INFORMATION STORAGE AND PROCESSING IN BIOLOGICAL SYSTEMS

GIULIA IPPOLITI

Department of Bioengineering

McGill University, Montreal

July 2020

A thesis submitted to McGill University in partial fulfilment of the requirements of the degree of Doctor of Philosophy

Table of Contents

List of Figures

List of Tables

Abstract

As the impending limits in the development of semiconductor devices – Moore's law – are going to be hit eventually, many computational problems gaining importance in modern societies are yet to be solved. The context of two of such problems, one independent from the other, is as follows: (i) Today, given the ever-increasing computing power, cryptographic systems rely on true random numbers to encrypt large amount of private data and protect secrecy from malicious attacks. To generate true randomness, computers, being strictly deterministic machines, must obtain entropy from external physical processes, hardly compatible with digital systems. (ii) Due to the enormous implications of solving NP complete problems efficiently, e.g., replicating human intelligence, such problems have attracted considerable attention and yet, a brute force polynomial time algorithm was never discovered in favour of the P = NP statement.

The research presented here discusses the mathematical principles and state of the art in the field of network computing with biological agents, which provides a novel computing model to tackle NP-complete problems. Network computing with biological agents, which for convenience will be referred to as 'network computing' throughout this work, harnesses millions of autonomous bacteria (biological agents) to explore a microfluidic network in parallel, where the network encodes a problem of choice. Here, the scaling related key technological challenges of a network computing device are identified in terms of chip fabrication, readout reliability and energy efficiency. The estimated computing time of massively parallel or combinatorially operating biological agents is then compared to that of electronic computers when solving instances of an NP-complete problem. Finally, the same network-based technology is demonstrated to also function as an entropy source generating random numbers out of the stochastic motion of bacteria in the network, which proved suitable for the development of a true random number generator. Possible weaknesses, solutions and 'auxiliary' technologies that, if adopted, would enable the device to scale successfully are identified. The novel results obtained provide a road-map to future developments and applications in the field of network computing.

Acknowledgements

First and foremost, I would like to thank my advisor Prof. Dan V. Nicolau for his help and support over the time of my Ph.D. pursuit. He was able to identify and understand my needs, as a student and as a person, and to provide me with the right tools and work context for me to successfully complete my journey. He gave me the chance to study topics of personal preference which, in this once in a lifetime experience, was fundamental for me to demonstrate and exploit my skills and eagerly learn new ones.

I would like to thank other members of the team for their contribution to the personal growth I experienced over the time of my Ph.D. pursuit. They all have been for me an example to follow as well as source of help and collaboration. In particular, I would like to thank Dr. Ayyappasamy Sudalaiyadum Perumal for his invaluable experimental work. We often collaborated and complemented each other's work, where I was involved in the computational and theoretical side of our projects. I would like to thank Dr. Falco van Delft for teaching me about new perspectives and approaches to science. Last, but not least, I am greatly thankful to Prof Dan V. Nicolau Junior for his knowledgeable and useful advices.

Contribution to original knowledge

This work includes a number of elements that can be considered distinct contribution to knowledge and original scholarship. Given the novelty of network computing, a thorough analysis of the current ‘state of the art’, challenges, scaling capabilities and computing performance of existing and ‘ought to be’ devices in the field was developed for the very first time. In particular, an innovative perspective was offered by comparing the performance of classical, quantum, and network computing, the latter with and without bacterial division, when solving an NP-complete problem of increasing complexity. The presence of division in the network as the computation proceeds represents, for the first time, the demonstration of computing resources increasing in time to match those required to solve a computationally hard problem. The impact of this achievement is demonstrated here. Finally, the suitability of a network computing device for generating high entropy data, subsequently post-processed by existing technologies to produce cryptography secure true random numbers, was established. The proposed true random number generator is currently a unique example of a hardware-software hybrid device built around a network computing model.

Contribution of Authors

None of the chapters of this research thesis benefited from external contribution with the exclusion of Chapter 2, 3 and 4. For what concerns Chapter 2, the contribution to authors is as follows:

FvD	conceived the run mode scaling models, the read-out nomograms and Tag&Trace,
GI	programmed and carried out numerical scaling simulations,
DVN Jr.	conceived the SSP bio-computing method from mathematical principles,
ASP	collected data concerning bacterial motility and energy consumption,
OK	modelled parallel networks and created 3D drawings,
SK	modelled stitching methods for optical microscopy readout,
SWH	collected data concerning microscopy methods, Field-of-View and resolution,
DVN	calculated energy consumption, initiated and coordinated the project,
All authors	contributed to work planning, to data interpretation, to writing the manuscript and gave final approval for publication.

For what concerns Chapter 3, the contribution to authors is as follows:

GI	programmed and run simulations and emulations of the computing device and electronic chips respectively, generated the data plotted in Figure 13, run regression analyses for data extrapolation and wrote the chapter material
DVN	derived the relationship between MIPS and problem cardinality

Finally, for what concerns Chapter 4, the contribution is as follows:

GI	conceived the random number generator design and the object recognition algorithm, developed the full code base of the model, performed image processing, derived statistics, tested the full output and wrote the manuscript
ASP	run experimental session required to generate stochastic behaviour and performed image processing

Chapter 1

Introduction

Given the current limitations of silicon-based computing, alternative technologies have recently emerged to tackle key computational problems in modern societies. Among them, network computing proposes an alternative parallel-computation system in which a given problem is encoded into a graph, or network, that is embedded in a nano- or micro-fabricated planar device. The network is explored by a large number of independent biological agents to solve the problem in a parallel fashion. The research presented here aims to assess the suitability of a network computing device to solve two model problems and to effectively scale with the size of the problem to be solved. This section is going to (i) discuss current limitations of silicon-based computing, (ii) develop understanding of alternative technologies and (iii) layout the architecture and operation of the network computing device.

1.1 Limitations of Silicon Based Computing

This section discusses limitations of modern silicon-based computers from the perspective of chip design, size, energy consumption and computing time, which depends on the inherent mode of operation of the chip.

1.1.1 The Decode Bottleneck

Since its conception, the ‘von Neumann’ architecture [1] provided foundation to all future generations of modern computers. Its core components are a Central Processing Unit (CPU) consisting in a Control Unit (CU) and an Arithmetic and Logic Unit (ALU), a memory segment and I/O subsystems. The CU interprets data and instructions fetched from memory and coordinates system operations while the ALU aggregates data and performs logical and arithmetical operations, such as AND, OR, addition and subtraction. The sequence of instructions decoded and executed by the CPU is referred to as instructions stream [1]. The CPU executes every instruction stream sequentially. As a distinguishing feature, the von Neumann architecture uses a single

memory unit for both data and instructions, as opposed to the Harvard Architecture [2] which separates the two in different physical spaces. In general, today we use the term 'von Neumann computers' referring to computers operating sequentially, hence executing a single sequence of instructions; that is the typical computers available today [1]. The performance of von Neumann based computers suffers of profound limitations in the 'one instruction at a time' execution paradigm, also named *decode bottleneck*. In fact, in a sequential mode scenario, even if two operations are recognised as independent, they are still executed in sequence [3]. Improving the CPU performance is a complex task. One may start from increasing the clock frequency - measured in cycles per seconds (Hz) – and consequently the processor speed. However, technological limitations prevent unlimited clock rate increase and instructions are still executed one at a time. Moreover, some instructions are more complex than others and take longer execution time: as a result, all instructions are limited by the slowest instruction. Today one key feature introducing some level of instruction parallelism is pipelining in modern CPUs. A pipelined implementation takes advantage of the independent actions needed to execute a single instruction and breaks each instruction into subunits to be executed by a different set of circuitries within the CPU [3]. In this way, if an instruction takes considerable time, a shorter instruction may start getting executed on a different CPU subunit. In theory, a *k* stage pipelining would provide approximately a speed up a factor of *k* over a non-pipelined system but in practise this is only possible to the extent that one instruction does not depend of any 'nearby' executed instruction [4]. In real world situations, the next instruction may not always be executed in the next clock cycle resulting in various types of hazards. Among them, most common are *data* hazards, when a planned instruction cannot execute due to its dependence on data not yet available and *branch* hazards where, due to a branch in the execution path, it is not known which instruction shall be the next one to execute. The existence of such hazards makes actual real-world pipeline performance far less than ideal and generates even more complications when longer pipelines are used [3].

1.1.2 Multiprocessing

Other than instruction parallelism, a speed up on computing performance can be achieved with data and task parallelism. Such methods involve distributing data and code to execute across multiple processor cores. High-performance computing today mainly relies on multi-core scaling

as a principal strategy for performance growth, thriving to meet increasingly demanding industry standards [5]. It is expected that, given N parallel units and a sequential system running the same algorithm with sufficiently large inputs, one can simulate the parallel system on the sequential system by dividing its computing time by N. Nevertheless, this assumption does not consider the overhead involved to start and orchestrate multiple processors simultaneously. Amdahl's law [6], a fundamental tool to analyse performance as a function of parallelism (it provides a theoretical speedup of the latency of the execution of a program as a function of the number of processors executing it), highlights this constrain. Consequently, a high degree of parallelism may not provide any substantial improvement compared to modest parallelization, due to much larger overhead; the inherent sequential nature of computer internal operations puts a bound to massively parallel computation.

1.1.3 The "Utilization Wall"

Additional obstacles to improving modern electronics arise from the perspective of energy limits. In 2013, the International Technology Roadmap for Semiconductors (ITRS) [7] highlighted the management of system power and energy as the main challenge of current integrated circuits. In 1974, Dennard's scaling theory [8] showed how to proportionally reduce dimensions of transistors characteristics to enable subsequent size shrinks and improvements in density, speed, and energy efficiency. Since then, every year transistor count has doubled, and frequency has increased by 40% on a constant chip area. Since Dennard's law started losing practical relevance due to size limitations, sustaining doubling transistors every generation came at an increasingly high cost of poor transistors switching speed and energy efficiency [5]. Under this dramatically slowing trend, the question of whether multicore scaling will be able to effectively provide doubling transistors and sustain performance improvements remains open. According to a detailed analysis [5] projecting upper bounds achievable with multicore scaling, regardless of chip organization and technology, multicore scaling is power limited and it is only possible if cores are slower, simpler and less utilized with each additional generation. According to this view, modern CPUs seem to be approaching a 'utilization wall' where the power consumption of transistors available reduces more slowly than their size, which is subjected to Moore's law [9]. Consequently, both transistors energy and size concerns are putting a strict limit to the future

manufacturing of modern integrated circuits. Such alarming perspective encourages research in alternative hardware and technologies.

1.1.4 New Boundaries: Information and Complexity

In addition to aforementioned limitations concerning energy, time, space and design, current electronics have proven unsuitable to tackle increasingly central problems in modern societies, hence highlighting new boundaries. The computation of non-polynomial (NP) time problems in polynomial time and the generation of truly random numbers are among them. Within the space of existing limits to computation, solving NP problems and generating true random number extend across dimensions of time, space, information and complexity [9].

1.2. NP-complete Problems

This section introduces NP-complete problems and their importance. It discusses existing attempts of alternative technologies, i.e. DNA, quantum and microfluidic computing, to develop polynomial bound algorithms of an NP-complete model problem.

1.2.1 Definition and Importance

To appreciate the principles of complexity theory, one must understand how to measure execution time of a computer algorithm. Such evaluation of running time, or time complexity, is computed with respect to the input length. In asymptotic analysis [10], which analyses the limiting behaviour of algorithms, the *worst-case analysis* is made with respect to the longest input length among all possible inputs, while the *average case analysis* with respect to the average input length. An estimation of asymptotic bound is given by the big-O notation which only considers dominant terms and disregards constant factors. For instance, given that an algorithm takes time $f(n) = O(n^2) + O(n)$ which is indeed a function of the input size n, $O(n^2)$ being the dominant term on $O(n)$, the expression is equivalent to $f(n) = O(n^2)$. Bounds of the form n^c for c greater than 0 are called *polynomial bounds* and those of the form $2^{(n^d)}$ for d greater than 0 are called *exponential bounds* [10]. As the big-O notation shows, some problems are computationally harder than others,

and among them, some may never be computable by an efficient algorithm. The Turing machine, theorized by Alan Turing in 1948 [11], provides a useful model of computation to evaluate algorithmic complexity. Informally, the complexity class **P** (polynomial time) is the set of all decision problems that can be solved by an efficient algorithm, meaning they scale with worst-case polynomial time-complexity on a sequential and deterministic Turing machine. Instead, the complexity class **NP** (non-polynomial time) is the set of all decision problems that can *only* be verified in polynomial time-complexity by a sequential and deterministic Turing machine, and that are not practically solvable by any efficient algorithm, meaning they scale, in the worst-case, exponentially or worse. In particular, NP-complete problems [12] represent the hardest problems in NP, such that, if a polynomial time algorithm is found for any one of the NP-complete problems, all problems in NP will also be solvable by a polynomial time algorithm. Theoretically, only a nondeterministic Turning machine (NTM) whose execution is defined by an infinite number of branching parallel computations, may hope to efficiently compute a problem in NP; such a machine does not exist yet.

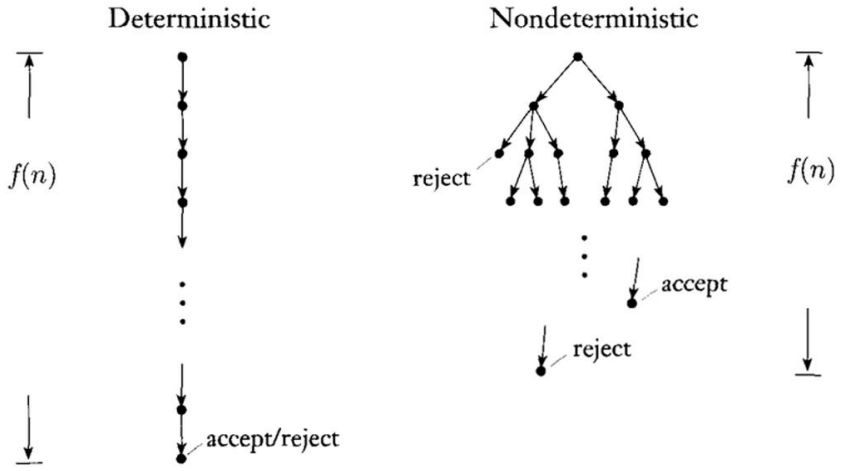


Figure 1: Turing machine**.** A deterministic Turing machine performs $f(n)$ steps in sequence. A non-deterministic Turing machine performs $f(n)$ steps in parallel branches corresponding to different possible computing paths. An end state over a given decision problem may be accepted or rejected.

A decision problem is assigned a complexity class based on the fastest known algorithm which can solve it. Therefore, one may hope to discover faster algorithms to assign problems in NP to

P. Certainly the P class is contained in NP, but the possibility of whether every problem in NP is also in P has never been proven. The two possible alternatives are illustrated in Figure 2. If ever it would be found that P = NP holds true, implications on our society would be enormous: suddenly, we would solve NP complete problems such as the Subset sum, the Travelling salesman and the Satisfiability problem [13], we would be able to compute an explosive number of combinations in a considerable forward time e.g., accurately forecast weather, we would be able to understand creativity and truly replicate human intelligence [14].

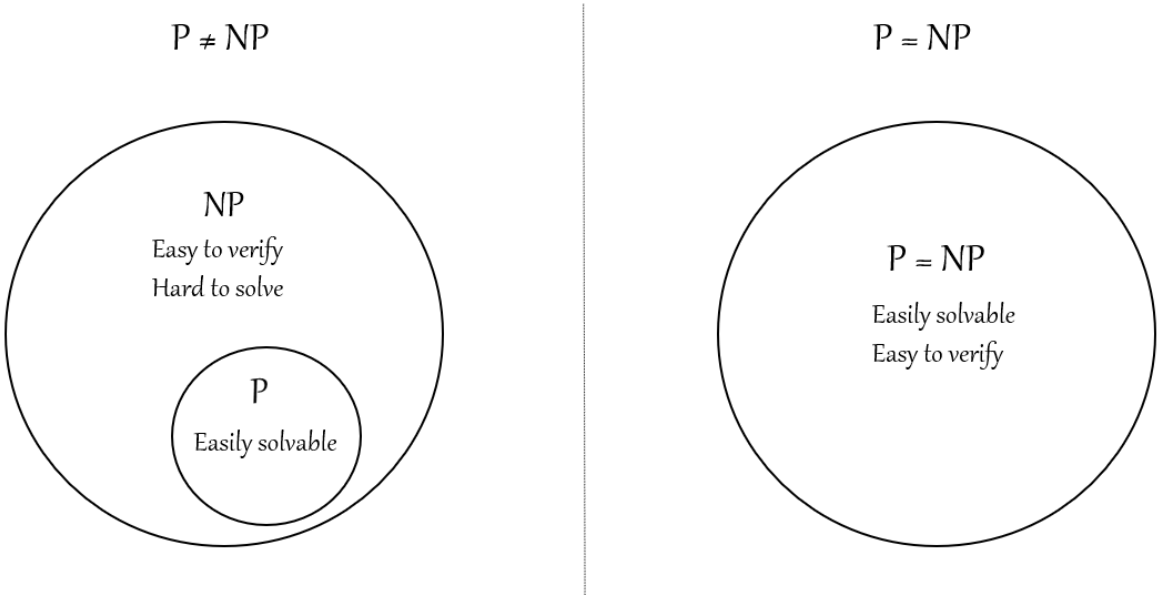


Figure 2: P vs NP problem. Simplified diagrams of the two possible scenarios, naming where P≠NP and where P=NP, are shown with the respective implications of each case.

1.2.2 DNA Computing and NP-complete problems

With existing evidence of the limitations of sequential computers, novel paradigms of computation have emerged. In 1994, Leonard Adleman [15] proved the possibility of computing with individual molecules [16]. Inspired by Richard Feynman's visionary description of a 'sub-microscopic' computer [17], Adleman solved the NP-complete Hamiltonian path problem [18] which asks whether, given a directed graph G with designated vertices v_0 up to v_N, there exists a "one-way" path of edges e_0, e_1,…, e_N such that every vertex is visited exactly once. To generate random paths, Adleman assigned a different 20 base long DNA oligonucleotide O_i to each vertex, i. For each edge $i \rightarrow j$ in the graph, an oligonucleotide $O_{i \rightarrow j}$ was created that was the 3' 10-mer of

O_i followed by the 5' 10-mer of O_j. Oligonucleotides were mixed in a test tube to generate all possible paths. Therefore, a series of polymerase chain reactions (PCR) were carried out to eliminate paths *not* beginning with v_0 and ending with v_N and *not* having exactly N vertices. A series of sequential affinity purification steps were performed to eliminate all paths having one or more edges appearing more than once; only if a path would remain after pruning, then there would exists a Hamiltonian path for the given graph G. Adleman exploited the organization and complexity of living molecules and the nature-refined mechanisms of data storage and processing in DNA. The first accomplishment of using single molecules to encode independent descriptions of a Turing machine and lab protocols and enzymes to induce successive sequences of modifications which the machine would 'execute', generated excitement. Nevertheless, the benefit of massively parallel computation involving 3×10^{13} copies of oligonucleotides for each edge was compensated by practical time-consuming processing and readout operations. The problem was solved in 7 days and involved a series of lab steps and considerable human intervention. According to Adleman [15], the quantity of DNA used during the in-tube ligation step should be sufficient to ensure that a molecule encoding a Hamiltonian path will be formed with high probability if such a path exists in the graph. This quantity would grow exponentially with the number of vertices in the graph and would soon become larger than the available DNA on earth (3×10^{13} copies of oligonucleotides for each edge correspond to around six hundred million molecules for a 7 edges graph already). In fact, using randomly generated paths to discover the existence of a unique solution inevitably generates combinations outside the domain of possible paths e.g. $v_1 \rightarrow v_1 \rightarrow v_1 \rightarrow v_1 \rightarrow v_1$, considerably increasing the required quantity of DNA molecules. In addition, problems concerning accuracy may occur. The synthesis of DNA strands is liable to errors, such as mismatching pairs, and is highly dependent on the accuracy of the enzymes involved. Given Adleman's small network involving less than 100 possibilities, errors were not a high concern. Nevertheless, a fully operational molecular computer performing thousands upon thousands of calculations would see the chance of errors increasing exponentially [19].

