid (DNA) molecule. A *DNA computation* consists of encoding information in the DNA molecule, separating out erroneous encodings, and using chemical-bond based operations to conduct a simultaneous search over a given problem space. A basic discussion of the characteristics of DNA are provided below.

DNA is composed of polymer strands of connected nucleotides. There are four possible nucleotide bases: adenine (A), guanine (G), cytosine (C), and

[1] This parallelization is on a level that outpaces classical computers—for example, 2^n inputs can be explored in just a few steps in BMC or in one step in QC.

thymine (T). Bonding takes place when two such strands come together and form a larger strand. Current research is heavily focused towards using DNA, however, it is important to abstract away to the possibility of using RNA instead. It might also be possible in the future to use other molecules/chemical combinations to compute. We focus on the actual algorithms and operations rather than draw on this implementation-level distinction. In section 8 we will examine this in the context of current limitations and possible work-arounds.

We refer to the details of the bonding process in the discussion of the actual computing steps. The two fundamental characteristics of BMC are:

1. Massive parallelism of DNA strands and their storage capability,
2. Watson-Crick complementarity.

In detail, we see that:

Parallelism and storage As discussed in section 3.1, most of the intractable problems can be solved by an exhaustive search. A large amount of information can be stored in a DNA strand. When coupled with the ability to make multiple copies of the possible solution space, we get an efficient way to carry out an exhaustive search. Since every possible solution is "considered" at the same time, we can effectively eliminate incorrect solutions in one step. This parallelism is different from classical machines, where sets of solutions are distributed over a cluster—but the cluster size does not necessarily relate to the input size.

Watson-Crick complementarity DNA strands can only form together if their corresponding bases match by pair-wise attraction. A will always bond with T and G with C. This is known as complementarity. This "feature" becomes essential in developing a mathematical basis for BMC. Our template molecules will only match with complementary molecules—if we encode the template molecule with a specification for a problem (can be viewed as constraints), then generating an entire solution space allows us to find correct solutions in this one 'matching' step.

We will use these two advantages to show how BMC can be used to tackle the problem spaces discussed in Section 3. We develop the necessary tools for an analysis of BMC algorithms in the following section.

4.1.1 Operations on DNA

Every algorithm (or series of computational steps) is based on operations carried out in some sequence. In DNA, the classical notion of "operations" such as storing and fetching data in registers is replaced by chemical-bond based reactions and results. There are five essential physical operations necessary:

Merge This stage combines the contents of two containers (usually test tubes) into one.

Detect Using radioactive labeling, the detection of DNA strands in given container.

Synthesize Using automated lab techniques, a synthesizer sequentially connects up lined up molecules. The molecules can be pre-lined manually or using a automated laboratory aid.

Anneal This operation requires the length operation given below. All 'imperfect' DNA strands have to be removed — the details are available in [29].

Length This is needed to ascertain if merged strands have the correct length — this provides a way to eliminate incorrect or erroneous solutions.

In addition to these steps, we also need mathematically based operations which we can use to develop a theory that is independent of lab techniques. These are shown in the following:

remove S This operation removes, in parallel, any strings that contain the substring S.

union This operation merges a given set of strings into a multi-set.

copy U This takes a multi-set and produces a given number of copies.

select U This simply selects an element of the multi-set U at random and returns it — if the set is empty then empty is returned.

These operations might seem foreign and quite distant from the classical notion of operations on a machine. Analogies can be drawn, however, between the UTM and BMC. The UTM's ticker-tape computation is a series of marking and deleting operations on a tape that contains encoded information about a problem. Similarly, the DNA (or RNA) is the medium on which manipulations are made to "compute".

One major distinction to be made, however, is that the traditional view of computers as static, immutable objects falls away when we discuss both BMC and QC. This

In the next section, we see how these operations are combined to form the steps of a bio-molecular algorithm.

4.1.2 DNA based algorithms

A summary of Adleman's Hamiltonian path problem is provided here as an example of a BMC algorithm. In the next section we will further extend this discussion and deal with a traditionally "hard" classical problem.

We start with a seven vertex directed graph that represents the travel path between cities [Figure 2]. A Hamiltonian path is one that involves every vertex exactly once (in Figure 2, route $1-2, 2-3, 3-4, 4-5, 5-6$ is Hamiltonian). This implies that $v_{in} \neq v_{out}$.

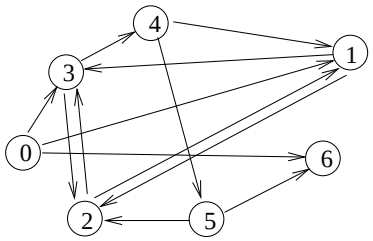

Figure 2: Adelman's test graph of 7 nodes

The Hamiltonian Path Problem (HPP) simply asks whether any given graph has a Hamiltonian path. To consider the entire graph, it is obvious that a an exhaustive search is necessary. Although algorithms exist that check for special preconditions, none can avoid the exponential worst case time for an arbitrarily given general graph [32].

Adleman's technique is given below:

Input: A given directed graph G with n vertices.

1. Generate all paths in G randomly in large quantities.
2. For all paths that do not begin with v_{in} and end in v_{out}, reject using the operation *remove*.
3. For all paths that do not contain all n vertices, reject using the operation *remove*.
4. For each of the n vertices v, reject all paths that do not involve v [32].

Since we reject all invalid paths, the remaining molecules represent all valid paths for the given graph. Adleman demonstrated this on a fairly small case, but it can be seen that this could be extended to larger cases without any modification.

As all these steps are done a large set of generated possible solutions (essentially the entire solution space), we see exactly how massive parallelism can convert an inefficient algorithm into a polynomial time series of steps.

Furthermore, the second important aspect—Watson-Crick Complementarity, is used to eliminate paths that are not in the graph G (or conversely, maintain paths in G).

The essential points here are that an inherently nondeterministic classical algorithm can be still be used in BMC—once wildly inefficient, the massive parallelism acting on the entire solution space can drastically speed up computation.

4.2 Using BMC to break the Data Encryption Standard

A powerful application of BMC is seen in the field of cryptography. Given a BMC's ability to generate the entire solution space in parallel allows for the search for keys for a given encryption scheme [11]. The Data Encryption Standard (DES), one such scheme, was recently shown to be vulnerable under the strength of a BMC [3].

Recent classical parallel machine attacks have been shown to crack the DES scheme—this, however, does not imply that this discussion is fruitless. Under traditional computing, it took 90 and 36 days to crack the first and second challenge (offered by RSA Security) in a distributed fashion to crack a specific encrypted plain text message. Using BMC, less than one gram of DNA can generate every key possible in the encryption scheme. The only hindrance to a speedy solution has been the lack of automated laboratory techniques. Whereas classical machines would need a significant increase in parallelization to handle larger keys (greater than 56 bits), BMC is scalable regardless of the solution size and would yield the answers in the same amount of time. A top-level view of this development follows.

DES relies on a two-way system—the encryption happens under a 56 bit key, which is used both to encrypt and decrypt information. The only complete approach to finding this key is an exhaustive search (same as the brute force method as employed by current classical approaches). Since DES uses a 56 bits, we would have to search through 2^{56} possible keys. As described in [11]:

1. Build the entire possible solution key space (using the physical operation *synthesize*.
2. On each strand that represents a memory complex, compute the cryptotext obtained by encrypting the known plaintext by the key represented by the memory complex [32].
3. Select the memory complex whose cryptotext matches the known cryptotext and read its key.

It is easy to imagine this being done a classical parallel setup—the 2nd step would be distributed over a cluster of machines (or processors) with a given section of the problem space. An analogy to BMC would be if we had one processor devoted to each key. This is where the real strength of the computation model lies. Regardless of the problem size, we are able to consider every possible solution.