DNA has a unique ability to carry out multitasking and perform large number of operations simultaneously. Differently from von Neumann computers, DNA computers are stochastic machines with high-density storage capacity and parallel processing suitable to run exhaustive parallel searches through the solution space of combinatorial problems. Nevertheless, despite the progress achieved in DNA computing [20, 21], obstacles concerning expensive and time-

consuming processing and readout operations, versatility of DNA to solve a wide variety of computational problems, difficulty in scaling to large systems, management of errors and unwanted hybridization and the requirement of an exponentially large concentration of reactance with respect of the size of the problem [20], all prevent DNA-based solutions from outrunning silicon-based computers.

1.2.3 Quantum Computing and NP-complete Problems

As for classical computers, whether quantum computers are able to solve NP-complete problems in polynomial time is still an open question [14]. The closest attempt was in 1994 [22], when Peter Shor theoretically investigated the quantum computing model by solving the problems of finding the discrete logarithms and factorising integers [23, 24]. Generally, such problems are considered NP-intermediate on a classical computer, meaning they are in the complexity class NP but are neither in the class P nor NP-complete [25]. Nonetheless, the factoring problem has very important applications in cryptography, since a polynomial-time algorithm for factoring would crack the Rivest–Shamir–Adleman (RSA) public-key system [26]. By demonstrating a quantum algorithm taking polynomial-time on $\log(N)$ where N is the input size [27], Shor unveiled the potential of a quantum speed-up over classical algorithms. However, a practical implementation of the algorithm relies on a large and scalable number of quantum bits (qubits) and currently remains a challenges given the errors introduced by decoherence when a (relatively) large number of qubits and gates are used [28]. A recent study [28] proposed a first proof-of-principle demonstrating Shor's factoring algorithm on numbers N = 15, 21 and 35 using five, six, and seven superconducting qubits respectively. The least possible number of physical qubits were used and the circuits were designed to reduce the number of gates to the minimum. The experimental results showed to be in agreement with the theory for N = 15 and 21. However, the experiment succeeded for N = 35 only about 14% of the time, due to the cumulative errors coming from the number of two-qubit gates becoming too large [28]. The deleterious decoherence, which results from interaction between quantum computer and its environment introducing errors, is currently the major physical bound to scaling quantum systems [14]. At least theoretically, quantum algorithm demonstrated considerable speed up on solving a restricted class of problems of intermediate hardness [25], but they may not provide more than a $O(\sqrt{N})$, where N is the size of the input

domain, improvement when solving NP-complete problems, which is still an exponential bound [29].

1.2.4. Microfluidic Computing and NP-complete Problems

As an alternative to DNA and quantum computing, microfluidic-based computing has emerged providing the basis for new types of parallel computation [30]. With the increasing complexity of microfluidic systems, it became possible to perform complex processes - logic operations and fluid control – directly on the chip. This new hardware-embedded approach exploits the simultaneous searching through parallel branches of a microfluidic network lithographically defined on the chip, where the network encodes a mathematical problem of choice [13]. A recent contribution [30] proposed a 3D network to solve the NP-complete Maximal Clique problem [31] for a simple graph with six vertices using fluid flow containing fluorescent beads and a parallel optical readout of solutions. Informally, the clique problem asks, for a network *G* of *n* vertices and *p* edges, if there are subsets of nodes with *k* vertices within *G* that have the property of all their members being completely connected to one another ('cliques') [13]. The network is composed of three layers: each layer contains a reservoir representing one edge *p*, the wells for all of the subgraphs of *G* that contain the edge, and the channels that connect these wells to the reservoir. Quantification of the connectivity of each subgraph was accomplished by measuring the flow from reservoirs into wells which contained a uniform suspension of fluorescent beads [30]. The microfluidic system proposed is an analog computation device with two potential sources of errors: (i) the possible deviation of fluorescence intensities from the expected values due to bias splitting of fluorescent beads at each channel branching and (ii) the misalignment between layers that results in error in the integrated fluorescence intensities. Even if the aforementioned drawbacks limit this approach from scaling, the success achieved suggests broader application for microfluidics in computation [30].

1.3 The Challenge of Generating True Random Numbers

This section describes the nature of randomness, the challenge of generating truly random numbers and their importance in cryptographic applications. An outline of existing attempts to generate randomness using DNA and quantum technologies is provided.

1.3.1 Deterministic vs Non-deterministic Processes

A von-Neumann digital computer is constructed from a set of switches as logic gates (performing AND, OR, Exclusive OR and NEGATION operations) and flips-flops as memory devices [3]. Complexity arises from the interconnections of many of these components creating complex circuits. The computational process is strictly deterministic [32]. As the signal travels through the circuits, it resolves into a digital conversion at the input and output stages. The introduction of errors is generally limited and the addition of errors is neither admitted not amplified by the computation steps unless "bugs" are present in the program or hardware [3]. On one side, strict determinism provides the reliability associated with digital computers but on the other, it deliberately suppresses internal dynamics. In a digital computer, all components are designed to recognise either the '0' or '1' state, making the concurrent use of digital and analog processing impossible [32]. Contrary, the DNA, quantum and microfluidic based models of computation do not abide by this kind of determinism, but rather by a weaker form, which is neither fully deterministic nor completely random. To better understand this point, consider a general model of computation which is deterministic, precisely like the Turing machine model. Control laws are defined by a set of rules transforming (mapping via a mathematical operation) a set of input variables to output variables. In a deterministic model, given an input signal, a sharply defined single output value is returned. Consider the same input being evaluated multiple times and the resulted output values being mapped to a distribution where the frequency versus output relationship is a Gaussian curve. The distribution will be centred around the most occurred output value and in fact it may even resolve to a delta-function if the dispersion is zero i.e., the control law is strictly deterministic. Generally, a *non-zero* dispersion may be attributed to a) errors at the inputs and outputs, or b) control laws. In a deterministic system, the latter is not possible. One simplified example of deterministic control laws is the Newton's equation of motion [33], according to which, given knowledge of the position and momentum of a particle at a given time,

the position and momentum of that same particle at a subsequent time step can be strictly determined. Repeated experimental measurements of a particle position yield possible variations in the output depending on the chosen boundary conditions or on the level of precision of the measurement itself (uncertainty). On the other hand, in non-deterministic models of computation, the spread of measured values from the mean of the output distribution is attributed to control laws themselves (mathematical operators mapping inputs to outputs) [32]. Examples of non-deterministic control laws can be observed within the rich repertoire of biochemical reactions where inherently random processes are, at some levels, restrained by the free energy minimization scheme. It is when the deterministic set of rules dies out that true randomness emerges. To visualize it, one can imagine the opposite representation of the aforementioned delta-function strictly defined over a single value and having infinite height and infinitesimal width. A truly random distribution has in fact equal frequency of occurrences at any given point (it can be approximated to a horizontal straight line), since, given a single input value and repeated measurements, the resulted output values are completely unpredictable and can take any of the possible values with equal probability.

1.3.2 Randomness in Cryptography

The strict determinism inherent in modern electronic computers is a fundamental limit for the purpose of generating random numbers. As discussed, true randomness is not computable by a deterministic algorithm and can only emerge out of stochastic processes. Random values have wide range of applications in network security, digital communication, computer simulations, statistical sampling [34] and, in particular, they are a key requirement for building secure cryptosystems. Cryptographic protocols rely on the generation and use of secret encryption and decryption keys that must be unpredictable to potential attackers and, for this purpose, true randomness is a unique and essential feature [35]. To be considered truly random, a secret key must have the following properties: a) each number must be statistically independent from the previous and, given each number in the sequence, a particular value is not more likely to follow than all other possible ones, b) numbers in the sequence must follow a uniform distribution and must have the same frequency of occurrence and finally c) numbers in the sequence must be unpredictable, both in terms of backtracking and predicting future values [36].

1.3.3 DNA-based True Random Number Generators

Given their digital nature, electronic computers are hardly compatible with external, entropy-providing physical processes. Such incompatibility is making random numbers expensive and less available. True random number generators (TRNG) are inevitably non-deterministic systems requiring a hardware component to sense entropy in the environment and turn it into non-deterministic numbers [36]. The DNA computation model is endowed with such property of non-determinism and its control laws generate stochastic biological processes and dynamics. The random construction of DNA oligonucleotides sequences [37] and the DNA molecular motion behaviour [34] have been proposed as sources of noise, or entropy. At present, DNA-based RNG approaches were able to successfully generate random sequences, but on the down side, current limitations of DNA technologies including the considerable human intervention, the expensive production of large amounts of DNA and the time constraints required to process and read oligonucleotide sequences [37], prevent such methodology from scaling and providing large amount of data in a fast and automated fashion.

1.3.4 Noise-based True Random Number Generators

A stochastic source worth considering is noise, obtained out of physical systems. Noise-based TRNGs are built to periodically sample analogue voltage and compare it to a certain pre-defined threshold: if higher, then “1” is generated, otherwise “0” [38]. Noise-based TRNGs face a variety of problems that make them difficult to prove effective [39]. They often produce very small voltage that requires the use of strong amplifiers which in turn introduce further deviations and non-linearity and can be interfered for cryptographic attacks. The sampling and digitizing procedure are also disturbed by the time-consuming process of fine tuning the threshold, which is one of the major challenges to ultimately maintain probabilities of 0s and 1s to be roughly the same for maximal entropy [39]. In addition, most types of noise sources are exposed to some long range, motion-induced correlations of quantized electric charge in conductors [40].

1.3.5 Quantum Random Number Generators

Control laws orchestrating many physical processes of quantized matter are probabilistic from a classical perspective. Random numbers can be generated through the fact that *observing* the state of a particle (measuring which leads to a state collapse of the particle wave function) while in superposition gives a true 50/50 outcome (qubit value 0, 1) [41]. This concretely random output for the value of the single bit can be used to build integers comprised of larger numbers of bits. This process is known as a Quantum Random Number Generator (QRNG) [42]. QRNGs use photon counting [43], vacuum fluctuation [44], phase noise of laser [45, 46], Raman scattering [47], particle arrival times [48, 49], branching path superposition [50] and generally exploit optical and non-optical quantum phenomena as sources of randomness [51]. Among the cited approaches, common challenges include unstable bias being sensitive to temperature variations and only mechanically adjustable [39], correlations in the final sampling period as discussed with noise-based TRNGs, unbalanced detectors and limited single-photon resolving capability [51]. In addition, the fundamental problem of measuring time intervals, e.g., photon arrival time, incurs in the risk of reducing quantum information into a clocked, correlated classical form [39].

1.4 Introduction to Network Computing

Combinatorial computational problems i.e., NP-complete problems, which cannot be solved efficiently by sequential electronic computers, ask for the development of alternative, massively parallel computing technologies. While a number of such technologies have been proposed, including quantum and DNA computation, none have scaled so far, due to noise, errors and various technological challenges. In a recently introduced approach [52], the autonomous motions of molecular-motor driven cytoskeletal filaments inside microfluidic networks designed to encode a small instance of an NP-complete problem, were harnessed to search a large candidate solution space in parallel. The proposed proof-of-concept was demonstrated solving an instance of the subset sum problem (SSP) which asks whether, given a set $S = \{s_1, s_2, .., s_N\}$ of N integers, there exists a subset of S whose elements sum to a target integer, T [13]. Exploring all possible subsets to find all subsets sums requires testing 2^N different combinations. Figure 3 demonstrates the operational functionalities of the device and the network geometry encoding the SSP. Biological agents enter the network from the top and swim downwards towards the bottom exits. The network

is comprised of 'split junctions', designed to let agents take any of the two possible directions with 50/50 chance and 'pass junctions' designed for agents to continue on their straight path. The size of the SSP network is determined by (i) its unit cell size, (ii) the number of elements in the set *S* and (iii) the compactness of the series, i.e., the relative distance between the numbers in the set [13], which here was chosen to be the prime numbers series. Each agent represents an independent unit of computation having its internal state updated when moving between junctions; turning *left* at a junction corresponds to adding a 1 to its internal state, while turning *right* corresponds to adding a 0. Each exit is numbered with the corresponding number of diagonal steps required to reach it. Each network path maps to one of the 2^N possible solutions. By not visiting 'incorrect' exits (Figure 3 - magenta), agents automatically exclude erroneous paths, i.e., incorrect solutions of the problem. The final result is extracted based on the visited exits.

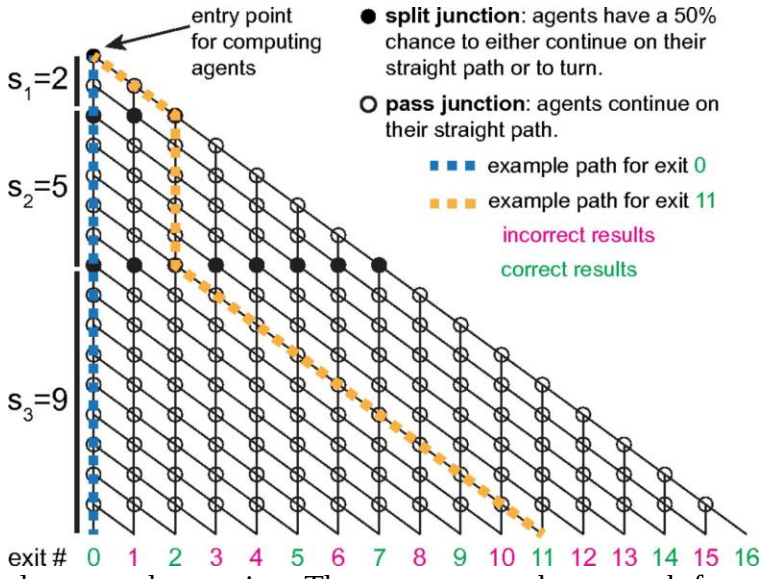


Figure 3: The chip layout and operation. The agents enter the network from the top-left corner. Filled circles represent split junctions where it is equally probable that agents continue straight ahead or turn. Empty circles represent pass junctions where agents continue straight ahead. Moving diagonally down a number of split junctions corresponds to adding that integer (numbers 2 and 9 in the yellow path example). The exit numbers correspond to the target sums *T* (potential solutions) represented by each exit; correct results for this particular set {2, 5, 9} are labelled in green, and incorrect results (where no agents will arrive) are labelled in magenta.

The development of the stochastic computational device just described laid down the fundamental research case addressed in this work. This research aims to assess whether such

device will be able to scale, solve other problems and potentially compete with electronic computers. Compared to other approaches [30], the device benefits of the self-propelled motion of myosin II and kinesin-1 molecular motors which eliminates the need of external forces to drive computation. This need inherently prevents other microfluidic approaches from scaling up, because the pressures needed to pump fluid through the network become prohibitively large for large N [52]. Molecular motors operate in a highly energy-efficient manner. As a result, the approach demonstrated consumes orders of magnitude less energy per operation compared with both electronic and microfluidic computer. Finally, the networks are planar, which solves potential engineering challenges associated with building large-scale 3D microfluidics devices [30, 52], and comprise of standardized modules, which makes them fully scalable. Among the challenges encountered, the single-entry point of the network represented a bottleneck resulting in channel clogging and high booting time - cumulative time that all agents take to enter the network given their velocity and the channel dimension [52].

In the next sections, the principle, design, implementation, operation, scaling, challenges and future maturation of the aforementioned device are analysed from an engineering perspective. The impact that self-replication of agents in the network would have on the booting and total computing time of the device, the scaling capability of the device and the possibility of errors arising from agents taking prohibited paths at pass junctions (which played a key role in the recognition of correct and incorrect results [52]) are projected and discussed. Moreover, this contribution brings the concept of a network-based computing device a step forward by proposing an alternative network suitable to generate random numbers. Here the device functions as an entropy-providing component of a TRNG by exploiting the stochastic motion of *E. coli* bacteria inside the network.

The next chapters are organized as follows: Chapter 2 discusses scaling-related engineering challenges in terms of fabrication, readout reliability and energy efficiency of the device. The proposed guidelines aim to define challenges, but also best practices and methodologies, associated with the success of the network computing field. Chapter 3 puts in perspective the computing time performance of the device with those of classical and quantum computers solving the same instances of an NP-complete problem. Chapter 4 treats the device from the application perspective of generating cryptography-secure random numbers. A full model producing random

binary sequences with high throughput and reliability is demonstrated. It follows Chapter 5 with a general discussion and finally, Chapter 6 with a conclusion.

Chapter 2

Engineering Challenges and Future Road-map of Network Computing

Like other alternatives to sequential computing, network-based computing faces scalability issues. It is in fact crucial to identify both engineering bottlenecks blocking its progress and possible technological advancements for its success in real-world applications. Consequently, the following contribution, published in Interface Focus – The Royal Society Publishing, discusses such fundamental issues with respect of the present state-of-the-art in the field.

Something has to give: Scaling combinatorial computing by biological agents exploring physical networks encoding NP-complete problems

Falco C.M.J.M. van Delft[1a+], Giulia Ippoliti[2+], Dan V. Nicolau Jr.[1,3], Ayyappasamy Sudalaiyadum Perumal[2], Ondřej Kašpar[2,4], Sara Kheireddine[2], Sebastian Wachsmann-Hogiu[2], Dan V. Nicolau[2]*

[1] *Molecular Sense Ltd., Liverpool, L36 8HT, United Kingdom*

[2] *Department of Bioengineering, McGill University, Montreal, Quebec, H3A 0E9, Canada*

[3] *School of Mathematical Sciences, Queensland University of Technology, Brisbane, QLD 4000, Australia*

[4] *Department of Chemical Engineering, University of Chemistry and Technology, Prague, Technická 5, 166 28 Prague 6, Czech Republic*

[+] *these authors contributed equally*

[*] *to whom all correspondence should be addressed*: dan.nicolau@mcgill.ca

[a] *present address: Nanovalk, Gertrudisdal 15, 5551BD Valkenswaard, the Netherlands*

Abstract

On-chip network-based computation, using biological agents, is a new hardware-embedded approach which attempts to find solutions to combinatorial problems, in principle, in a shorter time than the fast, but sequential electronic computers. This analytical review starts by describing the underlying mathematical principles, presents several types of combinatorial (including NP-complete) problems, and shows current implementations of proof of principle developments. Taking the Subset Sum Problem (SSP) as example for in-depth analysis, the review presents various options of computing agents, and compares several possible operation 'run modes' of network-based computer systems. Given the brute force approach of network-based systems for solving a problem of input size C, 2^C solutions must be visited. As this exponentially increasing workload needs to be distributed in space, time, and per computing agent, this review identifies the scaling-related key technological challenges in terms of chip fabrication, readout reliability and energy efficiency. The estimated computing time of massively parallel or combinatorially operating biological agents is then compared to that of electronic computers. Among future developments which could considerably improve network-based computing, labelling agents ‘on the fly’ and the readout of their travel history at network exits, could offer promising avenues for finding hardware-embedded solutions to combinatorial problems.

Keywords: Network-based computation, Bio-computation, Combinatorial problems, NP-complete problems, Hardware-embedded solutions, Subset Sum Problem

2.1. Introduction

Many combinatorial problems of practical importance, including NP-complete problems, appear to require that an extremely large number of possible candidate solutions is explored in a brute-force manner in order to discover the actual solutions. Examples of such problems are the design and verification of circuits [1], the folding [2] and design [3] of proteins, optimal network routing [4], formal reasoning [5] and data clustering in complex networks [6]. When the size of these problems grows, the time required to find solutions on sequential computers grows exponentially. Consequently, solving these problems by any computer that performs computations sequentially, including electronic computers, requires unreasonable computing times, even for medium-sized problems, as implied by the NP-Hardness Assumption [7]. Therefore, to solve these problems in practice will require efficient parallel computation approaches [8], but those presently proposed raise various critical technical difficulties. For instance, DNA computing generates mathematical solutions by recombining DNA strands [9, 10], or DNA-static [11] or -dynamic [12] nanostructures, but this approach requires impractically large amounts of DNA [13-16]. Quantum computing appears to be limited in scale by decoherence and by the small number of qubits that can be integrated [17]. Finally, microfluidics-based parallel computation [18] is difficult to scale-up with the size of the problem due to the rapidly diverging physical size and complexity of the devices.

A recently proposed alternative, network-based computation [19], may be capable of overcoming some of these scalability problems. A network-based computing device comprises a network which is a physical embodiment of a graph representing an instance of a mathematical problem. The network-based computation consists in the directed movement of motile physical objects – computation agents, through the entries, conduits, nodes, and exits, of the computer network. The history of the positions of the exploring agents through the encoded network, if decoded, represents the solution to the mathematical problem. Consequently, DNA computing does not represent a subset of network-based computation, although it does attempt to solve an NP complete problem which does have a classical graph-based representation. The core concept of network-based computing is to map the set of all possible solutions of mathematical problems into physical structured pathways to encode the ‘content’ of the problems, and then to find the

solution(s) by exploring these pathways using a large population of autonomous, self-propelled ‘agents’, such as molecular motors-driven cytoskeletal filaments [20], or microorganisms [19]. This approach was used to demonstrate the principle of solving in a combinatorial manner a small instance of an NP-complete problem, the Subset Sum Problem (SSP) [20]. The estimated energetic efficiency in this computation approach also was orders of magnitude higher than that of electronic computers, suggesting that such a technology might circumvent the heat dissipation which is one of the limiting factors in developing ever-larger classical supercomputers [21].

Like other alternatives to sequential computing, network-based computation, in particular using biological agents, faces scalability issues of its own. Consequently, it is imperative to identify the engineering bottlenecks blocking the progress and explore possible avenues for alternative solutions. This methodological approach is expected to lead to the aggregation of a “road map”, similar to the one formally developed by the community of the semiconductor industry. To this end, this contribution maps the current state-of-the-art in network-based computation, with an emphasis on the use of biological agents, starting with the mathematical principles, comparing various types of computing agents, technological challenges related to fabrication and readout, and opportunities regarding energy efficiency. Drawing this all together, we attempt to identify the advancements in several ‘service technologies’ that are likely to be necessary for network computing with biological agents to become useful for real-world applications.

2.2. Network-based computing with agents

2.2.1. Network-based computing: concepts and tentative implementations

Supposing that specific NP-complete problems can be formulated as *graphs* [19], it is possible to translate these into *designs of physical networks*, i.e., graphs with dimensions, e.g., distances between vertices, widths of the lines connecting these vertices, etc. These designs then can be the basis of the fabrication of physical networks, such as microfluidic structures comprising channels, nodes, entries and exits. These devices are essentially *computer networks* that encode the NP-complete problem of interest, which then "waits" to be solved through the stochastic exploration, in parallel, by a large number of independent agents which act as '*processors*' (pseudo-central processing units (CPUs)), each searching independently for a solution, through the process of moving from one junction to another 'downstream' from the entry towards (one of) the exit(s). Essentially, the physical network is not the computer per se, but it is the physical input to calculations. Therefore, network-based computing combines the 'hardware' design of networks, encoding mathematical problems of interest (see electronic supplementary material, SI-1, for the mathematical formulation), with the 'software' or information-processing capacity of a population of agents freely and stochastically exploring this network in a combinatorial fashion.

This staged process, i.e., graph encoding a mathematical problem → design of a physical network → fabrication of a microfluidic device → massively parallel exploration of the network by a large number of agents, has been recently proposed [20] as a proof of principle for solving the SSP. However, this strategy is amenable to other graph-based formulations of NP-complete problems.

Various implementations of network computing schemes encoding NP-complete problems, using various computational agents, have been attempted. Importantly, all reported implementations used solely the combinatorial run mode (see section 2.2.2), i.e., a large number of agents exploring simultaneously a physical network encoding the mathematical problem. Some of these problems, and their implementation in proofs of principle devices, are reviewed (and presented in Figure 4).

2.2.1.1. Subset Sum Problem

The *subset sum problem*, a benchmark combinatorial problem [22], asks whether, given a set $S = \{s_1, s_2, ..., s_n\}$ of n integers, there exists a subset of S whose elements sum to a target integer, T (Figure 4*a*(i)). SSP has applications in various fields, especially when optimising resource usage under constraints, and the "hardness" of the problem is harnessed in certain cryptographic systems to generate encoded messages [23], due to its simple construction and resistance to quantum attacks [24]. Also, SSP, or its variant, the knapsack problem [25], finds applications in resource allocation for specialised producers, in efficient throughput and congestion allocations despite selfish users' behaviour, in the allocation of bandwidth in communication networks based on user requests, and in auctions. A recent review [26] provides an insightful discussion on existing and possible applications.

A methodology to solve the SSP that uses biological agents has been proposed [19] and recently demonstrated [20], using cytoskeletal filaments, i.e., actin filaments, or microtubules, propelled by protein molecular motors, i.e., myosin, or kinesin, respectively (Figure 4*a*(ii, iii)). Interestingly, there have also been proposals for solving SSP by optical computing [27, 28], but it was found that the energy required for large problems is prohibitive.

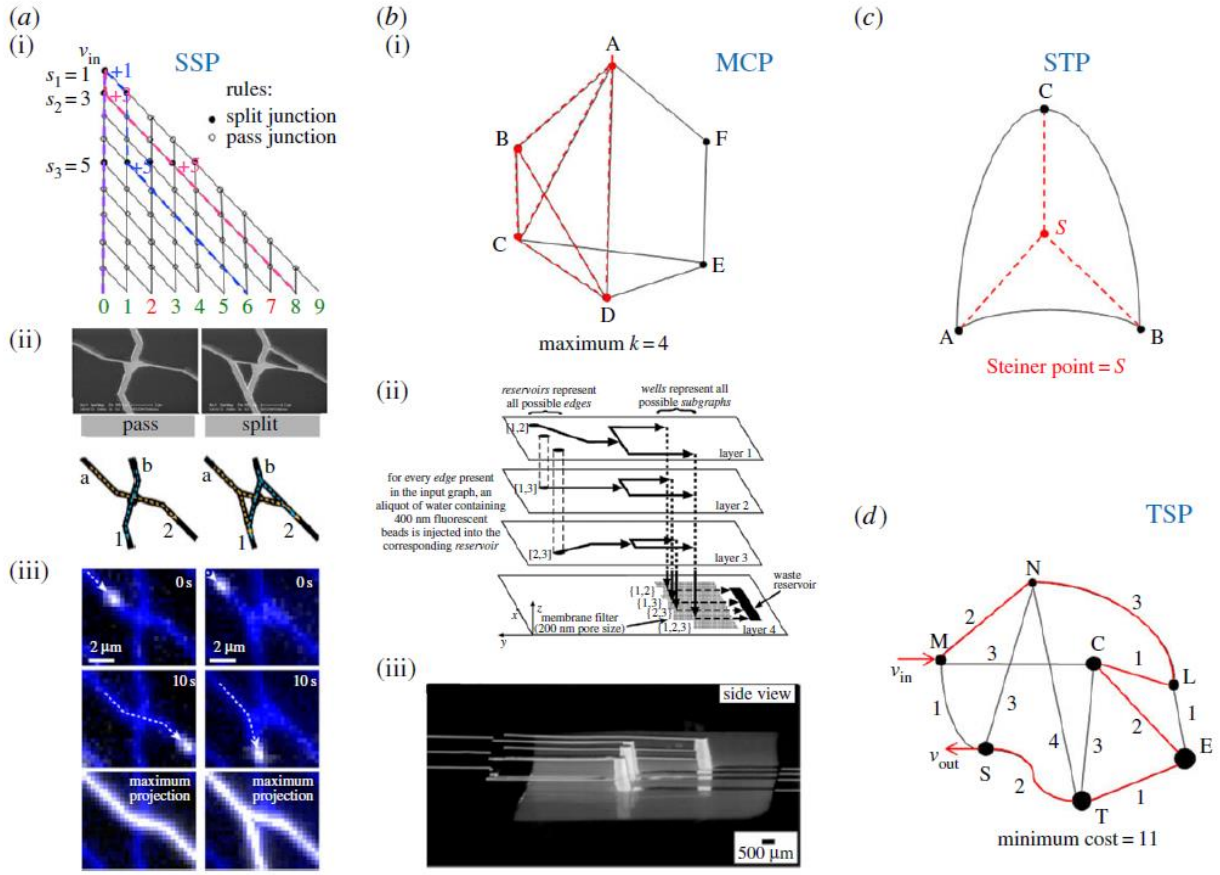


Figure 4: (*a*) Subset sum problem (SSP). (*a*(i)) Representation of a computation network for the subset sum of {1,3,5}. The agents start in the top left-hand corner. The junctions of the paths are: *filled circles*: SPLIT junctions where the agents have a 50% probability of continuing their straight path or to turn, or *empty circles*: PASS junctions where the agents always continue their straight path. Moving straight down at a split junction corresponds to not adding an integer to a running sum (purple example path). Moving diagonally down at a split junction corresponds to adding that integer (numbers 1 and 5 for the blue example path). The actual value of the integer potentially added at a SPLIT junction is determined by the number of rows of PASS junctions following that particular SPLIT junction (numbers indicated on the left of the paths). Green exit numbers represent sums for which a matching subset exists, and red numbers represent sums for which no matching subsets exist. (*a*(ii)) SEM graphs and schematic of pass (left) and split (right) junctions [20], where entrance and exit channels are labelled a and 2 respectively for agents travelling on diagonal paths, while entrance and exit channels for agents moving in a vertical path are labelled b and 1. The yellow dotted lines indicate diagonal paths and blue dotted lines indicate straight paths. (*a*(iii)) Fluorescence micrographs highlighting paths of moving microtubules across a pass

(left) and a split (right) junction. Images in the third row show the maximum projection of agents in motion. Figures (*a*(ii)) and (*a*(iii)) are adapted from Nicolau et al. [20] (Copyright 2016) which also presents an animation detailing the function of the SSP computing principle. (*b*) Clique problem (CP). (*b*(i)) Maximum clique problem (MCP) computed on a given undirected graph G comprising nodes n = {A, B, C, D, E, F}. The maximum clique (highlighted in red) results to be of size k = 4 with vertices subset {A, B, C, D}. (*b*(ii)) Schematic of a four-layer microfluidic device used in solving an MCP for a graph having three vertices [18]. This three-dimensional microfluidic system has reservoirs—where a plug of fluorescent beads is injected—to represent all of the possible edges of a graph with three vertices, and wells—where the fluorescent beads are collected by a size filter sandwiched between the bottom and the top three layers—to represent all possible subgraphs of a three-vertex graph. The arrows in the schematic indicate directions of fluid flow; suction (house vacuum) is applied at the waste reservoir to drive fluid flow from the reservoirs representing edges to the waste reservoir. (*b*(iii)) Fluorescence photograph of the actual device for solving a three-vertex graph, viewed from the side. Figures (*b*(ii)) and (*b*(iii)) are adapted from Chiu et al. [18] (Copyright 2001 National Academy of Sciences, USA). (*c*) Steiner tree problem (STP). Out of the possible connection paths between three nodes on an undirected graph, n = {A, B, C} a single Steiner point, S joins the vertices with minimum distance. (*d*) Travelling salesman problem (TSP). A representation of the TSP by an undirected graph with designated vertices v_{in} = M and v_{out} = S, for which the minimum cost Hamiltonian path is M → N, N → L, L→ C, C → E, E → T, T → S with as total cost, C = 11.

2.2.1.2. Clique Problem

The *clique problem* (CP) asks, for a network *G* of *n* vertices and *p* edges, if there are subsets of nodes with *k* vertices within *G* that have the property of all their members being completely connected to one another ('cliques'). Several formulations of the problem exist, of which the maximum clique problem (MCP) is the best-known. The MCP consists in listing all maximal cliques that cannot be enlarged by solving the decision problem on whether *G* contains a larger clique than the current size *k*. MCP asks to determine a complete subgraph of maximum cardinality, or maximum vertices [29]. Figure 4*b*(i) represents the MCP problem for a given graph G = (*n*, *p*) with *n* = {A, B, C, D, E, F}. A brute force algorithm exploring all possible solutions,

finds out that the set of vertices {A, B, C, D} is a maximum clique of G, and therefore, that the maximum $k = 4$. MCP is notable for its relevance to a large number of applications, e.g., bio- and chemo-informatics, coding theory, economics, examination planning, financial networks, scheduling, signal transmission analysis, social network analysis, and wireless networks and telecommunications. A recent review [29] provides a comprehensive bibliography.

CP has been solved [18] by means of network computing using a multi-layered, 3D microfluidics structure (Figure 4*b*(ii),(iii)), which encodes the MCP for a simple graph with six vertices, explored by beads carried by fluid flow. While the calculation and the readout are done in parallel, the computing process is biased, as the beads will follow the lowest pressure lines in the flow, rather than independently explore the solution space. Also, the power required for pumping the fluid in microfluidic channels grows exponentially with the size of the problem, resulting in an unreasonable pressure build up [20]. The MCP has also been solved by DNA computing [15].

2.2.1.3. Steiner Tree Problem

The *Steiner tree problem* (STP), and one of its special cases, the Minimal Steiner Tree Problem (MSTP), asks, given a network *G* comprising *n* nodes and *p* edges, and a special subset of those edges (usually called terminals), for a tree that contains all these terminals (but which may include additional vertices) [30]. As with most NP-complete problems, there are a number of variants, but all ask, ultimately, for an optimal interconnect for a given set of objects, *subject* to a predefined objective function. Figure 4*c* represents the STP problem computed on an undirected graph G = (*n*, *p*) with *n* = {A, B, C}. An algorithm searching for the minimum cost connected set that joins all the given nodes, eventually decides that the solution lies at a single Steiner points, S. STP is relevant to many applications, e.g., VLSI physical design, FPGA routing placement, telecommunication network design, keyword-based selection of relational databases, data-centric routing in wireless sensor networks, multicast packing, network topology control, and access strategies design for ISP networks. A recent report [31] provides a detailed bibliography regarding these applications.

STP, in particular the MSTP, has been solved [31, 32] using a slime mould-, i.e., *Physarum polycephalum*-inspired algorithm. It should be noted, however, that solving optimisation problems, such as the STP and the traveling salesman problem (see below) using slime moulds does not perfectly fit the definition of network computing with agents, because the edges of the graph are not pre-determined, and because the slime mould represents a collection of agents, i.e., tubular elements, which do not operate independently.

2.2.1.4. Travelling Salesman Problem

The *traveling salesman problem* (TSP) is one of the best known NP-complete problems. Given a graph such that cities are vertices and the distances between them correspond to the graph's weighted edges, the problem asks for the shortest route (or another performance criterion, e.g. lowest travelling cost) between 'cities' under the condition that each valid route visits each 'city' only once. In other words, the problem asks to find the Hamiltonian cycle, being the path that visits every node once, at minimum cost. Figure 4*d* shows an undirected graph G = (*n*, *p*) with *n* nodes being the set of cities and *p* edges being the possible paths. For each new node visited, the total cost, C is incremented by the weight, equivalent to the distance travelled to visit the new node. The algorithm computes by brute force all possible tours under the initial conditions of start, v_{in} = M and finish point, v_{out} = S. Thus, the Hamiltonian cycle of minimum cost is M → N, N → L, L → C, C → E, E → T, T → S with C = 11. A generalisation of the TSP, very relevant for network computing with agents, is the multiple TSP [33], which consists of determining a set of routes for *m* salesmen who all start from and turn back to a home 'city' (depot). Aside of the obvious relevance to traffic and scheduling, TSP is being used in applications as diverse as drilling of printed circuit boards, overhauling gas turbine engines, X-ray crystallography, computer wiring and order-picking in warehouses. A recent review [34] provides a comprehensive compendium of TSP applications.

Despite being the first NP-complete problem to be solved by brute force non-electronic computers, i.e., by DNA computing [9], and despite the Hamiltonian graph being, arguably, conceptually the closest to a physical network in its native problem form, TSP has not been solved by *physical* network computing using multiple agents, although a multicellular organism, i.e.,

Physarum polycephalum, has been used [35] to generate an approximate solution of TSP. This under-representation of solving TSP by network computing using multiple agents is even more surprising as an elaborate mathematical framework exists for the exploration of TSP networks using ant colony algorithms [36]. Instead, TSP appeared as an operational problem in running digital microfluidics, which needed to be solved by efficient algorithms [37]. The TSP has also been solved by optical networks [38], where the agents are essentially photons.

2.2.1.5. Maze solving

Maze-solving, asks, given a maze (a grid of *n x n* regularly arranged nodes in which only connections between adjacent nodes are permitted), for a path from an entrance point to an exit point. Although the classical version is computationally tractable, i.e. requires polynomial time on a sequential computer, some versions of maze-solving, such as the simultaneous maze-solving problem [39] are NP-complete. Importantly, a great deal of experimental work has been done using many different kinds of agents to solve mazes [40-44].

Maze-solving has been classically used to experimentally assess the optimality of behavioural response, or intelligence of many organisms including ants, bees, mice, rats, octopi, and humans[45], and more recently by fungi [8, 46, 47], bacteria [48], *Caenorhabditis elegans* [44] and by an amoeboid [41], as well as artificial intelligence-enabled robots. Despite this very large body of experimental methodology, and very diverse use of biological agents, and despite the demonstration of the efficiency of the space searching algorithms developed by microorganisms, e.g. fungi [49], the exploration of mazes by multiple agents has not been used as a means to solve any NP-complete problem, e.g., the simultaneous maze-solving problem.

2.2.1.6. Satisfiability problem

The satisfiability problem (SAT) asks, given an input Boolean formula built from variables and constraints using the NOT, AND, and OR operations, if TRUE or FALSE can consistently replace the elements of the input formula in such a way that the overall formula evaluates to TRUE. Satisfiability is an important NP-complete problem, with extremely varied applications, from

ordinary ones, e.g., schedule events depending on the availability of actors and venues, and seating assignment consistent with various imposed rules, to critical decisions, e.g., design and verification of digital circuits, planning in artificial intelligence with practical use in space exploration and industrial microprocessor verification [50]. Despite this importance, and despite often using graphs to articulate relevant algorithms, SAT was not yet translated in a design of a physical network amenable to the exploration by biological agents.

Sum-up. There is a rather larger body of experimental work attempting to implement various solving approaches of NP-complete problems using the framework of network computing, and the majority of these efforts use a large variety of biological agents. Despite this interest, there are NP-complete problems intrinsically encoded as a network, e.g., the TSP, which have not been solved by multiple biological agents, whereas others, such as the SAT, e.g., 3-SAT, are waiting to be theoretically encoded in designs of networks amenable to the exploration by biological agents.

2.2.2. Agent run modes for network computing

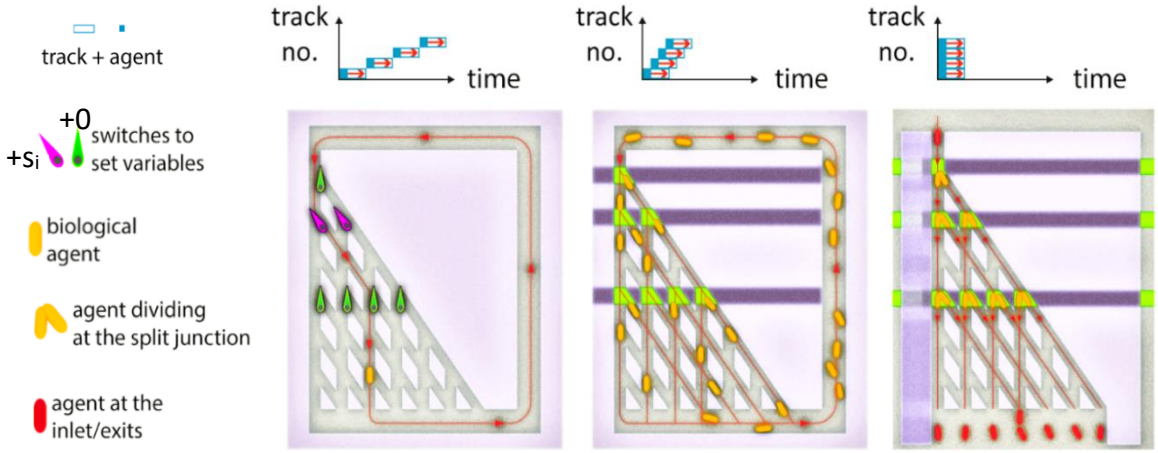
The exploration of networks encoding combinatorial problems, such as NP-complete problems, by motile agents, can be conducted in various run modes. Figure 5 schematically presents these operational modes, taking the SSP-encoded network [19, 20] as a benchmarking example. The SSP network has a triangular structure with a single starting point (top left corner in the panels in Figure 5). The network features split junctions (where traffic can change direction) and pass junctions (where traffic crosses without interaction). The exits at the bottom, representing the solutions, are connected by a feedback line to the starting point (if agents are to be recycled). An agent can, therefore, be considered to be a 'moving processor' (a pseudo-CPU).