As we can see, the brute-force massive parallelization of BMC can be used to solve intractable problems of arbitrary size in reasonable time. Next we will see a different type of parallelization being used in QC to solve a similar problem.

5 Quantum computation

Quantum theory is a relative newcomer to the realm of physics. Since it describes basic behaviors of atomic and sub-atomic particles, it becomes relevant

in all forms of physics-related discussions. In computer science, however, this theory was restricted to being referenced in terms of physical hardware contexts such as that of nano-technology and materials science. Only recently, however, have scientists begun to understand that quantum phenomenon can be used in the simulation and representation of classical computation.

A basic primer is given before we introduce quantum algorithms and complexity issues. For a detailed history of quantum computation see [23].

5.1 Building a quantum computer

The basis of building a quantum computer is the *qubit*. This term refers to a physical system that can be arranged in one of the two states commonly labeled as $\mid 0\rangle$ and $\mid 1\rangle$ or in a superposition of these two states. It is because of this ability to be in multiple states at one time, that a QC can store many inputs.

Why is the qubit important? Simply put, it is a mechanism which is analogous to the state-based classical computers of today. Our dominant paradigm is based on two states: on and off (or high and low current). The qubit is a powerful extension of this. Not only can we represent two separate states, we can represent any linear combination of these two states in one qubit. This in turn leads to massive parallelism on a scale that can not be imagined in classical computers.

The intricacies of this are seen in [23] and [15]. We will focus, instead, on the effects of the physical processes of quantum mechanics. These will be reflected in efficient algorithms for classically intractable problems. There are, however, two important concepts to note:

Interference This refers to the superpositioned states—incorrect 'answers' cancel themselves out, leaving only correct solutions [23].

Probability A random exploration of a problem space can be beneficial. A QC's overall state fluctuates between many possible configurations. This in turn leads to true randomized behavior, which can then be exploited in trying numbers for factoring of exploring a solution space.

How can computation proceed when a multi-state system is inherently non-deterministic? This is true to an extent, but the real promise of QC comes when we allow randomness into our computations. This is referred to as probabilistic computation. In the section given below, it will be shown how exactly a quantum system "computes."

5.2 Quantum algorithms

It can be seen that quantum algorithms require a different mind-set than traditional ones. The major distinction is that a QC returns singular solutions—meaning that algorithms that return a set of solutions or values have to be adapted to produce a singular value.

Furthermore, what is a QC really "computing"? The idea that a probabilistically correct solution is the only one left after a quantum computation is very different from a traditional computer. A QC is really a method finding the resulting state of a probabilistically correct computation. This "state" is conceptually identical to the term found in traditional computer science. This representation of a correct solution as a "state" can be adapted to fit quantum algorithms.

In detail, we see that qubits can be arranged in registers—with each register being represented by a function that describes its behavior [15]. A classical computer will compute inputs one at a time (in general), however, each register in a QC can be "held" to contain a superpositioned number of configurations—these configurations are the collective inputs to a computation. The resulting "state" that we discussed previously is result of the interaction of these registers.

In the next section, we examine these concepts with a specific algorithm.

5.3 Using QC to efficiently factor an integer

The problem of efficiently factoring an integer arises from number theory as well as classical problems as key-based encryption. Modern day techniques are highly inefficient as the only method we know of is one that tries every number possible. This provides the strength in public-key encryption as modern day computers are not fast enough to factor large integers. Quantum computation, on the other hand, is superb at probabilistic computation. We try different numbers until the right ones are found. Furthermore, as a direct consequence of interference, incorrect answers cancel themselves out, leaving only probabilistically correct ones.

The actual algorithm is a combination of Euclid's technique and wave function analysis. The wave function analysis comes in because to get the "answer" at a given step, we determine the period from the mapping generated by a superpositioned qubit [15].

Shor's algorithm is as follows:

1. Consider the function:

$$f(x) = y^x (mod N), \quad for\ x = 0, 1, 2, 3, \ldots, \tag{1}$$

   where $a(mod\ b)$ is the remainder of a/b.

2. Take random values of y and compute for $x = 0, 1, 2, 3, \ldots,$

3. Check the period (T) of the function $f(x)$

4. For the greatest factor of the number N, compute:

$$z = y^{(\frac{T}{2})} \tag{2}$$

Let the greatest divisor of $(z + 1, N)$ be p and let q be the greatest divisor of $(z - 1, N)$. Both p and q are factors of N.

Clearly, the actual "computation" is in the stage where we find the period T of the function $f(x)$. This is done as described in the previous section—we assemble registers such that each register represents the combined inputs for the function $f(x)$. Upon combining the registers, we entangle states in a way such that the periodicity amplitudes of the qubit states represent probabilistically "correct" periods [15].

This method describes a way to find two factors of a number if the quantum computation finds the period correctly. The error range of this algorithm has recently been shown to be accurate to 60%-95% accuracy.

What does these calculations imply? Since the factorization of relatively large integers composed of the product of two primes seems to be an intractable problem, we can see that QC poses a significant efficiency gain. Modern techniques are able to compute step 3 in the given algorithm, but because of the nature of QC, we are able to do this much faster than classical machines. Modern day cryptography systems rely on the limited abilities of modern computers to build security systems. This perceived security, however, is completely breached with the power of a QC.

Not only is this a striking result for modern security schemes, it is also an indicator of our UTM biased notion of complexity. Using information gained from the ability of QC to solve intractable problems, we re-think our view of complexity.

6 Comparisons between the models

The similarities between the two models discussed are immediately noticed:

1. Massive parallelism
2. Both models address similar types of problems

The inherent capabilities of either model to perform exhaustive searches allows us to drastically increase efficiency. This is mainly a consequence of the parallelism available at the base physical level in either model.

This similarity in turn leads to a discussion of the exhaustive search NP-complete problems solved by these models. We will follow this in the next section.

The differences in BMC and QC can be seen in three major areas:

1. Physical processes
2. Randomness in QC and brute force in BMC
3. Space/physical requirements

The first item is immediately clear. BMC uses biologically-based molecules to try a massive search through the problem space, while QC uses the "states" in physical systems to probabilistically arrive at solutions. The inherent differences

in the underlying "architecture" of the models are great, yet both models yield similar results. Furthermore, this dissimilarity at the physical level reflects itself in point 2. The algorithms for BMC are more in line with traditional step-based way of thinking of computation. In QC, however, we see probabilistic algorithms in which the real "computation" or solution finding is done in one step (through superpositioned states).

A similar extension can be drawn when discussing the space and physical requirements for computation in these models. It is natural to link problem input size to actual number of molecules required in BMC. For large problem sizes, the amount of DNA required grows to unreasonable sizes. Drawing along Adleman's experiment, it can be seen that for large problem sizes, the initial generation of the problem space becomes a bottleneck.

A partial solution to this would be to design better encoding/generation schemes such that physical molecule characteristics are further exploited to use less space. It would also be worthwhile to explore the possibility of "evolutionary computing" where problem and solution space is dynamic over the span of a calculation.

In QC, however, the relative problem input sizes have a limited impact on physical considerations. The more observable states there are, the better the chances of a probabilistically correct solution.

We discuss physical considerations in section 9, while we continue the discussion of problem spaces in the next discussion.

7 Towards a more robust notion of the "limits of computation"

As discussed before, our current view of computation is highly biased towards a notion of a sequential linear mechanical device. A more encompassing view of computation would admit the probabilistic and massively parallel models that we have discussed. A more robust view would also partition problem spaces based on techniques that solve them best (efficiently).