The sequential run mode. The exploration of the network by only one agent at a time is equivalent to a purely sequential processing, even if that individual agent is recycled at the end of the computation. The green and purple flippers in Figure 5 represent logical switches at the split junctions, which are systematically set before every exploration run by the computing agent, in

order to explore the complete parameter space. This run mode is operationally equivalent to the computing process in a typical single-core electronic computer system.

The combinatorial run mode. This run mode, demonstrated recently using cytoskeletal filaments [20], is essentially a concatenation of two serial processes, i.e., the feeding of the network by agents waiting in a queue, which is equivalent to the booting of the computer, and the actual computation. If the agents are fed to the computer at a frequency higher than that equivalent to the full exploration of the networks by an individual agent, as in the purely sequential run mode, the computation progresses in a 'super massively parallel' manner, or more appropriately, in a combinatorial manner. Indeed, parallel computation processes, including "massively parallel" ones, involve the processing of information by a *constant*, even if large, number of processors during the calculation, whereas for the run mode described here, the number of 'processors' (initially) increases as the computation progresses. Furthermore, the larger the network and the feeding frequency, the larger the number of agents (as before, the agents can also be recycled).

As an agent in the queue, before entering the network, does not need to wait for the previous one to exit the network (but just to leave the entering point), the computational network accumulates agents exploring the network in parallel. As a consequence, however, in this run mode, no switches can be operated at the split junctions, and the right- or left-direction of an agent in the split junctions is a purely stochastic process, preferably with a 50%-50% distribution, induced by a local mirror-symmetric design of the split junction. Consequently, some combinations of SSP parameters, i.e., a specific subset sum, may appear multiple times before all various combinations have been visited. In order to have a very high probability that all combinations are being explored, the number of agent runs has to be enlarged by a factor that can be estimated using the formalism of the 'coupon collector's problem' [51]. A major drawback of the purely combinatorial mode is that the inefficient, serial 'upstream' booting process will limit the overall computing time.



	Sequential: one-by-one similar to 1 CPU system	**Combinatorial** agents simultaneously	**Multiplication** dividing bacteria model
Number of agents in the network	1 (constant) recycling	ramping up until agents' total exit rate = entry rate	increasing exponentially with time
Booting frequency	1/(run+recycle-time)	agent speed/agent length	-
Total booting time	2^C*(run+recycle-time)	2^C / booting frequency	0
One agent run time	longestpath/agentspeed	longest path/agent speed	longest path/agent speed
Total run time	total booting time + one agent run time	total booting time + one agent run time	one agent run time
Possible agents	electrons / currents	AM, MK, *E. coli, Euglena*	*E. coli ,V. natriegens*
Critical factors	- Sequentiality - Switches	- Booting (network entry) - Clogging at entry point	- Doubling rate vs. space - Clogging downstream

Figure 5: Possible operation modes in network computing. The top row shows the agents (blue squares) running in white rectangles (network tracks). The top three plots in the run mode panels show the start time delays for the various run modes. In run mode 1, the second calculation can only start after the agent finished its exploration (and, possibly, it has been recycled to the starting point). In run mode 2, the second calculation can start as soon as the first agent has left the starting point, i.e. there is physical space available for the next agent. In run mode 3, only one agent starts, and its 'off-spring' agents reach all the endpoints simultaneously, i.e., if all tracks would have had the same length. As the SSP network is an asymmetric triangle, the track to exit 0 is the shortest, and the track to the full sum exit is the longest. For calculating the expected computation time (see further), the latter track length (the longest arch) has been used for run time estimations.

The total booting time can be calculated as 2^C divided by the booting frequency, where the power C stands for the cardinality of the set, i.e. how many members the set has (as shown in Figure 5). The booting frequency is derived from the agent speed divided by the (average) distance between two agents (effective body length). This distance can be chosen as the agent body length (assuming head to tail queueing), or, e.g. twice the body length of the agent (assuming a 50% duty cycle at the entering point of the network).

The multiplication run mode. Given the inefficient, seriality-based booting of the network computing running in combinatorial mode, a fundamentally more efficient strategy would be based on multiplication of the agents *inside* the computing network, i.e., downstream from the feeding point. Intuitively, the maximum benefit in computing time will occur if the agents multiply at every split junction. For example, bacteria could undergo cell division while exploring the network. If this would be possible, at the starting point only one computing agent would be needed, multiplying 'on the fly', and all the routes and exits are visited by the off-spring of the original, 'mother' agent. However, while the multiplication after each split junction is an ideal option, multiplication itself, at a reasonable frequency, would counter the exponential increase in the number of possible solutions vs. problem size with the exponential increase in the number of computing agents. The consequences on the traffic density in the network depend on the compactness of the specific series encoded in the SSP, as will be discussed in section 2.4.1.

2.3. Biological agents

In order to efficiently explore the networks encoding combinatorial mathematical problems, in particular NP-complete ones, the computing agents must possess several performance parameters: (i) they need to be available in large numbers, to be able to explore the whole 'solution space', which for problems challenging sequential computers could run in the range of millions to billions; (ii) they need to have similar dimensions, to allow standardised designs of the networks, e.g., channel widths; (iii) these dimensions are preferably small, in the nanometre or micrometre range, to allow a high density of calculations per unit area; (iv) importantly, the agents must be autonomously motile, i.e., each agent needs to possess its own propulsion, with higher speeds translating into shorter computing times; (v) the agents must not interact with each other, to enable an independent search of the 'solution space'; (vi) while small, the agents must be independently distinguishable by a readout system; moreover, preferably the agents should be independently identifiable, i.e., each having its own 'ID'; and (vii) they need to exhibit additional physical properties as required by the respective implementation of the computing networks, e.g., non-adherent to the walls of the microfluidic channels and non-clogging.

The computing agents asked to explore mathematically encoded networks could have an *abiotic*, or *biological* nature. In the class of abiotic agents, laminar fluids have been used [52] to 'solve' mazes and more recently micrometre-sized abiotic beads have been used as computing agents [18] to solve the NP-complete clique problem. However, although the beads would bring some stochasticity into the computation, they do not have independent propulsion systems, as they are carried by (and follow) the minimum pressure paths of the fluids passing through the microfluidic network, thus not exploring comprehensively (and independently) the solution space. In principle, the Janus particle technology [53], in particular self-propelling anisotropic beads [54], could fulfil many of the desiderata outlined above, but presently their application appears to be limited by their size (mm range), generation of micro-bubbles (making them ineligible for movement in microfluidic networks) and possibly shorter lifetime of movement.

In contrast to the early development of potential abiotic computing agents, *biological agents* exhibit an extremely large variety – the result of evolution in motile biological systems, from

biomolecules to cells and multicellular organisms. Table 1 presents a synthetic comparison of the estimated performance of various biological systems attempting to solve SSPs.

Cytoskeletal filaments, which are aggregates of proteins, i.e., actin filaments, or microtubules, propelled by protein molecular motors, i.e., myosin, or kinesin, respectively, have the potential of fulfilling most of the technical requirements for motile computing agents. Indeed, both systems have been used to solve a small instance of the SSP [20]. The small size, reasonable velocity (in particular for actin filaments), distributed energy consumption (they require ATP (adenosine triphosphate) from the surrounding environment), and availability of elaborate biomolecular engineering techniques for tagging, functionalisation and splitting, are among the many advantages of cytoskeletal filaments. Presently, their further use as computing agents may be hampered by the 'open' architecture of the microfluidic devices required for easy access and renewal of ATP, leading to computational errors due to accidental loss or addition of filaments. Finally, the technology for multiplication of filaments, required by specific designs of NP-problem-encoded networks, such as SSP, is difficult.

Because network-based computing using biological agents is a relatively new development, presently only cytoskeletal filaments have been used in proof of principle bio-computation devices. However, unlike cytoskeletal filaments with only two types of agents, *prokaryotes*, comprising the large classes of bacteria and archaea, are vastly more diverse. While usually larger than cytoskeletal filaments, some of the bacteria [55] and archaea [56] can move at very high velocities, with body lengths per second one or two orders of magnitude higher than that of cytoskeletal filaments. Although the optimum in vitro growth conditions are not fully known for many of these rapid swimmers, for some, e.g., *Escherichia coli*, a large body of knowledge exists, including regarding a multitude of fully described, genetically engineered mutants. Additionally, prokaryotes can live in aerobic, or anaerobic conditions, making them amenable to various growth conditions in confined spaces.

Table I: Comparison of motility parameters for various biological agents (upgraded from [55], and [19])

Biological agents	Size		Velocity ($\mu m\ s^{-1}$)	B/s (s^{-1})	Arch (mm) SSP259	One-agent run time (s) SSP (2,5,9)	Booting frequency 2BL (Hz)	Computing time (hrs) SSPpr C30	Ref
	length (μm)	**width (μm)**							
Cytoskeletal filaments									
Actin filaments (myosin)	2	0.02	5	2.5	0.08	16	1.25	$5.1*10^6$	[20]
Microtubules (kinesin)	2	0.06	0.5	0.25	0.16	320	0.125	$5.1*10^7$	[20]
Prokaryotes									
Bacteria									
Pseudomonas aeruginosa	1.5	0.5	55	37	1.6	29	18.3	$3.5*10^5$	[57]
Chromatium okenii	9	4.5-6	45	5	15	333	2.5	$2.5*10^6$	[57]
Escherichia coli	2	0.5-1.5	16	8	1.6	100	4	$1.6*10^6$	[57]
Bacillus licheniformis	3	0.8-1.3	21	7	2.6	124	3.5	$1.8*10^6$	[57]
Sarcina ureae	4	2	28	7	6.4	229	3.5	$1.8*10^6$	[58]
Vibrio comma	4	0.45	200	50	1.4	7	25	$2.5*10^5$	[55]
Vibrio natriegens	2.5	0.4-0.6	14	5.6	1.3	93	2.8	$2.2*10^6$	[59]
Thiovolum majus	15	10	600	40	32	53	20	$3.2*10^5$	[55]
Archaea									
Methanocaldococcus jannaschii	1.5	0.5	380	253	1.6	4.2	127	$4.9*10^4$	[56]
Methanocaldococcus villosus	1	0.5	287	287	1.6	5.6	144	$4.3*10^4$	[56]
Eukaryotes									
Flagellated									
Cerratium fusus	420	15-30	235	0.56	48	204	0.28	$2.3*10^7$	[60]
Euglena viridis	53.3	10-17	80	1.5	32	400	0.75	$8.5*10^6$	[60]
Monas stigmata	6	6	270	45	19	70	22.5	$2.8*10^5$	[60]
Gyrodinium dorsum	32.8	24.5	328	10	78	238	5	$1.3*10^6$	[60]
Cilliated									
Tetrahymena sp.	70.4	20	500	7.1	64	128	3.55	$1.8*10^6$	[61]
Paramecium sp.	213	48	1000	4.7	154	154	2.35	$2.7*10^6$	[61]
Fungi									
Neurospora crassa	40	7	0.03	$7x10^{-4}$	22	733000	$(7.5\ x10^{-4})$	$8.5*10^9$	[47]
Pycnoporus cinnabarinus	67	5	0.033	$4x10^{-4}$	16	485000	$(4.9\ x10^{-4})$	$1.3*10^{10}$	
Nematodes									
Caenorhabditis elegans	1000	80	350	0.35	256	731	0.175	$3.6*10^7$	[62]

Notes: B/s = cell body length/s. Arch = device diagonal (mm). One agent run time (s) for SSP {2,5,9} problem in [20]. Booting frequency for double body length. Computing time (hrs) for SSP with cardinality 30 (first 30 prime numbers, SSP without cell division, Coupon Collectors correction). Underlined: High performance.

Finally, *eukaryotes* appear to have an even larger diversity than prokaryotes. However, their larger sizes, leading to larger areas required for computation, and their lower velocities relative to their body dimensions, translating into excessively long computing times, suggest that eukaryotes are unlikely to be serious contenders for efficient biological agents solving combinatorial problems. Instead, capitalising on their more complex space searching and space partitioning strategies [8, 47], eukaryotes are likely to offer insights into efficient natural algorithms, which can be subsequently reverse-engineered [49].

Sum-up. Network computing can benefit from an extremely large variety in biological agents of different nature, i.e., biomolecular, mono-cellular, or multicellular organisms, exhibiting various properties relevant to bio-computation, i.e., sizes, velocities, and motility mechanisms. In fact, this large variability of parameters makes the choice of biological agents for network computing difficult, as many other, less studied parameters, e.g., behaviour in confined spaces, could downgrade their expected performance in bio-computation.

2.4. Scaling of networks

2.4.1. Scaling the computing area and number of agents

The size of an SSP network is determined by (i) its unit cell size, designed for a specific computing agent; (ii) the cardinality of the problem, i.e., the number of elements in the set; and (iii) the compactness of the series, i.e., the relative distance between the numbers in the set.

The SSP unit cell size is determined by the geometrical parameters of the computing agents, e.g., width, length, and secure distance between two agents. The SSP cardinality determines the number of computing agents required to solve the problem, including some additional number to offset possible errors. Consequently, for a given compactness of the series, the size and the number of computing agents needed determine the area of the SSP computing system. In principle, a larger combinatorial problem requires, by necessity, a larger number of computing agents. However, network-based computing, as described before for SSP [19, 63] presents specific advantages, and disadvantages, regarding its scalability when compared to other massively parallel biocomputing approaches, e.g., DNA computing [9]. Indeed, DNA computing [9] requires an impractically large mass of DNA [13], as all the DNA mass needed for the calculation (approx. 2^C) must be simultaneously present in the reaction step, leaving the 'pruning' of all combinations to a sequence of post-computation biochemical selection processes. In contrast, in networks-based computation of SSP, the exploration of the 2^C computations paths is distributed in time and space, by recycling of agents. Consequently, network-based SSP calculation will use considerably less mass of agents, but at the expense of a much larger computation time.

Presently, network-based computing of SSP assumes [20] that the agents do not perform any function other than visiting junctions, and thus calculating various paths in the SSP-encoding network. In principle, as discussed further, the agents could perform additional functions, e.g., recording the history of their trajectories, and report on this at their exit, or in real time. However, this higher technological complexity of the agents, while valuable in accelerating the overall calculation, will not decrease the number of agents required to solve the problem, which is determined by the SSP cardinality. Moreover, it is possible that additional 'hardware' associated

with each computing agent will increase their size, thus increasing the overall area of the computing system.

2.4.2. Complexity classes

The SSP specifications (ii) and (iii) mentioned above determine together the total sum of the set. The compactness of the series also determines the type of complexity of the SSP network. Figure 6 presents the two complexity classes of the SSP, explained using three small example sets.

In *Complexity Class I* there is only one possible route to every legal exit, and consequently, there are only split- and pass junctions active. The series in the set is strongly expanding with the cardinality. For this case, the exponential series is shown, displayed in two forms: (i) with descending numbers (binary tree); and (ii) with ascending numbers and crossing traffic lines at pass junctions, but still with the same number of routes and exits (in compliance with the commutative property of addition).

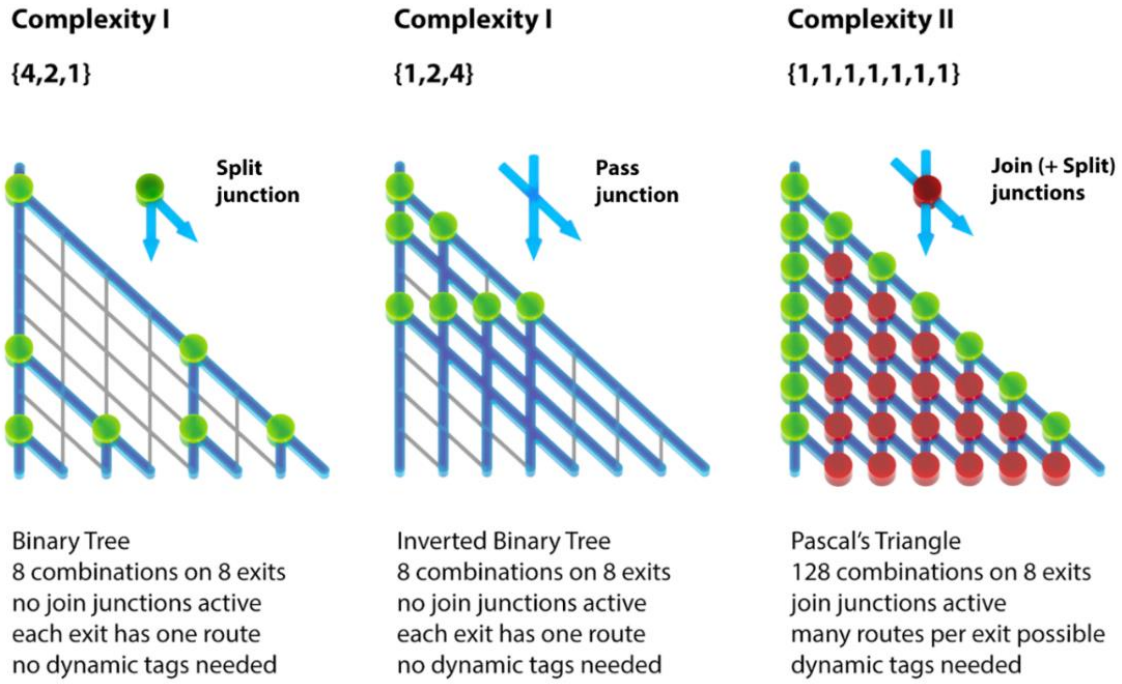


Figure 6: Sub-set sum complexity classes I and II explained in terms of split- and join junctions. The complexity class I networks {4,2,1} and {1,2,4}, are shown, as well as the complexity class II network {1,1,1,1,1,1,1}. (See main text in section 2.4.2 for a detailed description.)

Conversely, in *Complexity Class II* there are exits that can be reached through multiple routes and, hence, there are also join junctions active. The series in the Complexity Class II sets can be very compact. For instance, the most compact series possible is Pascal's Triangle. Tellingly, the set for Pascal's Triangle has cardinality 7, compared to cardinality 3 for the Binary Tree, but occupies the same area. The fundamental difference between the two complexity classes, i.e., single or multiple routes towards the legal exits, reveals the combinatorial (NP-complete) nature of the SSP problem: the solution to the problem goes beyond the discovery of the set of legal exits, also discriminating all possible routes towards these exits.

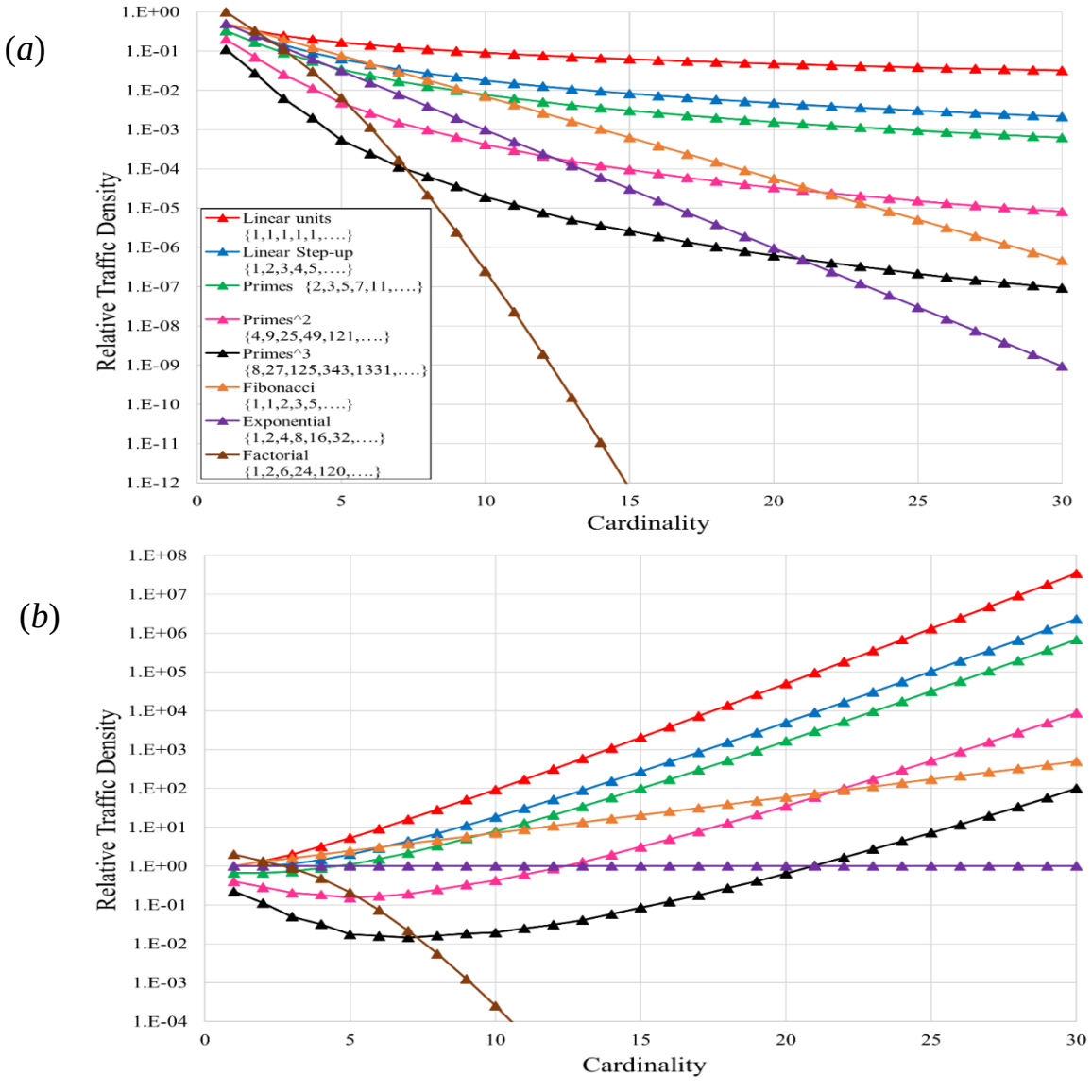


Figure 7: Relative average traffic densities $1/(\sigma+1)$ and $2^c/(\sigma+1)$ versus cardinality C in sub-set sum networks with total sum σ, for respectively (a) the combinatorial and (b) the multiplication run modes, shown for the most compact series {1,1,1,…, 1} (cf. Pascal's Triangle) to the airy exponential series {1,2,4,8, …., 2^c}, and beyond. Note that only the exponential series would show constant traffic density in case of the multiplication run mode**.**

The advantage of the compactness of Complexity II class comes, however, at yet another price; Figure 7 presents the relative average traffic density as a function of the cardinality in series with various degrees of compactness, for the combinatorial and multiplication run modes. In the combinatorial run mode, the traffic density is falling (orders of magnitude) for all series, and the bottleneck (risk of traffic jam) is located at the starting point of the network. Conversely, in the multiplication run mode, beyond a threshold cardinality value, the traffic density is rising (again, orders of magnitude) for most series, resulting in a traffic jam further down the network. Only the exponential series would show (with multiplication at the split junctions) a constant traffic density, but at the price of an exponentially expanding network size (and consequently also an exponentially rising computation time, as will be shown in section 2.5).

2.4.3. Scaling the readout

Solving SSP by means of network computing requires that the sequence of coordinates each and every agent passes through, or, at the very least, the sequence of the junctions it passes by, is fully recorded. This means that, until there is a reliable implementation that enables each agent to report this sequence, either 'on the fly', or at the end of the computation (to be addressed in section 2.7), the overall movement of all agents needs to be tested and optically recorded, at a precision in space and time, which will not permit errors regarding the history of the positions of each agent. Consequently, the tracks of all the agents should be captured, preferably, in one optical Field-of-View (FoV), and at a resolution allowing the identification of individual agents. Alternatively, if the overall computing area is too large to be visualised in one FoV, the optical recording needs to visit several sectors covering the overall movement, but at a frequency high enough to avoid confusion regarding the positioning or identity of the agents.

Three traffic scenarios should be considered, discussed in order of decreasing tracking complexity:

i. agents can crawl over- and cover each other in the channels (out-of-plane; *z*-direction);
ii. agents can overtake each other only laterally in the channels (in the *x*-*y* plane); and
iii. all agents move through the network channels in singular queues (no overtaking at all).

Note that channel widths and heights of <2 times the agent widths would prevent overtaking, but the risk of clogging is too large, therefore larger channel widths and heights (e.g., four times the agent widths) should be allowed in practice.

Obviously, the first scenario cannot be tracked error free, as optical tracking is performed in the *x*-*y* plane only; if one agent crawls over others, temporarily obscuring them, the tracking information becomes unreliable afterwards. Here, agents reporting their own travel history (as briefly mentioned above, and as will be elaborated in section 2.7) would be the only way to obtain reliable traffic information; this is how –in the end- a debugged large computing system should run. The second scenario would need a pixel size smaller than the agent width (and the agent length) in order to preserve reliable traffic information when agents pass each other (e.g. on the bottom of the channel).

In Figure 8, the expected chip size is shown as a function of the agent width for the prime numbers SSP devices for various cardinalities. The horizontal black dashed lines delimit the sizes of 4-, 6- and 8-inch silicon wafers – the standards in semiconductor industry. The vertical blue bars indicate the agent width for molecular motors-driven cytoskeletal filaments, i.e., actin filaments and microtubules, as well as for small (*E. coli*) and large (*E. viridis*) microorganisms. Because of the competition between resolution and the FoV [64], the whole imaging of the computing area requires the employment of the maximum useable pixel size (MUPS) that can still resolve individual agents, i.e. the MUPS value should be smaller than the agent width (and the agent length). The black crossed arrows indicate the intersection of the largest attainable FoV (as a square root) with the minimum attainable pixel size for various optical imaging technologies, i.e. their resolution limits. The useable optical range is obtained by the intersection of sqrt(FoV)-MUPS range with the diagonal black line indicated as ‘unity’. At the point where the top horizontal

border meets the 'Unity' line, the MUPS value is equal to the total FoV, meaning that only one pixel fits in the frame, obviously far from any reasonable application. To fully exploit the frame size available, the MUPS value should be as close as possible to the resolution limit. If the spot where the blue bars meet a specific diagonal cardinality line, is inside a 'technologically achievable' sqrt(FoV)-MUPS triangular area, then the corresponding optical technique is, in principle, useable for monitoring the computation process using a single FoV.

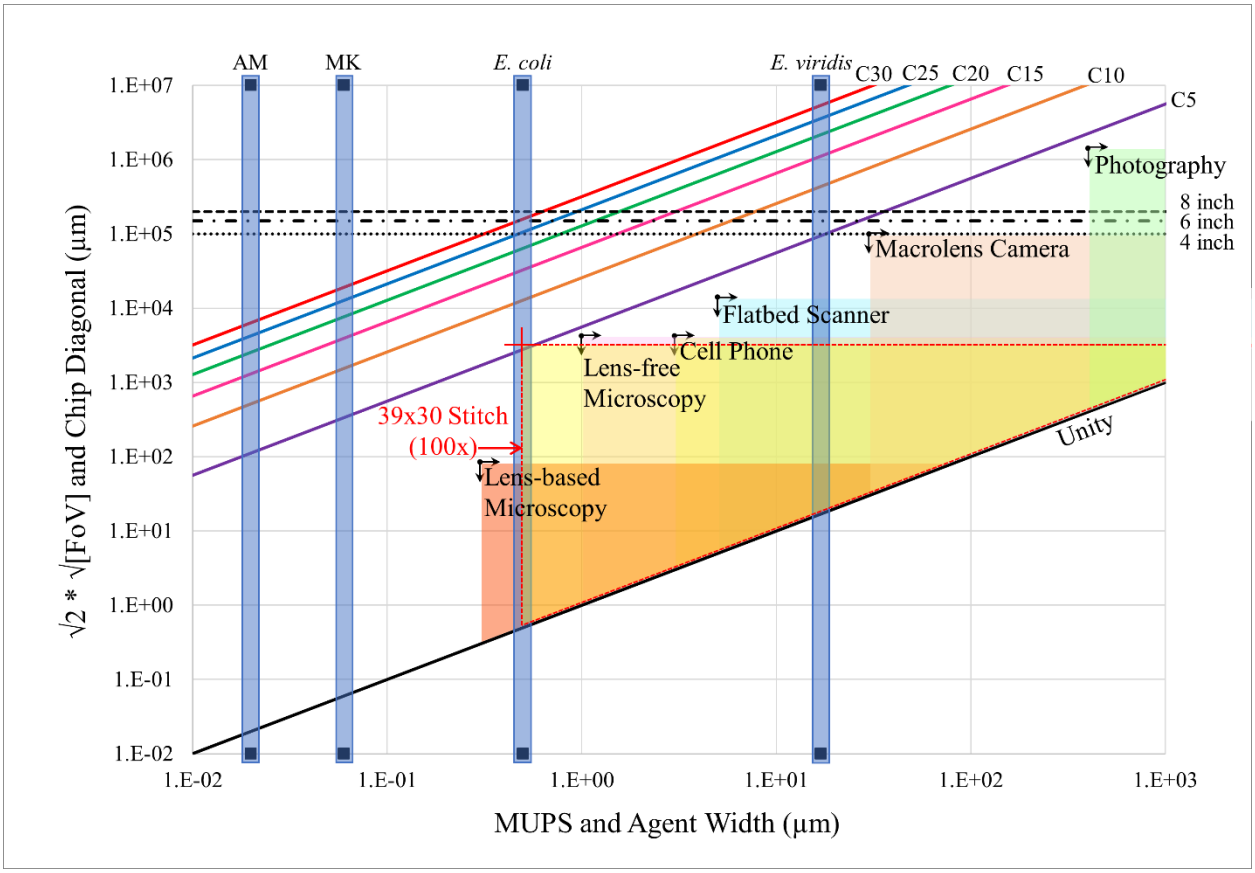


Figure 8: Chip size versus agent width and Field of View (FoV) versus Maximum Useable Pixel Size (MUPS). The triangular work windows are shown for various microscopy techniques. (See main text in section 2.4.3 for a detailed description). Also shown is an example of an enlarged work window by image stitching (red cross); the limits of this method are discussed briefly further below in section 2.4.3, and in detail in SI-3.

It follows from Figure 8 that actin and microtubule filaments are out of reach for optical monitoring, if the agent width should be resolved. The *E. coli* cell width can be resolved by a high resolution optical microscope, but the field of view would not allow more than one unit cell in one

FoV. For *E. viridis,* the FoV and resolution of a flatbed scanner would allow the capture of 3x3 unit cells in one frame.

The third scenario is described in detail in SI-2 and the nomogram in Figure SI-1. It follows that a network with cardinality 5 for *E. viridis* can be monitored by a macro-lens equipped camera, and that the cardinality 5 network for *E. coli* and the cardinality 15 network for microtubules can be monitored by a lens-less microscope, all in one FoV, but under the naïve assumption that no agents are overtaking each other in the channels.

When the area to be imaged (and monitored in time) exceeds the FoV of the imaging system, a powerful option to enlarge the effective FoV is image stitching of cyclic sampled frames. The loss of information can be minimised through faster switching speeds, which in turn are limited by the mechanical capabilities of the microscope stage. In the SI-3, the possibilities and limits of image stitching for our SSP calculation networks are modelled. In the case of high density traffic, agent speed and body length determine the sample frequency, and in the case of low density traffic, agent speed and junction distance are decisive. In Table SI-I it is shown that in a typical setting used to monitor *E. coli* in the SSP prime numbers network, stitching could indeed be employed to image and monitor larger SSP networks. For traffic scenario (iii), at a resolution of 2 µm with a 10x objective, instead of a cardinality 4 network in one FoV, a cardinality 15 network can be monitored in time by (cyclic) stitching of 14x11 frames (shown by a red cross in Figure SI-1). For traffic scenario (ii), at a resolution of 0.5 µm with a 100x objective, a cardinality 5 network can be monitored in time by (cyclic) stitching of 39x30 frames (shown by a red cross in Figure 8).

One corollary of the above analysis is that, for *E. coli*, or an agent with similar motility and size parameters, the area of a device solving a prime numbers SSP with a cardinality of 30 is slightly larger than a 6 inch wafer. As this area cannot be captured in one FoV by any known optical monitoring system with the proper resolution needed, the readout would need an array of 60x60 frames of a lens-less microscope being continuously switched in order to keep track of all agents simultaneously. Even if that were technically feasible, the data storage needed would be very large. Moreover, from a fabrication point of view, such large chips are very vulnerable to fatal errors by dust particles in the lithographic steps. Indeed, one dust particle on a wafer with 100

chips lowers the yield from 100 to 99%, but on a one-chip wafer it leads to 0% yield (i.e. 100% failure). Alternatively, a purposefully designed and fabricated optical chip, built in the 'floor' of the computing chip, with pixels smaller than half the channel width (or even better, half the agent width), and covering only the actual computing area, is a technologically achievable, albeit non-trivial solution.

Sum-up. It appears that the scaling of networks, in particular for solving SSP, is the most problematic, albeit technological and not fundamental, aspect of network computing with biological agents. Indeed, the chip area, which grows with the size of the problem, requires FoVs which are not presently available. Alternatively, to limit the explosion of the chip area with the size of the problem would require smaller agents, which in turn would require a higher resolution, but this would further raise problems for the achievable FoV. Ultimately, a technology that allows the agents to report their own travel history (at the exits), would not need optical recording of the total network.

2.5. Computing time

2.5.1. Computing time versus run modes

In the first instance, the time to solve an SSP depends on the mode of operation of the computing agents, the extent of the series, i.e., its cardinality, and the structure of the series of numbers. More compact series will result in a smaller computing area and consequently a shorter computing time. Figure 9 presents the relationship between the estimated computing times for *E. coli* (Table I) in the three run modes (detailed in section 2.2.2) versus the longest track in the SSP chips for four number series (four compactness types: Pascal series, prime numbers, Fibonacci series, and exponential numbers) and cardinality (shown as a label in steps of 5 next to the calculated points). For a given series and given cardinality, the track length is the same for all run modes.

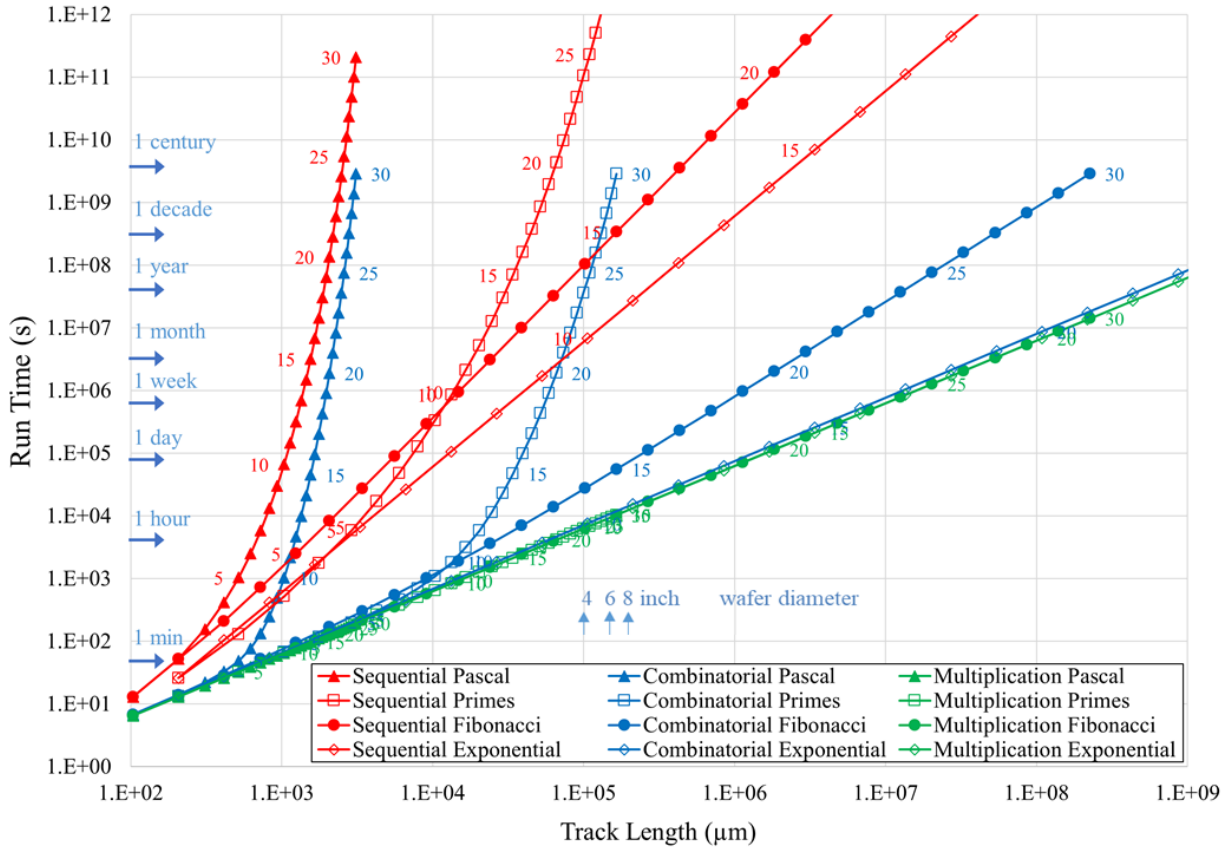


Figure 9: Run time versus track length as a function of cardinality for sub-set sum networks with 2 µm track width, 16µm/s agent speed and 2 µm body length (*E. coli*). A correction factor derived

from the ‘Coupon Collectors Problem’ [51] (section 2.2.2) was included to deal with the stochastic nature of the parameter exploration in the combinatorial run mode.

As expected, the highest computing times are observed for the sequential run mode, and the lowest are observed for the multiplication run mode. The difference in run time between the sequential and the combinatorial run modes is small for compact series, but quite large for expanding series. Importantly, in the combinatorial run mode, the estimated run times at higher cardinality become independent of the compactness of the series, due to the fact that the total booting time needed to accommodate large numbers of agents in the network is orders of magnitude larger than the time needed to run a single track (compare also the booting frequency data in Table I, explained in Figure 5, section 2.2.2). The multiplication run modes for the various series are all following the same straight line because, effectively, only one agent starts and the off-spring that takes the longest track is monitored, but all are assumed to run with the same average speed.

In Figure 9 the sizes of 4, 6 and 8 inch wafers are indicated (by blue arrows) for the possible fabrication limits. A network with cardinality 30 would only fit on a standard wafer for the prime number (and the Pascal) series. Also indicated, by blue arrows, are time frames. Only the multiplication run mode would allow a cardinality 30 network to be run in a reasonable time.

2.5.2. Benchmarking biological agents based network computing with electronic computing

While electronic computers perform computations in a serial manner, they are many orders of magnitude faster per operation than it is reasonable to expect from network computing with biological agents. Consequently, the immediate scaling question is to what extent an *ideal* set of agent parameters, i.e., size, speed, multiplication rate, which inform the design of the computing network, would make network computing using biological agents possibly competitive with the electronic computers. In order to have a comparison between the ideal performance of network

computing with biological agents, and electronic computers, two sets of simulations have been performed.

At this junction an important distinction must be made when comparing the performance of electronic computers with any other alternative computing devices, including the one recently proposed for solving SSP [20]. It was argued [65] that SSP has a known solution that runs in O(NT) time, and that there are algorithms, e.g., Pisinger's [66], which can solve SSP very quickly if run by electronic computers. However, the alternative computation approaches, including DNA, quantum, and networks-based computing, to name a few, propose in the first instance computing *devices* with associated *operational procedures*, rather new algorithms, which indeed might be required to be developed to capitalise on the potential benefits offered by the new computing hardware. Consequently, and taking into consideration the tentative or early stage of development of the new computing devices, any meaningful comparison of the computing power of electronic computers and any new paradigmatic computing device must use comparative algorithmic procedures, rather than the most advanced ones, which by virtue of decades long history of microelectronics have been solely and specifically created and optimised for sequential electronic computers.