At first sight, it may seem impossible to develop an 'absolute' view of complexity. A view that is independent of a basic model seems flawed as the notion of computation is inherently based on a underlying descriptive model. This might, however, turn out to be another 'biased' view—non-traditional techniques and their corresponding algorithms could show that there are other ways of viewing computation. It is possible, however, to classify problems based on their distinctive patterns and the time/space dependencies of specific models.

We can construct a class based view of models of computation in which these goals are met by mixing complexity theory and what we have discovered about "unconventional models." It is obvious that the UTM model is appropriate for solving problems in P space. Perhaps a more stronger view of "limits of computation" would separate models based on appropriate problem spaces. The UTM might be a practical model for the P class, but BMC and QC may turn

out to be appropriate models for classes outside of P.

We have also seen that using the massively parallel unconventional models, we are able to efficiently solve some problems in NP-complete space. Combining our knowledge of these differences, we can build a broader view of "limits." We discuss this in the next section.

7.1 The NP-complete 'shift'

The discussions of BMC and QC seem to indicate that there is a separate problem space that corresponds to each of these models. We have seen how each provides techniques for an exhaustive computation over the space of all possible solutions for specific NP-complete problems. It is also seen, however, that these models only capture part of NP-complete problems. As we move from model to model, the space of NP-complete problems seems to change size as a specific model captures problems in that space.

This 'shift' can be used as evidence that there is a well defined boundary between P and NP spaces. If we were able to solve one NP-complete problem in P time, we would be able to solve all NP-complete problems in P time. As the current research shows, we are not far from this. The algorithms presented in sections 4 and 5 are theoretically on the border of NP-complete space. If we extend these algorithms such that they solve a NP-complete problem in P time, we will have the ability to solve all problems in the class NP.

This 'shifting' of problem spaces from one model also indicates that perhaps it is better to classify problems differently for each model. In other words, it might be useful for us to view certain problems under QC and BMC but traditional problems might be better suited under the UTM. We explore this in the next section.

7.2 BMC and QC solvable intractable problems

Both the models we have discussed thus far offer massive parallelism on a scale that would support decision problems. BMC also supports traditional combinatorial graph problems. QC takes us a step further and provides the theoretical ability to "crunch" numbers in a relatively short period of time. These different type of problems can be classified broadly into the area which classical computer science defines as intractable exhaustive search based problems.

Extending these problem classes into separate model-based classes could be one possible way to think about "computation." This would also allow us to add more models and classes as we learn about them. Abstracting away from the idea of a sequential UTM based theory of computation to one that is encompassing of "alternative" techniques is necessary to solve and/or understand some of the really "hard" problems.

Following this notion, we build classes for QC that contain problems solved best in a non-deterministic probabilistic manner. We define the class QP to contain all problems solveable in polynomial time for a QC.

For BMC, we include problems where combinatorial techniques are more appropriate. These would contain graph problems as well as decision based problems. Again, we define a class BMP to contain all problems solveable in polynomial time for a BMC.

The questions now arises: does $BMP = QP$? Also, is the class NP-complete contained in QP and BMP? Clearly, both QP and BMP contain a subset of NP-complete. If we follow the reducibility argument, we can say that NP-complete is contained in QP and BMP. This would seem to imply that NP-complete $\subset (BMP \cup QP)$.

Current research indicates that QC and BMC are indeed capable of capturing most NP-complete problems. The relationship between the models given above should not, however, imply that $BMP = QP$. It might turn out that there are problems in QP which cannot be in BMP and vice versa.

It can be seen that the inherent parallelism in both these models differs. In QC, we do not necessarily generate solutions and then act on them in parallel as in BMC—rather, a probabilistic algorithm 'investigates' possible answers as a direct consequence of the superpositioned states in a qubit system. Interference also is necessary as incorrect guesses or solutions are picked up as wave-function peaks that cancel out over time as the computation evolves. This limits QC, as we understand today, into problems where this method can be smoothly applied.