On this background, a computer program was designed to solve the SSP by brute force (i.e. no efficient 'heuristic' algorithms have been used) for electronic computers, using emulators of various generations of computer chips. To ensure a more conservative approach, the program has been coded in C++ to allow the maximum use of computer chip RAM, low-level memory access, efficient mapping to machine instructions and flexibility. The program is described in detail in SI-4. Separately, the operation of a network computer using biological agents, both used before and hypothetical, has been simulated for selected agents from Table I, for cardinalities considerably larger than presently possible in experiments. The program is described in detail in SI-5.

Figure 10 presents the estimated run times for solving an SSP problem by means of network computing [20], with various cardinalities, using (i) biological agents, either cytoskeletal filaments, i.e., actin filaments propelled by myosin and microtubules propelled by kinesin, or hypothetically, several bacterial agents (exhibiting superior parameters): *M. janaschii*, which has

a high speed, *M. villosus*, which, due to its small size, calculates faster, *V. natriegens*, which multiplies frequently and -as a reference- *E. coli* (see Table I), and (ii) various generations of computer chips i.e., Intel's 286, 386, 486 and single core Pentium and a present-day MacBook chip. As opposed to all electronic chips, which perform computation in a sequential run mode, the simulated computation by biological agents is performed in the combinatorial run mode, for cytoskeletal filaments and the chosen bacterial agents, and in the multiplication run mode for the latter, assuming multiplication rates reported in the literature; the following doubling times have been used: *M. janaschii*: 74 min., *M. villosus*: 45 min., *E. coli*: 30 min., and *V. natriegens*: 15 min.

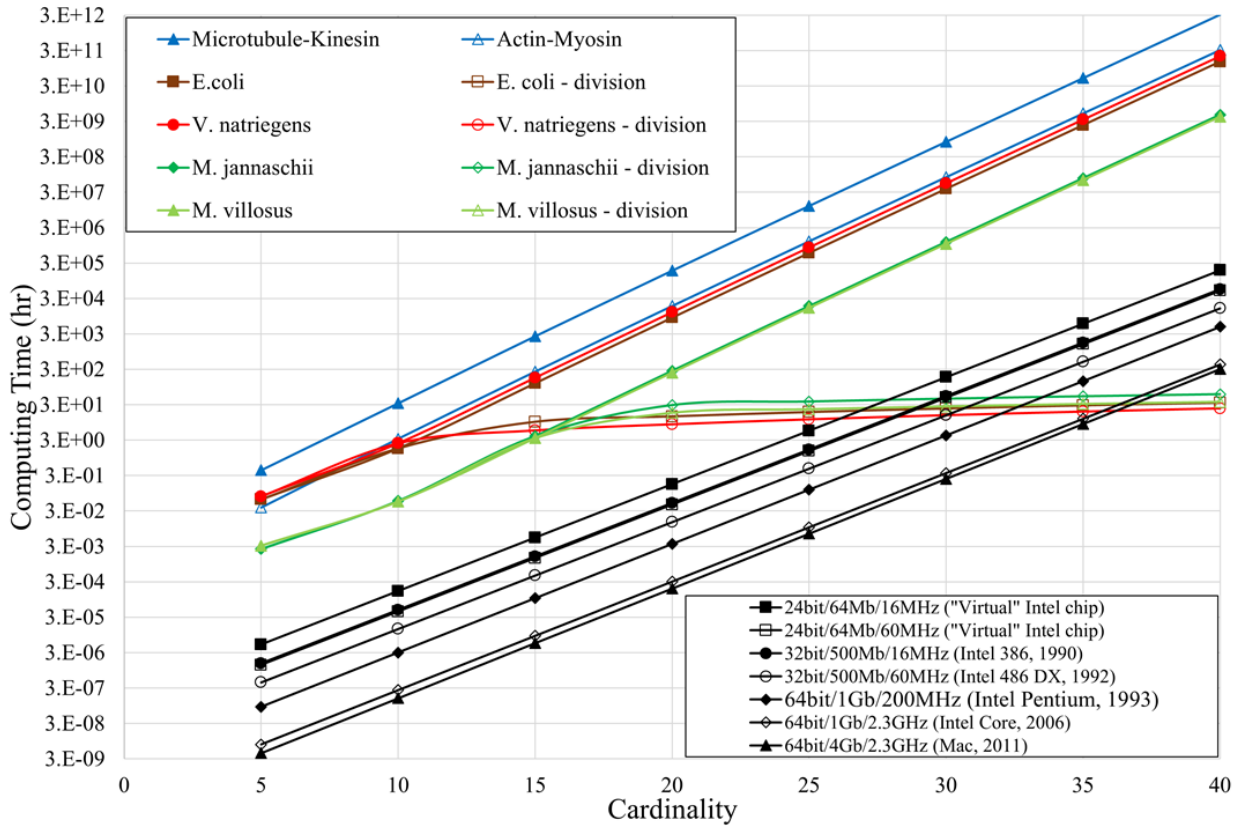


Figure 10: Comparison of the computing performance of the electronic computers (bottom-right half) and biological computers (top-left half) solving the prime numbers Subset Problem. Correction factors for the 'Coupon Collectors Problem' and for a 45-55% instead of 50-50% split junction distribution were included.

Even a cursory inspection of the computing performance comparison of the electronic and network-based computers reveals several evident trends.

- As a group, the electronic chips, operating in the sequential run mode, outperform — by a few orders of magnitude — the biological ones operating in a combinatorial run mode. Moreover, this performance gap remains constant, or increases slightly, throughout the range of cardinalities tested. Indeed, even if electronic computers operate in the sequential run mode (Figure 5), they also operate at clock frequencies in the order of GHz, whereas the biocomputers will operate at typically 0.1 to 10 Hz (Table I).
- The difference in performance of electronic computers computing SSP, i.e., in [20] which used RAM-intensive software (MATLAB), vs. the present study, which inheres from a more efficient use of RAM, also reveals the importance of the allocation of chip memory, a perennial problem for electronic devices. It is important to observe that eventually any electronic chip solving SSP (or any NP-complete problem) will hit an intrinsic "memory wall"[67] when all chip memory is used for ever larger problems. Note that for the cardinality at which this happens, it will result in a truncation of the black lines in Figure 10 at the right hand side. In contrast, and aside from other physical limitations (assessed in the previous sections, e.g. chip area, readout), network computing using biological agents should not experience any similar "memory wall".
- The motility speed of the biological agents appears to have only a secondary, albeit positive, effect on the performance of network computing, but by itself it will not be able to make the bio-computer outperform the electronic one. Indeed, the speed of the faster biological agent, i.e., *M. villosus*, would need to be raised from its already high value of 287 μm/s three orders of magnitude, i.e., approximately 30cm/s, only to catch up with the slowest electronic chip tested, i.e., Intel's 286. On the other hand — and crucially —it is estimated that biocomputers operating in the multiplication mode quickly outperform electronic chips, almost independent of the clock speed of the latter, as shown in Figure 10.

Sum-up. While some improvement can be achieved, in principle, using faster biological agents operating in the pure combinatorial run mode, the computing performance of electronic computers will remain unmatched for the foreseeable future. While more analysis would be required to explore the possible collapse of performance of (single core) electronic computers for larger SSP problems ("memory wall"), a more advantageous avenue will likely be based on the use of biological agents running in a multiplication run mode.

2.6. Scaling the energy required for computation

With alternatives to ‘classical’ electronic computers not being fully demonstrated, or most likely being in early stages of development, presently only high performance computation (HPC) is the closest to tackling large scale combinatorial problems. However, in itself, the scale of combinatorial or complex problems of practical importance translates into large amounts of energy used, if the computation is performed by sequential electronic computers. For instance, solving large complex problems, even if not necessarily combinatorial in nature, would require scaling-up HPC to exascale computing, i.e., 10^{18} floating-point operations/s (Flops) [68]. However, as the most powerful supercomputer, Sunway TaihuLight, requires 42 MW of power (an average hydroelectric facility generates 57 MW), scaling it to exascale regime would require 450 MW, with running costs of US$270 mil./year [69]. Arguably, a similar result regarding energy consumption would be obtained for a technical solution involving myriads of smaller scale PCs interconnected in a very large computer network, such as a very large ‘computer farm’. Consequently, and aside from the difficulty of solving large combinatorial problems, it appears that electronic computers are also unsustainable energy-wise.

The computational systems able to solve, in principle, combinatorial problems, can be aggregated into three classes (Table II). The most energy-efficient systems are, expectedly, *molecular computers*, of which the most well-known is DNA computing [9], followed by numerous variations [70]. Indeed, since in molecular computers, the mathematical operations *are*, actually, chemical reactions, the energy/operation required by DNA computing is the closest to the thermodynamic limit calculated elsewhere [71]. At the opposite end of the spectrum considered, *silicon-based computers*, including ‘classical’ HPC and quantum computing, are seven to eleven orders of magnitude more energy consuming, per operation, compared to molecular computing (and there are two orders of magnitude between the most performant HPC system and an advanced quantum computing system).

Table II: Energy efficiency of various computing systems

System	Implementation	Measure-ment	Energy J/operation	Explanations, (sources)
Molecular computers	Thermo-dynamic limit	Theory	2.90×10^{-22}	Thermodynamics [71]
	DNA	Estimated	5.00×10^{-20}	Thermodynamics [9]
Si-based computing	Electronic	Actual	1.65×10^{-10}	Sunway TaihuLight [72], [1]
		Actual	5.88×10^{-11}	Shoubu system B [72], [2]
	Quantum	Estimated	2.00×10^{-13}	DWave system [69]
Computing with networks	Microfluidics	Estimated	1.29×10^{-12}	Beads in microfluidics [18], [3]
	Cytoskeletal filaments/mole-cular motors	Estimated	4.95×10^{-14}	Kinesin/microtubules [20]
		Estimated	2.00×10^{-14}	Myosin/actin filaments [20]
	Micro-organisms	Estimated	1.43×10^{-13}	*Escherichia coli* [4]
		Estimated	2.76×10^{-13}	*Vibrio natriegens* [4]
		Estimated	8.67×10^{-14}	*Methanocaldococcus jannaschii* [4]
		Estimated	1.16×10^{-13}	*Methanocaldococcus villosus* [4]
		Estimated	2.01×10^{-9}	*Euglena viridis* [4]

Notes. [1] Top computing speed for top 500 supercomputers in 2017. [2] Top energy efficiency for top 500 supercomputers in 2017. [3] Pumping energy for a chip with d=200nm x L=1000nm, cf. [20]; using capillarity principles will result in considerable lower energy consumption. [4] Energy consumption estimated using the general formula for energy consumption in motility of prokaryotes [73, 74]

Importantly, the energy consumption for silicon-based computers reported in Table II includes only the energy required 'core phase', that is, for the section of the workload that undergoes parallel execution. It typically does not include the parallel job launch and teardown, which is required to run for at least one minute. Consequently, no energy consumption is reported for environmental, e.g., cooling, and auxiliary, e.g., lighting, needs. Finally, systems performing *computing with agents exploring networks* present an estimated energy consumption/operation in between molecular and Si-based computing, but over a very large range, i.e., between thirteen to six order of magnitude higher than molecular computers. As with silicon-based computers no energy consumption is estimated other than that for computation proper. Within this class is microfluidics-based computation, which relies on beads being pushed, with some level of randomness, through networks encoding an NP-complete problem [18] (although it is possible that the energy required by microfluidics-based computation could be decreased substantially by using capillary-driven flows). The exploration of a network by larger microorganisms, e.g., *Euglena*, has an energy consumption/operation similar to the Si-based computers. However, the use of nano- or small micron-sized biological agents, i.e., cytoskeletal filaments [20], or bacteria [73, 74], respectively, is estimated to bring the energy consumption/operation one order of magnitude down, or similar to that of an advanced, and energy consumption-competitive quantum computing system.

Sum-up. The estimated energy consumption per operation for network-based computing using micro- or nanometre sized biological agents is in the range of 10^{-14}-10^{-13} J/operation (and much larger for tens of micrometre-sized agents), which is similar to the reported energy performance of quantum computation, and three to four orders of magnitude better than present supercomputers. Additionally, biological computers would have the advantage of distributed energy consumption, in contrast with Si-based computers, including quantum computers.

2.7. Perspectives and future work

The scaling analysis presented above identifies several challenges ahead for the further development of network computing using biological agents, taking the solving of SSP as a benchmark case. These challenges are either fundamental or related to the underdevelopment of the presently available service technologies required. To this end, further areas of research and development, as well as under-utilised opportunities, are as follows:

Recording the traffic history on each individual agent

Unlike SSPs of Complexity Class I (Figure 6), Complexity Class II problems constitute "true" combinatorial problems. The consequence is that agents that have taken different routes towards the same exit have to be clearly discriminated in order to be able to solve the combinatorial problem. To demonstrate, by counter-example, the SSP presented in Figure 5 can be replaced by a very fast electronic device, consisting of parallel arrays of switching transistors (Supplementary Information, SI-6). This device, which has all the architectural characteristics of a network-based computer, but which lacks the capability to differentiate between computing agents, shows the correct exits essentially instantaneously, but as the 'agents', i.e., the electrons, are 'anonymous', the routes of the individual agents cannot be discriminated. Consequently, this very fast device is not truly able to solve combinatorial problems. The area and the energy needs of this device scale quadratically with the total sum in unary coded form, which in turn scales exponentially with the regular binary coded representation of numbers used in sequential electronic computers.

Individual bacterial agents in a network can be monitored by video tracking techniques, but for higher cardinality problems, the amount of (image) data to be stored and interpreted will rise exponentially. For instance, for a cardinality 30 problem, more than 1 billion agents will have to be tracked simultaneously. Instead of the troublesome high resolution tracking (in time) of identical agents simultaneously, as described in section 2.4.3, one could discriminate each individual agent by adding a unique static label (i.e. a label that is not changed during the run time of the experiment) and lower down the image capturing frequency. However, still full (video) tracking of all the agents would be necessary to retrieve all the routes taken by individual agents. For instance, for a cardinality 30 problem more than 1 billion agents with unique labels would be

needed – a clear technological impossibility. As there are most probably not enough unique labels available, one could try to employ a limited selection of labels to compose binary coded 'words' (2^{30} words by using 30 labels). For very large networks, however, the process of coding and decoding of the labels may constitute in itself an operation rising exponentially in time.

A dynamic labelling system "Tag & Trace", however, could store the necessary information about the route followed by the individual agent, on the agent itself, as shown in Figure 11. At every split junction the agent proceeding in the direction associated with the addition of that particular number will get a label (a 'stamp', or a tag). At the exits, the agents will be interrogated which labels have been collected on the route towards that particular exit. In this way, the routes of individual agents arriving at joined exits can be discriminated, and combinatorial problems can be solved. Although for a cardinality 30 problem still more than 1 billion agents with 1 billion label 'words' composed of 30 unique labels collected 'on the fly' would be needed, the video tracking (with the image data explosion and image stitching, described in section 2.4.2) would no longer be necessary.

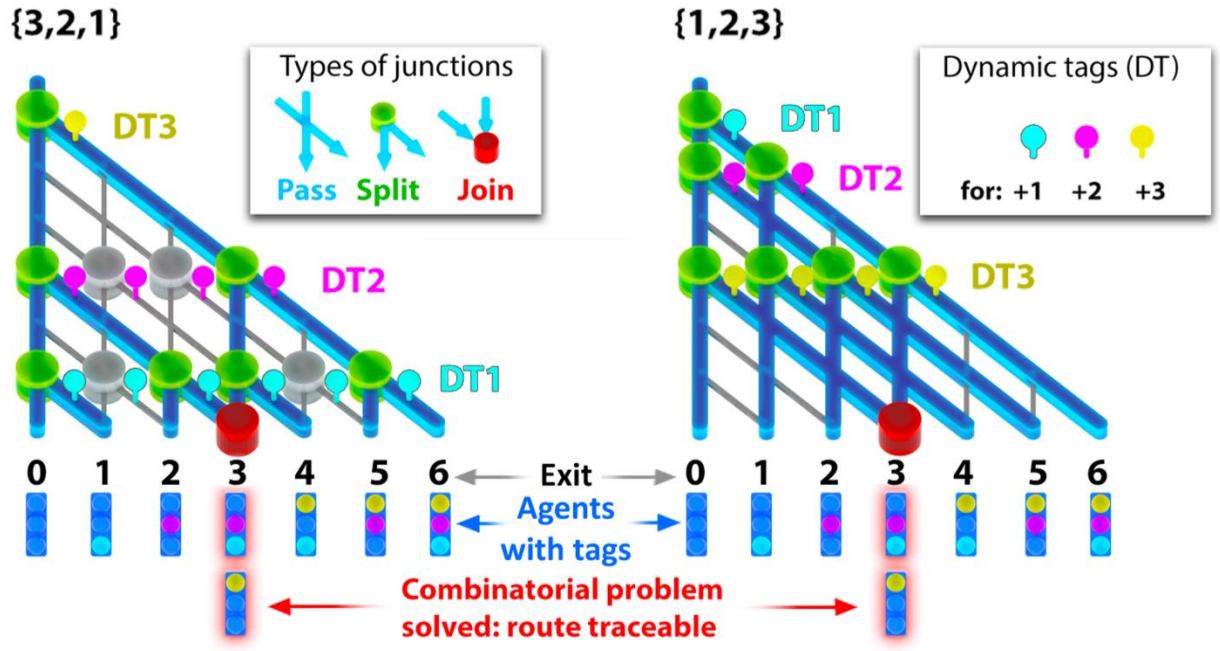


Figure 11: Dynamic tags for solving combinatorial problems in the very small complexity class II networks {1,2,3} and {3,2,1}, are shown. (See main text in section 2.7 for a detailed description.)

Importantly, when multiplication of agents in the network can be employed to address all possible combinations (in linear time), as e.g., in the proposed multiplication run mode, it is essential that all labels collected 'on the fly' are precisely copied at every multiplication event, or otherwise the information about the route -taken so far- is lost.

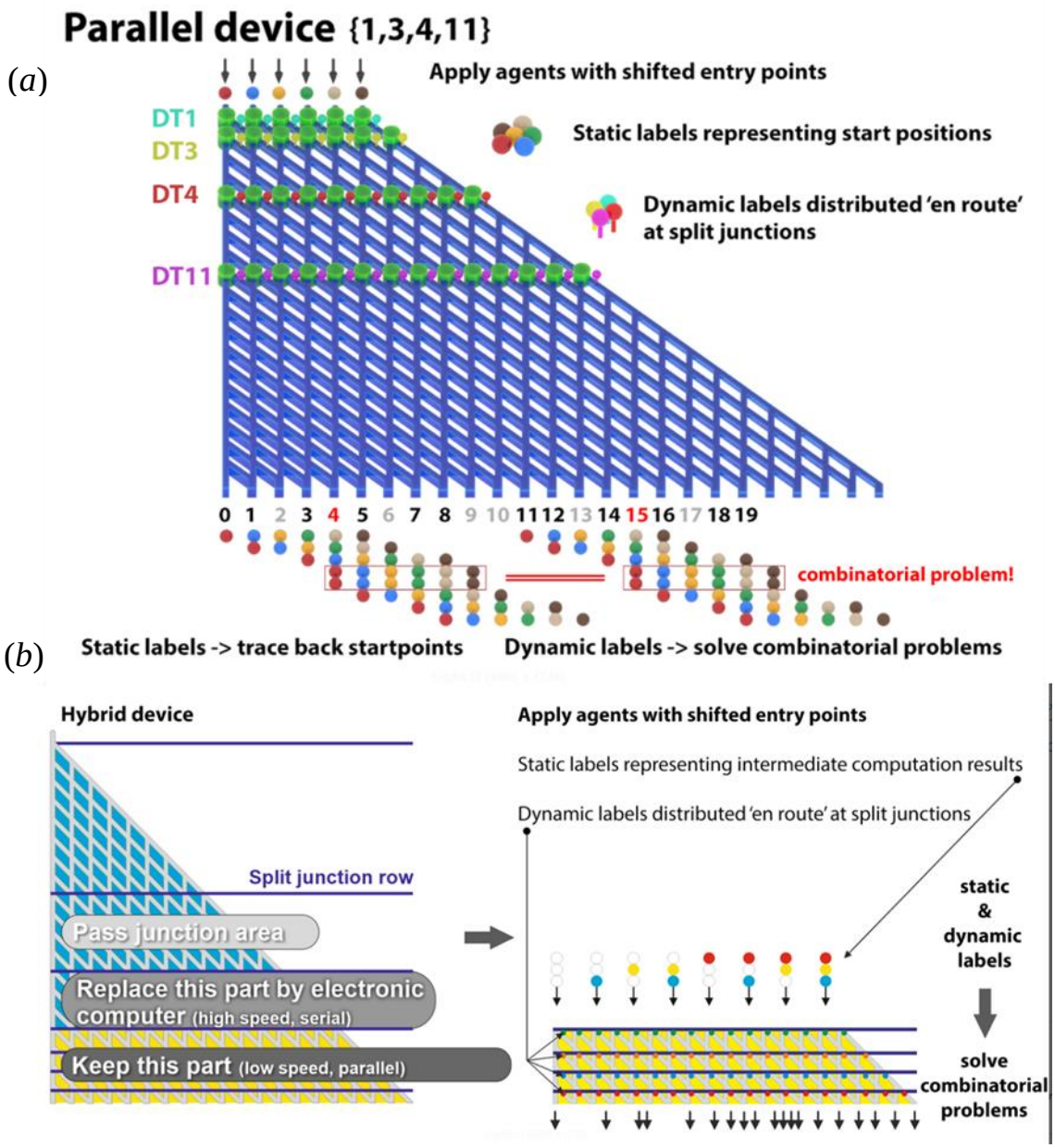


Figure 12: Static tags for parallel computation in: (a) a parallel device and (b) a hybrid bio-electronic device. Additionally, dynamic tags are needed for solving combinatorial problems. (See main text in section 2.7 for a detailed description.)