A graph traversal problem would be inherently more complicated in a QC as we would have to build a complete system to encode, and decode states as a part of the problem. Theoretically, if it can represented in terms of a probabilistic computation, a QC would be able to handle it. This separation in problem spaces implies that perhaps there are problems that solvable efficiently in one model or the other, but not both. Looking at the pattern that emerges from current research, it is probable that QC computation is specific to certain definable problem which employ techniques such as wave-analysis. The quantum Fourier transform (QFT) is another area of investigation for QC.

Furthermore, working on the side of QC, we can see that resource use might be a limiting factor to BMC. Whereas the space requirements of a QC are not related to the size of the input, in BMC they are directly related. This might provide further classification of problems into sets which are manageable in PSPACE for BMC (of course, only further research can show the real requirements for a BMC) and which problems fall outside of this domain.

All of these ideas support a view of computation as a "mixed-bag" of techniques rather than one overall standard methodology that applies to everything. Even though current research has shown very few physical demonstrations or implementations of BMC and QC, there is a lot of evidence that supports a shift from our dominant viewpoint to a more encompassing view. This view does not necessarily have to be absolute, but rather open to the existence of better techniques.

We continue the discussion of physical considerations started here in the next section.

8 Physical realities

An extension of Adelman's demonstration showed that in order to have graph nodes greater than 20 nodes requires a substantial number of DNA molecules. A graph of 200 nodes was shown to require enough molecules to outweigh the Earth. These practical limits are not absolute, however, as recent studies have shown that manipulating the molecules in a 3-dimensional fashion yields more information storage [25]. Even this advancement cannot override the amount of molecules necessary for a exponential time problem in which the size of the inputs grows equally fast in relation to the solution space. The practical reality is that even though we might capture some problems in PSPACE, we hit the space barrier.

Furthermore, current laboratory techniques are inadequate for fully automating the operations necessary for DNA algorithms. A lab worker is required to watch the computation and manually merge and sort results. It is also clear that there are resident error factors in current techniques that increase in magnitude as the solution space increases.

For quantum computation, the problem arises in holding qubit states in interaction-free long observation periods. Interaction is the phenomenon in which observational techniques "sway" the results or state of a physical system.

Quantum processes are also highly delicate and volatile. The requirements for assembling even a single qubit system are enormous when compared to classical computers. With the advancement of materials sciences, however, it is potentially feasible that we will have the tools to build specific purpose quantum computers. An area of research that is especially promising is that of low-energy super-conductors. With these materials, it may be possible to build low interaction, low energy quantum devices that have observation periods which last long enough for complex computations.

The concept of a dynamic, ever changing computational artifact, where the actual "computer" changes (and is destroyed) as it computes is also different from our current paradigm. We tend to think of computers as immutable standalone devices, but BMC and QC challenge this embedded belief. Perhaps it is really better for some types of tasks to view computation as being a one-time process. Traditional rationale would dictate that the state of the machine can be "stopped" and "restarted" (taking a snapshot of a TM, for example). For these alternative models, however, this way of thinking does not really apply. The one-shot computer might offer a rich way of viewing "computation" as more than a static process.

All of these points about current physical limitations should not be discouraging. These difficulties are similar to those faced by early computer designers. The initial experiments with computing machinery were doomed to failure as the technology was not available to produce precise parts. There was also no facility for automation available to the designers. As soon as this technology was available, however, computers became more and more feasible to build. A similar phenomenon is likely to take place in the future for BMC and QC.

9 Conclusion

It is clear that our concept of computation is highly dependent on underlying models and methods. The resident ticker-tape Universal Turing Machine model of computation provides a solid base for one mathematical theory of computation, but it restricts us to a small set of problems solvable in "reasonable-time". What if we were able to solve an *unreasonable* problem in *reasonable* time? We would have to redefine our limits to accommodate this. Further research in unconventional models, especially ones that challenge us to question our dominant notions of time and space as resources, will only lead to a more robust concept of computation.