If the traffic density rises in compact networks, as shown in Figure 7, and therefore clogging may occur, other methods for obtaining massively parallel operations need to be considered. When the multiplication of the agents cannot be employed to address all possible combinations, the total calculation time can be lowered by parallel processing.

In the combinatorial run mode, the choice of the combinations explored by the agents is a stochastic process. Therefore, the total calculation can be performed in a shorter time using distribution over multiple identical networks (parallel processing). This can be done on separate chips, or by applying parallel traffic in one network. In Figure 12a a parallel subset sum computation device is shown, where the agents are applied to various shifted starting points. At every starting point, a unique static label is attached to the agents. After running through the network, at every exit the labels are checked, and in this way the starting point can be retrieved, and the effective exit number obtained. Apart from the static labelling method for enabling parallel processing, dynamic labelling is needed simultaneously for solving combinatorial problems. After running through the network, at every exit the static and dynamic labels present on each and every agent have to be checked ‘on the fly’.

Additionally, when multiplication of the agents cannot be employed to address all possible combinations, another option to speed up solving combinatorial problems consists of employing the best of two worlds [63]: in Figure 12b a Subset Sum network is displayed with descending numbers. A hybrid device could be created by replacing the first part of the network by an electronic computer (serial, but high speed) while leaving the rest of the device for the bio-computer (low speed, but parallel). The agents are applied with shifted entry points calculated from the intermediate computation results of the electronic computer, and static labels are applied to the agents, in which labels code for the virtual ‘route’ taken so far, as calculated by the electronic computer. Apart from this static labelling, the dynamic labelling is again needed in the bio-computer part of the network for solving combinatorial problems. Likewise, after running the biological part of the network, at every exit the static and dynamic labels present on each and every agent must be checked ‘on the fly’.

Development of new designs of computing networks:

- While the proposed network computing approach for solving SSP [20] uses a brute force method implemented through a *physical* device, it has been argued [65] that SSP can be efficiently solved by efficient software *algorithms* without the need for alternative elaborate hardware. Consequently, SSP is likely to remain a benchmark method testing the prowess of various combinatorial computation methods [63], rather than finding other, more tangible applications. This limitation demonstrates the need for further approaches for encoding of other NP-complete problems in graphs, and subsequently into networks and computation devices, with immediate, but not exclusive, examples being the TSP and 3-SAT.
- The fundamentally new designs will also benefit from further, second order improvements, e.g., better area compactness using a 3D architecture of the chip, as proposed for cytoskeletal filaments [75], dynamic logical gates, as opposed to the present static pass- and split junctions [20], and hybrid electronic/network-based devices [20].

Biological agents and operation modes:

- The immediate realisation when contemplating the parameters of biological agents and their possible run modes is that, at least for solving SSP via the proposed approach [20], the only pathway to achieve a better computing speed than electronic computers is to enable biological agents run in the multiplication mode. Indeed, as the chips encoding NP-complete problems grow exponentially in some parameter, e.g., area or number of agents, the only option to counterbalance this trend is to use another exponential, i.e., multiplication. The multiplication of biological agents occurs naturally for microorganisms, but could be achieved, in principle, with cytoskeletal filaments too, by hijacking the biomolecular treadmilling.

- Separately, it should be noted that solving SSP by network computing using biological agents in the combinatorial run mode, does not suffer from the scaling limitations regarding the mass of agents, which is the major bottleneck in DNA computing [13], where all the DNA mass needed for the calculation must be simultaneously present in the reaction step (section 2.4.1). In networks-based computation of SSP, the exploration of the computations paths is distributed in time and space, by recycling of agents. Consequently, network-based SSP calculation will use considerably less mass of agents, but at the expense of a much larger computation time.

- The strategy of hard-wiring of computing tasks into a physical device should be extended to the computing agents. Indeed, presently the agents are passively exploring the allowable paths, translating into difficult to achieve tasks for the readout system, but in principle biological agents, if appropriately tagged, e.g., using fluorophores responsive to the local environment, can perform computing tasks 'on the fly', e.g. by recording autonomously the history of the visited gates (Figure 11), or by 'beaming up' events, as previously proposed for cytoskeletal filaments [76, 77].

Tug of war between area and read-out:

- The only option for solving large SSPs is to design and fabricate wafer-large optical chips, which is in principle achievable with present technology, but at a high cost and with high fabrication failure risk. A possible improvement would be to provide the readout, at the appropriate resolution, in the network paths only.
- An alternative would be to switch from an area-based, to an agent-based readout, if as suggested above, the biological agents might record their travelling history. This readout option is indeed used by DNA computing, with the difference that in the network computing case only a smaller number of agents would be interrogated at a time. Also, it is very likely that the optical readout, which is fast, would remain the technology of choice.

Energy:

- The estimated energy consumption per operation is already competitive with electronic computers, but there are various opportunities to increase this energetic efficiency. For instance, actual measurements of energy consumption, instead of the estimations (Table II) can reveal better energetic efficiency, in particular for biological agents belonging to the Archaea.
- Another energy-related area is the *sustainability* of computation, rather than its energetic efficiency. Indeed, while *E. viridis* appears to use orders of magnitude more energy than other biological agents (Table II), it can use light as a source of energy [78].

2.8. Conclusions

In brute force computing, for a problem of input size C, 2^C solutions have to be addressed, and this workload needs to be distributed in space and time: something has to give. The technological challenges related to scaling up the size of the problems have been identified in terms of chip fabrication, readout reliability and energy efficiency. The necessary computing time of parallel operating biological agents has been compared to the electronic single CPU computers. Labelling of biological agents 'on the fly' with accompanying readout of their travel history at the exit, seems a promising new development avenue for tackling combinatorial problems.

Competing interests

We have no competing interests.

Funding

This work was financially supported by the Defense Advanced Research Projects Agency under Grant Agreement (DARPA) grant no. HR0011-16-2-0028; and by the European Union Seventh Framework Programme (FP7/2007-2011) under Grant Agreements 228971 [MOlecular NAno Devices (MONAD)][and 613044 [Parallel computing based on designed networks explored by self-propelled, biological agents (ABACUS)].

References

1. Nam, G.J., K.A. Sakallah, and R.A. Rutenbar, *A new FPGA detailed routing approach via search-based Boolean satisfiability.* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2002. **21**(6): p. 674-684.

2. Fraenkel, A.S., *Complexity of protein folding.* Bull. Math. Biol., 1993. **55**(6): p. 1199-1210.

3. Pierce, N.A. and E. Winfree, *Protein design is NP-hard.* Protein Engineering, 2003. **15**(10): p. 779-782.

4. Hopfield, J.J. and D.W. Tank, *"Neural" computation of decisions in optimization problems.* Biological Cybernetics, 1985. **52**(3): p. 141-152.

5. Massacci, F. *Contextual reasoning is NP-complete*. in *Proceedings of the National Conference on Artificial Intelligence*. 1996.

6. Brandes, U., et al., *On modularity clustering.* IEEE Transactions on Knowledge and Data Engineering, 2008. **20**(2): p. 172-188.

7. Aaronson, S., *Guest Column: NP-complete problems and physical reality.* SIGACT News, 2005. **36**(1): p. 30-52.

8. Hanson, K.L., et al., *Fungi use efficient algorithms for the exploration of microfluidic networks.* Small, 2006. **2**(10): p. 1212-1220.

9. Adleman, L.M., *Molecular computation of solutions to combinatorial problems.* Science, 1994. **266**(5187): p. 1021-1024.

10. Lipton, R., *DNA solution of hard computational problems.* Science, 1995. **268**(5210): p. 542-545.

11. Mao, C., et al., *Logical computation using algorithmic self-assembly of DNA triple-crossover molecules.* Nature, 2000. **407**(6803): p. 493-496.

12. Qian, L. and E. Winfree, *Scaling up digital circuit computation with DNA strand displacement cascades.* Science, 2011. **332**(6034): p. 1196-1201.

13. Beaver, D., *Computing with DNA.* Journal of Computational Biology, 1995. **2**(1): p. 1-7.

14. Braich, R.S., et al., *Solution of a 20-variable 3-SAT problem on a DNA computer.* Science, 2002. **296**(5567): p. 499-502.

15. Ouyang, Q., et al., *DNA solution of the maximal clique problem.* Science, 1997. **278**(5337): p. 446-449.

16. Reif, J.H., *Scaling up DNA computation.* Science, 2011. **332**(6034): p. 1156-1157.

17. Ladd, T.D., et al., *Quantum computers.* Nature, 2010. **464**(7285): p. 45-53.

18. Chiu, D.T., et al., *Using three-dimensional microfluidic networks for solving computationally hard problems.* Proceedings of the National Academy of Sciences of the United States of America, 2001. **98**(6): p. 2961-2966.

19. Nicolau, D.V., et al., *Molecular motors-based micro- and nano-biocomputation devices.* Microelectronic Engineering, 2006. **83**(4-9 SPEC. ISS.): p. 1582-1588.

20. Nicolau, D.V., et al., *Parallel computation with molecular-motor-propelled agents in nanofabricated networks.* Proceedings of the National Academy of Sciences of the United States of America, 2016. **113**(10): p. 2591-2596.

21. Caulfield, H.J. and S. Dolev, *Why future supercomputing requires optics.* Nature Photonics, 2010. **4**(5): p. 261-263.

22. Schnorr, C.P. and M. Euchner, *Lattice basis reduction: improved practical algorithms and solving subset sum problems.* Mathematical Programming, Series B, 1994. **66**(2): p. 181-199.

23. Kate, A. and I. Goldberg, *Generalizing cryptosystems based on the subset sum problem.* International Journal of Information Security, 2011. **10**(3): p. 189-199.

24. Badawi, A.A., et al., *Accelerating subset sum and lattice based public-key cryptosystems with multi-core CPUs and GPUs.* Journal of Parallel and Distributed Computing, 2018. **119**: p. 179-190.

25. Fréville, A., *The multidimensional 0-1 knapsack problem: An overview.* European Journal of Operational Research, 2004. **155**(1): p. 1-21.

26. Darmann, A., et al., *The Subset Sum game.* European Journal of Operational Research, 2014. **233**(3): p. 539-549.

27. Oltean, M., C. Groşan, and M. Oltean, *Designing digital circuits for the knapsack problem,* in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2004. p. 1257-1264.

28. Oltean, M. and O. Muntean, *Solving the subset-sum problem with a light-based device.* Natural Computing, 2009. **8**(2): p. 321-331.

29. Wu, Q. and J.K. Hao, *A review on algorithms for maximum clique problems.* European Journal of Operational Research, 2015. **242**(3): p. 693-709.

30. Hwang, F.K. and D.S. Richards, *Steiner tree problems.* Networks, 1992. **22**(1): p. 55-89.

31. Liu, L., et al., *Physarum optimization: A biology-inspired algorithm for the steiner tree problem in networks.* IEEE Transactions on Computers, 2015. **64**(3): p. 819-832.

32. Nakagaki, T., et al., *Obtaining multiple separate food sources: Behavioural intelligence in the Physarum plasmodium.* Proceedings of the Royal Society B: Biological Sciences, 2004. **271**(1554): p. 2305-2310.

33. Bektas, T., *The multiple traveling salesman problem: An overview of formulations and solution procedures.* Omega, 2006. **34**(3): p. 209-219.

34. Matai, R., Singh, S., Mittal, M. L., *Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches,* in *Traveling Salesman Problem,* D. Davendra, Editor. 2010, IntechOpen.

35. Jones, J. and A. Adamatzky, *Computation of the travelling salesman problem by a shrinking blob.* Natural Computing, 2014. **13**(1): p. 1-16.

36. Dorigo, M. and L.M. Gambardella, *Ant colony system: A cooperative learning approach to the traveling salesman problem.* IEEE Transactions on Evolutionary Computation, 1997. **1**(1): p. 53-66.

37. Mitra, D., et al., *On-chip sample preparation for multiple targets using digital microfluidics.* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2014. **33**(8): p. 1131-1144.

38. Wu, K., et al., *An optical fiber network oracle for NP-complete problems.* Light: Science and Applications, 2014. **3**.

39. Funke, S., A. Nusser, and S. Storandt. *The simultaneous maze solving problem*. in *31st AAAI Conference on Artificial Intelligence, AAAI 2017*. 2017.

40. Held, M., C. Edwards, and D.V. Nicolau. *Examining the behaviour of fungal cells in microconfined mazelike structures*. 2008.

41. Nakagaki, T., H. Yamada, and Á. Tóth, *Maze-solving by an amoeboid organism.* Nature, 2000. **407**(6803): p. 470.

42. Ntinas, V., et al., *Oscillation-Based Slime Mould Electronic Circuit Model for Maze-Solving Computations.* IEEE Transactions on Circuits and Systems I: Regular Papers, 2017. **64**(6): p. 1552-1563.

43. Paolillo, A., et al., *Vision-based maze navigation for humanoid robots.* Autonomous Robots, 2017. **41**(2): p. 293-309.

44. Qin, J. and A.R. Wheeler, *Maze exploration and learning in C. elegans.* Lab on a Chip, 2007. **7**(2): p. 186-192.

45. Jellinger, K.A., *Comparative Cognition: Experimental Exploration of Animal Intelligence.* European Journal of Neurology, 2007. **14**(7): p. e34-e34.

46. Held, M., et al. *Dynamic behaviour of fungi in microfluidics - a comparative study*. 2009.

47. Held, M., C. Edwards, and D.V. Nicolau, *Probing the growth dynamics of Neurospora crassa with microfluidic structures.* Fungal Biology, 2011. **115**(6): p. 493-505.

48. Park, S., et al., *Motion to form a quorum.* Science, 2003. **301**(5630): p. 188.

49. Asenova, E., et al., *Optimal fungal space searching algorithms.* IEEE Transactions on Nanobioscience, 2016. **15**(7): p. 613-618.

50. Malik, S. and L. Zhang, *Boolean satisfiability from theoretical hardness to practical success.* Communications of the ACM, 2009. **52**(8): p. 76-82.

51. Boneh, A. and M. Hofri, *The coupon-collector problem revisited - A survey of engineering problems and computational methods.* Communications in Statistics. Part C: Stochastic Models, 1997. **13**(1): p. 39-66.

52. Fuerstman, M.J., et al., *Solving mazes using microfluidic networks.* Langmuir, 2003. **19**(11): p. 4714-4722.

53. Walther, A. and A.H.E. Müller, *Janus particles: Synthesis, self-assembly, physical properties, and applications.* Chemical Reviews, 2013. **113**(7): p. 5194-5261.

54. Howse, J.R., et al., *Self-Motile Colloidal Particles: From Directed Propulsion to Random Walk.* Physical Review Letters, 2007. **99**(4).

55. Garcia-Pichel, F., *Rapid bacterial swimming measured in swarming cells of Thiovulum majus.* Journal of Bacteriology, 1989. **171**(6): p. 3560-3563.

56. Herzog, B. and R. Wirth, *Swimming behavior of selected species of Archaea.* Applied and Environmental Microbiology, 2012. **78**(6): p. 1670-1674.

57. Vaituzis, Z. and R.N. Doetsch, *Motility tracks: technique for quantitative study of bacterial movement.* Applied microbiology, 1969. **17**(4): p. 584-588.

58. Schlegel, H.G., *Allgemeine mikrobiologie*. 1985, Stuttgart: Georg Thieme Verlag.

59. Weinstock, M.T., et al., *Vibrio natriegens as a fast-growing host for molecular biology.* Nature Methods, 2016. **13**(10): p. 849-851.

60. C Brennen, a. and H. Winet, *Fluid Mechanics of Propulsion by Cilia and Flagella.* Annual Review of Fluid Mechanics, 1977. **9**(1): p. 339-398.

61. Sleigh, M.A., Blake, J.R., *Methods of ciliary propulsion and their size limitations*, in *Scale effects in animal locomotion*, T.J. Pedley, Editor. 1977, Academic Press, Inc.: New York. p. 234-236.

62. Hahm, J.H., et al., *C. elegans maximum velocity correlates with healthspan and is maintained in worms with an insulin receptor mutation.* Nature Communications, 2015. **6**.

63. Nicolau, D.V., Jr., et al., *Reply to Einarsson: The computational power of parallel network exploration with many bioagents.* Proceedings of the National Academy of Sciences of the United States of America, 2016. **113**(23): p. E3188.

64. Potsaid, B., Y. Bellouard, and J.T. Wen, *Adaptive Scanning Optical Microscope (ASOM): A multidisciplinary optical microscope design for large field of view and high resolution imaging.* Optics Express, 2005. **13**(17): p. 6504-6518.

65. Einarsson, J., *New biological device not faster than regular computer.* Proceedings of the National Academy of Sciences of the United States of America, 2016. **113**(23): p. E3187.

66. Pisinger, D., *Linear Time Algorithms for Knapsack Problems with Bounded Weights.* Journal of Algorithms, 1999. **33**(1): p. 1-14.

67. McKee, S.A. *Reflections on the memory wall*. in *2004 Computing Frontiers Conference*. 2004.

68. Science, O., *Preliminary Conceptual Design for an Exascale Computing Initiative*, U.D.o. Energy, Editor. 2014.

69. King, J., Yarkoni, S., Raymond, J., Ozdan, I., King, A.D., Mohammadi Nevisi, M., Hilton, J.P., McGeoch, C.C., *Quantum Annealing amid Local Ruggedness and Global Frustration.* D-Wave Technical Report Series, 2017(14-1003A-C).

70. Yin, X., et al., *Computation in Chemistry: A Summary of the Development and Models of DNA Computing.* Progress in Chemistry, 2017. **29**(11): p. 1297-1315.

71. Landauer, R., *Irreversibility and heat generation in the computing process.* IBM Journal of Research and Development, 2000. **44**(1): p. 261-269.

72. Strohmaier, E., Dongarra, J., Simon, H., Meuer, M., *Top 500 Supercomputers.* 2017.

73. Mitchell, J.G., *The energetics and scaling of search strategies in bacteria.* American Naturalist, 2002. **160**(6): p. 727-740.

74. Mitchell, J.G. and K. Kogure, *Bacterial motility: Links to the environment and a driving force for microbial physics.* FEMS Microbiology Ecology, 2006. **55**(1): p. 3-16.

75. Lard, M., ten Siethoff, L., Generosi, J., Månsson, A., Linke, H., *Molecular motor transport through hollow nanowires.* Nano Lett., 2013. **14**: p. 3041-3046.