It cannot be stressed enough that the main reason for pursuing these "unconventional" models is to build a new framework for understanding computation. Even if we never successfully build a quantum computer, we might be able apply some of the techniques into classical machines. Exploring these models also allows us to ask probing questions about our current knowledge level.

We have also seen that our theoretical understanding of limits of computation are not universal. It is feasible in the near future to break the Data Encryption Standard (for larger keys) as well as solve the Hamiltonian path problem in polynomial time. These two problems only serve as small examples from a space of many other UTM based "hard" problems. Reconsidering them under alternative models shows us that our conceptual understanding of "limits" needs to be extended.

Furthermore, we should not abandon the hope that in the future, we might be technologically advanced enough to construct BMC's and QC's. Our current research will only speed the process of truly defining practical limitations in computation.

Acknowledgements

This research, although still a work in progress, would not have been possible without the insight, wisdom and support of Prof. Richard Molnar of the Macalester Mathematics and Computer Science department. I would also like to thank Prof. Susan Fox and Prof. Dan Schwalbe for their help in shaping this paper. In discussing "Unconventional Models of Computation", we have step outside of our traditional modes of thinking—a task that would not have been possible without the guidance of the aforementioned people.

References

[1] Leonard M. Adleman. On Constructing a Molecular Computer. In *DNA Based Computers*. Based on Manuscript, Computer Science Department, University of Southern California, January 11, 1995.

[2] Leonard M. Adleman. Molecular Computation of Solutions to Combinatorial Problems. *Science*, 266:1021–1024, November 11, 1994.

[3] Leonard M. Adleman, Paul W. K. Rothemund, Sam Roweis, and Erik Winfree. On Applying Molecular Computation To The Data Encryption Standard. In *Proceedings of the Second Annual Meeting on DNA Based Computers, held at Princeton University, June 10-12, 1996.*, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, May 1996.

[4] Gordon Alford. Explicitly Constructing Universal Extended H Systems. Unconventional Models of Computation, pages 108–117. Springer, 1998.

[5] M. Amos, A. M. Gibbons, and D. Hodgson. Error-resistant Implementation of DNA Computations. Research Report CS-RR-298, Department of Computer Science, University of Warwick, Coventry, UK, January 1996.

[6] Martyn Amos, Steve Wilson, David A. Hodgson, Gerald Owenson, and Alan Gibbons. Practical Implementation of DNA Computers. Unconventional Models of Computation, pages 1–18. Springer, 1998.

[7] José Balcázar, Josep Díaz, and Jaquim Gabarró. *Structural Complexity I.* Springer, Berlin, DE, 2nd edition, 1995.

[8] Eric B. Baum. DNA Sequences Useful for Computation. -, June 1996.

[9] Eric B. Baum and Dan Boneh. Running Dynamic Programming Algorithms on a DNA Computer. In *Proceedings of the Second Annual Meeting on DNA Based Computers, held at Princeton University, June 10-12, 1996.*, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, June 1996.

[10] Gennady P. Berman, Gary D. Doolen, Ronnie Mainieri, and Vladimir I. Tsifrinovich. *Introduction to Quantum Computers.* World Scientific, Singapore, first edition, 1998.

[11] Dan Boneh, Christopher Dunworth, and Richard J. Lipton. Breaking DES Using a Molecular Computer. In Eric B. Baum and Richard J. Lipton, editors, *DNA Based Computers*, volume 27 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science.* American Mathematical Society, 1996.

[12] Richard A. Brualdi. *Introductory Combinatorics.* Prentice Hall, Englewood Cliffs, NJ, second edition, 1992.

[13] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1997.

[14] Russell Deaton and Max Garzon. Thermodynamic Constraints on DNA-based Computing. Computing with Bio-Molecules, pages 138–152. Springer, 1998.

[15] Arthur Ekert and Chiara Macchiavello. An Overview of Quantum Computing. Unconventional Models of Computation, pages 19–44. Springer, 1998.

[16] Richard P. Feynman. Computing Machines in the Future. Feynman and Computation: Exploring the Limits of Computers, pages 225–240. Perseus Books, 1998.

[17] Richard P. Feynman. Simulating Physics with Computers. Feynman and Computation: Exploring the Limits of Computers, pages 133–154. Perseus Books, 1998.

[18] Lance Fortnow. One Complexity Theorist's View of Quantum Computing. Mar 9.

[19] Lance Fortnow and John Rogers. Complexity Limitations on Quantum Computation. *JCSS: Journal of Computer and System Sciences*, 59, 1999.

[20] Rudolf Freund and Valeria Mihalache. Molecular Computations on Circular and Linear Strings. Unconventional Models of Computation, pages 201–217. Springer, 1998.

[21] Yuzhen Ge, Layne T. Watson, and Emmanuel G. Collins Jr. Genetic Algorithms for Optimization on a Quantum Computer. Unconventional Models of Computation, pages 218–227. Springer, 1998.

[22] Tom Head. Hamiltonian Paths and Double Stranded DNA. Computing with Bio-Molecules, pages 80–92. Springer, 1998.

[23] Richard J. Hughes. Quantum Computation. Feynman and Computation: Exploring the Limits of Computers, pages 191–224. Perseus Books, 1998.

[24] Neil D. Jones. *Computatbility and Complexity: From a Programming Prespective*. MIT Press, Cambridge, MA, first edition, 1997.

[25] Nataša Jonoska, Stephen A. Karl, and Masahico Saito. Graph Structures in DNA Computing. Computing with Bio-Molecules, pages 93–110. Springer, 1998.

[26] Zhuo Li. Alegbraic Properties of DNA Operations. Computing with Bio-Molecules, pages 327–339. Springer, 1998.

[27] Alexandru Mateescu. Splicing on Routes and Recognizability. Computing with Bio-Molecules, pages 314–326. Springer, 1998.

[28] Alexandru Mateesu. Splicing on Routes: a Framework of DNA Computation. Unconventional Models of Computation, pages 273–285. Springer, 1998.

[29] Mitsunori Ogihara and Animesh Ray. The Minimum DNA Computation Model and Its Computational Power. Unconventional Models of Computation, pages 309–322. Springer, 1998.

[30] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading, MA, 1994.

[31] Gheorghe Păun. Distributed Architectures in DNA Computing Based on Splicing: Limiting the Size of Components. Unconventional Models of Computation, pages 323–335. Springer, 1998.

[32] Gheorghe Păun, Grezgorz Rozenberg, and Arto Salomaa. *DNA Computing: New Computing Pardigms.* Springer-Verlag, Berlin, DE, first edition, 1998.

[33] John H. Reif. Paradigms for Biomolecular Computation. Unconventional Models of Computation, pages 72–93. Springer, 1998.

[34] Arto Salomaa. Turing, Watson-Crick and Lindenmayer: Aspects of DNA Complementarity. Unconventional Models of Computation, pages 94–107. Springer, 1998.

[35] P. W. Shor. Polynomial time algorithms for discrete logarithms and factoring on a quantum computer. *Lecture Notes in Computer Science*, 877:289+, 1994.

[36] Micheal Sipser. *Introduction to the Theory of Computation.* PWS Publishing, Boston, MA, 1997.

[37] Colin P. Williams and Scott H. Clearwater. *Explorations in Quantum Computing.* Springer, New York, NY, 1998.

[38] Erik Winfree, Xiaoping Yang, and Nadrian C. Seeman. Universal Computation via Self-assembly of DNA: Some Theory and Experiments. In *Proceedings of the Second Annual Meeting on DNA Based Computers, held at Princeton University, June 10-12, 1996.*, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science., pages 172–190. American Mathematical Society, May 27, 1996. Draft.