76. Balaz, M. and A. Månsson, *Detection of small differences in actomyosin function using actin labeled with different phalloidin conjugates.* Anal. Biochem., 2005. **338**(2): p. 224-236.

77. Lard, M., ten Siethoff, L., Månsson, A., Linke, H., *Tracking actomyosin at fluorescence check points.* Sci. Rep., 2013. **3**: p. 1092.

78. Ooms, M.D., et al., *Photon management for augmented photosynthesis.* Nature Communications, 2016. **7**.

Chapter 3

Scaling Network Computing

Given the need to evaluate the scaling of network computing with biological agents, the computing times for solving SSP networks of increasing sizes were benchmarked against those of classical and quantum computers solving the same SSP instances. The program execution times of various electronic chips were plotted against experimental data (with cardinality 3 to 7) extended by stochastic simulations using both *E. coli* and *V. natriegens* exploring virtual SSP networks of larger sizes (Figure 13). Because of the long computing times required to solve large SSP networks, only dividing bacteria were considered for scaling analysis. The QC counterparts were obtained according to Grover's algorithm [29], which specifies a square root improvement over exponential time of classical computers. In-lab experimental proof of agent multiplication in the network made it possible to project via simulations the advantage of a multiplication run mode over a combinatorial run mode, where multiplication conditions in the network are not achieved. The simulation written in Scala (running on the JVM on a Lenovo laptop with Microsoft Windows 10 OS and with Intel(R) Processor Core i5-7200U CPU @ 2.50GHz (2 Cores, 4 Logical Processors)) used experimental data and parameters i.e., velocity, division rate, directionality statistics in the junctions, and their statistical spread, to (i) reproduce the (experimental) behaviour of the physical device solving the SSP with cardinality 3 to 7 and to (ii) scale its theoretical performance up to cardinality 16 (which has a network size of 328 exits). Given the inherently sequential nature of computers running simulations with exponentially increasing number of dividing bacteria, scaling via simulations could not progress beyond cardinality 16. From there, data were extrapolated via regression analysis up to cardinality 100 with a high degree of accuracy ($R^2 > 0.99$). The simulations were run using *E. coli* moving at an average speed of 4 um/s and stochastically multiplying with a period of 30 minutes at a multiplication rate of 40%, and *V. natriegens* moving at an average speed of 8 um/s and multiplying every 20 minutes at a multiplication rate of 35%. The pass junction error was set to 0.1% according to experimental results. The condition for halting the computation was set by Euler's formula (coupon collectors problem) [53] comprising a statistical minimum number of agents required to run the network in order to make sure that, with >95% probability, all 2^N combinations are consistently explored,

given the problem input size N (cardinality). Double (2x) the size set by this condition was adopted to obtain a reliable result, discernible with human eyes, so as to clearly mark the distinction between erroneous paths and correct ones. Similarly, experiments were run with the minimum number of agents required for a human discernible solution.

Separately, a recursive SSP "brute force" algorithm was developed in C, a low-level language with fast operational capabilities, which sequentially evaluates 2^N combinations of a given SSP instance with input size N. To assess the computing time of electronic computers, the algorithm was run on various generations of Intel chips (Intel 382DX, Intel486 DX, Intel DX2, Intel Pentium and Intel Core Quad 2 Processor and a Macbook 2011). The "brute force" approach was chosen to guarantee execution of the equivalent number of operations carried out by agents on the SSP device (2^N). In addition, various generations of chips were used in order to maintain a direct relation between the history of electronic chips and the infancy of the novel device. To overcome the challenge of having physical Intel chips available, initial results were compiled by running a virtual machine with Virtual Box and ISO images. As virtualization mimics the software of the intended target but not the hardware, emulation was introduced to gain a deeper understanding of the capacity of computing resources of old Intel Chips. PCem emulator [54], an IBM certified product, was employed to reproduce both hardware and software and test current results. To analyse results obtained with precision and spot potential unexpected behaviour, Valgrind [55], a tool for profiling and memory management detection, was used to break down and analyse RAM memory segments and usage over program execution with increasing cardinality. The program execution times of desired CPU and RAM configurations as a function of 2^N operations were obtained for any given SSP size up to cardinality 30. Knowledge of the operational parameters of the respective emulated computer chips allowed for an advanced regression analysis, which revealed that the computing time follows an exponential relationship with respect to SSP cardinality and Million Instructions Per Second (MIPS, a key performance parameter of computer chips, reported at peak performance by the respective developer). The highly accurate ($R^2 > 0.999$) correlation between computing time as output, SSP cardinality as input, and MIPS as parameter, allowed the estimation of the computing time needed to solve SSP for other high-performance electronic chips, e.g., AMD Ryzen Threadripper 3990X. Data were extrapolated up to cardinality 100 via regression analysis. Importantly, this correlation allowed the extrapolation of the performance of classical electronic computers if they would work in 'quantum computing mode', that is, if their performance would

follow Grover's theorem with computing time being proportional with $2^{N/2}$, where N represents SSP input size (or cardinality), as opposed to classical electronic computers, whose computing time is proportional with 2^N. For the sake of drawing a comparison between different computing models, errors in QC due to decoherence and other scaling challenges, i.e. simultaneous control over large number of qubits [14, 56], were assumed not be obstacles for QC scaling laws.

The program execution times (in hours) of electronic, (hypothetical) quantum and network computers running SSP, the latter using *E. coli* and *V. natriegens* in a multiplication rum mode, were plotted versus cardinality on a logarithmic scale in Figure 13. If the fastest electronic computer (AMD's Rayzen Threadripper) and a network computer run with *V. natriegens* would both require more than 5 weeks to solve an SSP of cardinality ~45, an intersection between a quantum and network computer would only happen towards cardinality 100, computed in approximately 2 years and 3 months (at cardinality 100, the fastest classical electronic computer will require ~10^{13} centuries!)

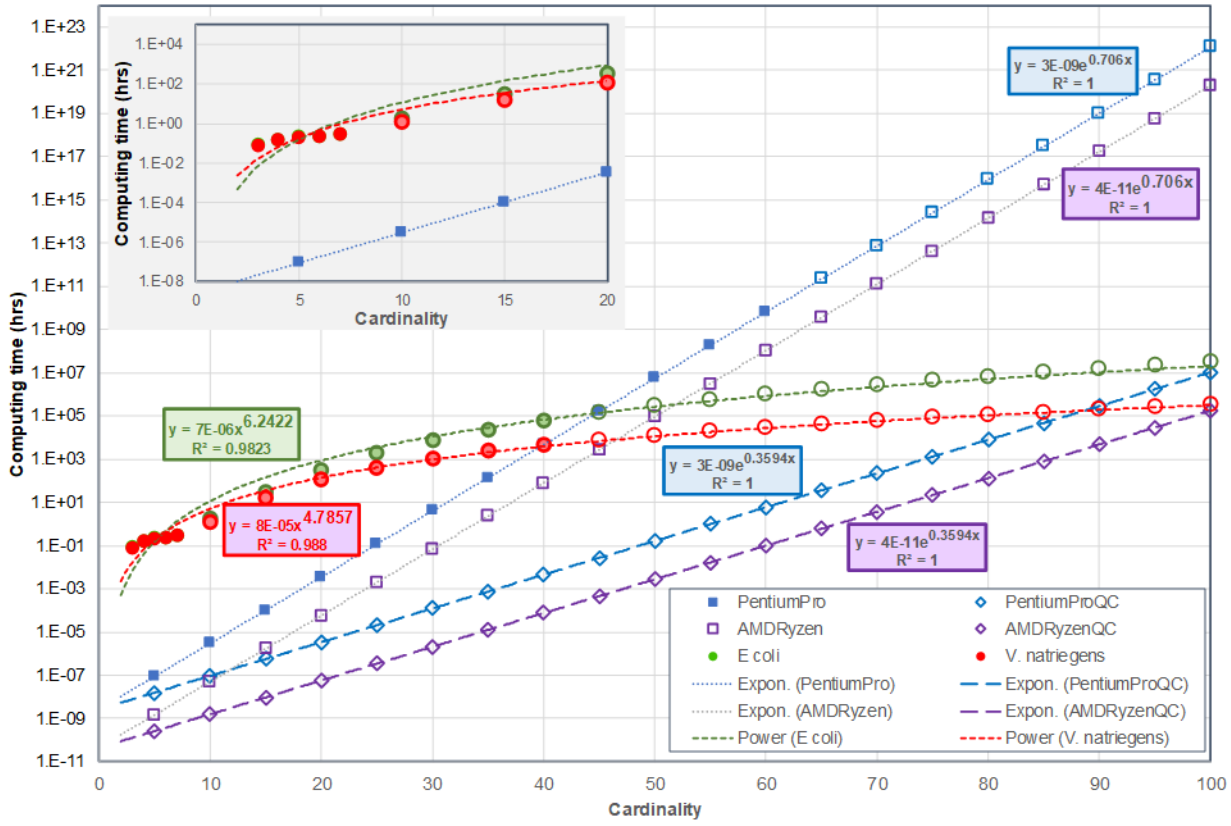


Figure 13: Comparison of electronic, quantum and network computing models. Scaling analysis of the computation time for bacteria-operated network computing (circle markers, e.g., ○), and

electronic computers, both classical (square signs, e.g., □), and operating in ‘quantum mode’ (diamond markers, e.g., ◊). The experimental data are indicated by filled markers (●, and ■, for bacterial and classical electronic computers, respectively). The scaled data based on regressions (equations and R^2 indicated close to the respective trends) are indicated by empty markers (○, □, and ◊, for bacterial, classical, and quantum computers), and the stochastic simulated data are indicated by light-filled markers (●). The experimental data for *E. coli* operated computers are obscured by those obtained for *V. natriegens* operated computers.

Chapter 4

A Network Computing -Based True Random Number Generator

As an alternative application to solving the SSP, two network computing devices (with 45 and 190 exits respectively) were used to generate high entropy data for a novel True Random Number Generator (TRNG) by exploiting the stochastic motions of *E. coli* HCB437 bacteria in the networks. The following contribution, currently under final revision for publication, discusses (i) the role of a network computing device as an entropy-providing source, (ii) its integration with other software-based components and (iii) a full proposal of a TRNG, including internals, operation and performance.

A True Random Number Generator using Bacterial Motility in Microfluidic Networks

Giulia Ippoliti[1], Ayyappasamy Sudalaiyadum Perumal[1], Falco C.M.J.M. van Delft[2], Dan V. Nicolau Jr.[2,3], Monalisha Nayak[1], Dan V. Nicolau[1*]

[1] *McGill University, Faculty of Engineering, Department of Bioengineering, Montreal, Quebec, H3A 0C3, Canada,*

[2] *Molecular Sense Ltd, Liverpool L36 8HT, UK,*

[3] *School of Mathematical Sciences, Queensland University of Technology, Brisbane, QLD 4000, Australia*

+These authors contributed equally.

**corresponding author, email: dan.nicolau@mcgill.ca*

Abstract

In a world envisioning global Internet service, smart cities and super-computing power, the need of cryptographic systems granting secure exchange of sensitive data is urgent. Random numbers are essential to cryptography as they provide the strongest method of data encryption, no matter how powerful a computer an adversary has. Given the non-computable nature of randomness, producing truly random data is a major challenge. This work demonstrates a hybrid software and hardware true random number generator which for the first time uses the stochastic direction of motion of bacteria autonomously swimming in a microfluidic network as a non-deterministic data source. The true random number generator offers low-cost and accessible biological resources, software compatibility, competitive high throughput and low energy consumption which makes it suitable for energy-expensive applications such as blockchain technologies. Importantly, data generated pass all 15 NIST Special Publication 800-22 randomness tests.

4.1. Introduction

While the secure cryptography-based transmission of information is of centuries-, if not millennia-old [1], the present highly interconnected world greatly increases its importance and criticality. Today, given the ever-increasing computing power, the success of technological advances, e.g. online banking [2], mobile communication [3, 4], Internet of Things [5-7] and smart cities [8], highly depends on trusted cryptographic systems granting secure exchange of sensitive data.

Cryptographic protocols rely on the generation and use of secret keys that must be unpredictable to potential attackers [3]. These secret keys are used by encryption and decryption to respectively transform plaintext into unreadable format, and vice versa, to decode and convert unreadable text to readable information.

The most secure cryptographic algorithms in use today, such as Advanced Encryption Standard (AES) [9], the Rivest-Shamir-Adleman (RSA) [10] and Blowfish [11], use random numbers to build reliable encryption and decryption keys. To be truly random, a secret key must present the following properties: (i) each number must be statistically independent from the previous therefore, given each number in the sequence, a particular value is not more likely to follow than all other possible ones; (ii) given a sequence, the numbers must follow a uniform distribution and must have the same frequency of occurrence; and finally (iii) the sequence values must be unpredictable, both in terms of backtracking and predicting future values [12]. The lack of frequency differential among values of the keys and consequently, the lack of potential clues, makes it impossible for an attacker to break the encryption: this is the strongest possible method of encryption, known as the *One-Time Pad* [13-15].

Producing truly random sequences for computational purposes is a major technical challenge. Because neither human minds, nor Turing deterministic machines, are able to generate true randomness [16], one must rely on the chaotic nature of physical processes as an entropy source. Given the non-computable nature of randomness [16], presently the most used random number generators (RNGs) are *pseudo*-random number generators (PRNGs), which use a deterministic mathematical formula to generate 'random looking' sequences of numbers completely determined by an initial state called a *seed*. However, given the same seed as a starting condition, a PRNG will always generate the same sequence [12]. Regardless their CPU-available environment and

low cost, none of the PRNGs is truly random and their use in cryptography represents a major security threat [17]. In contrast, *true* random numbers generators (TRNGs) are non-deterministic systems that work by measuring intrinsically random physical processes, such as thermal noise [18], photon arrival times [19, 20] and radioactive decay [21, 22]. The lack of compatibility between physical and human-made electronic systems makes TRNGs more expensive and less available [21].

In recent years, few examples of biological TRNG, relying on biochemical and biological systems to generate randomness, have been proposed as a potential alternative to common electronic-based TRNG. In fact, the stochastic nature of many biological processes makes them unpredictable by existing algorithms and consequently, a promising avenue for novel research. One stream proposes DNA computation techniques to construct DNA-based RNGs [23, 24], which use oligonucleotide synthesis to generate random sequences. However, current limitations of DNA technologies [25] including the considerable human intervention, the unreasonably large amounts of DNA and the time constraints required to process and read oligonucleotide sequences, prevent such methodology from scaling and providing large amount of data in a fast and automated fashion. Other examples include the use of bioelectrical (electromyography, electroencephalogram) or physical signals (blood volume pulse, galvanic skin response) [26] which all have the drawbacks of expensive set-up, discontinuous data availability, and low-quality statistical properties of generated bits. On the other hand, random digits fluctuations obtained from biometric readings [27] demonstrated adequate statistical properties for RNG, although finding appropriate biological phenomena with easy accessibility, fast sampling rate, high accuracy of measurement and variability of sampling rate is currently a challenge.

The flagella-driven motility attribute of bacteria is a phenotypic property that is difficult to predict, despite being an extensively researched problem. The source of randomness in bacterial motility is multifactorial, namely turn angle preferences, velocity deviations, interactions with the surrounding boundary layers, chemo signaling, quorum sensing, and crowdedness [28]. In this contribution, the inherent stochastic behavior of *E. coli* HCB437, having 50%-50% turn direction preference when confined in a microfluidic network and presented with perfectly symmetrical bifurcating junction, is used, for the first time, as an entropy-providing process of a TRNG.

In the following sections, a proof-of-concept of a bacterial-run TRNG is presented and demonstrated using fluorescently-labelled *E. coli* bacteria. A full working model is also discussed which places the TRNG to function as an entropy source for a PRNG.

4.2. Concept and Design

This contribution proposes and demonstrates a proof of concept of a TRNG hardware and software hybrid device which uses the stochastic direction of motion of bacteria autonomously swimming in a microfluidic network, as a non-deterministic, entropy-providing process.

The TRNG entropy source and its components, consisting in a stochastic data source, a digitization component, a conditioning component, and a 'health testing' component, are presented in Figure 14. The stochastic data source (hardware component) comprises of bacteria physically moving across microfluidic channels, designed and microfabricated on the microchip. Trajectories of their tracks are recorded, translated to images, and sent as inputs to the digitization component (software component). The digitizer is an object recognition and processing algorithm designed to recognise and translate the bacterial directions of motion to bits and release the raw digital output. A conditioning algorithm is used to remove eventual bias from the raw data and finally release the true random number sequences. Finally, health tests validate the behaviour of the entropy source.

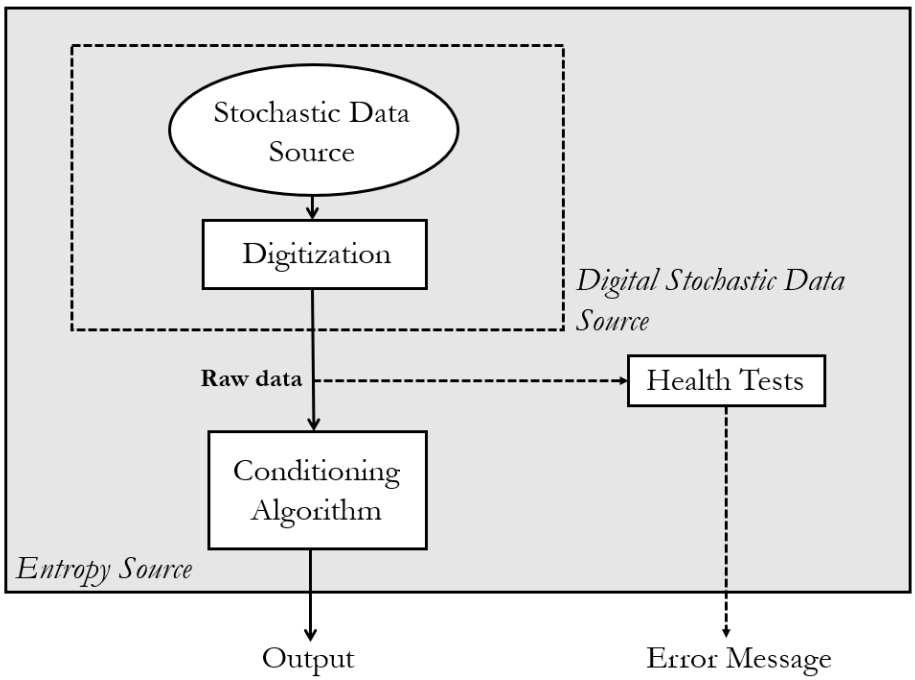


Figure 14: TRNG components layout. The digital stochastic data source comprises of a hardware component producing stochastic data other than binaries and a digitization software component responsible for translating data in binary format. The output, referred to as raw data, is then sent to a conditioning algorithm – a deterministic and cryptography-safe function -which works by

reducing bias present in raw data. Health tests check the validity of the entropy source behaviour. Truly random bits are released as output. Adapted from NIST Special Publication 900-80B [29] (Copyright 2018).

4.2.1. The Stochastic Data Source

In order to exploit the stochastic movement of bacteria in the most efficient and robust way, such that it can be directly employed for a TRNG in a digital computer, hardware units, i.e. network junctions, are needed that yield a clear binary output (i.e. '0' or '1') with 50% probability each. Moreover, to speed up the translation of bacterial movements to bits by parallel processing, an as high as possible density of such units is desired without compromising on the quality and independence of these stochastic events.

In fact, such units have already been employed in bio-computation networks [30]; the so-called Split Junctions (SJ's) are designed to allow agents to proceed in either the original direction of motion with a 50% probability, or to change lane to an alternative direction of motion with a 50% probability. Here we apply this concept to random number generation (instead of calculation) by stacking SJ's only; the equivalent calculation network would basically be a Subset Sum Problem network with unit steps only, i.e. it would generate Pascal's triangle with a 50%-50% binomial distribution [31].

Note, that the occurrence of a 50%-50% split depends on both the junction design and the inherent bacterial agent characteristics. The best possible junction design for a 50%-50% split is obtained by using a local mirror symmetry at the split (Y-split). Next, the bacterial species employed should not show any natural preference for turning left or right. This is indeed the case for *E. coli* HCB-437 [32, 33]. Hence, this is the agent that was used in this study.

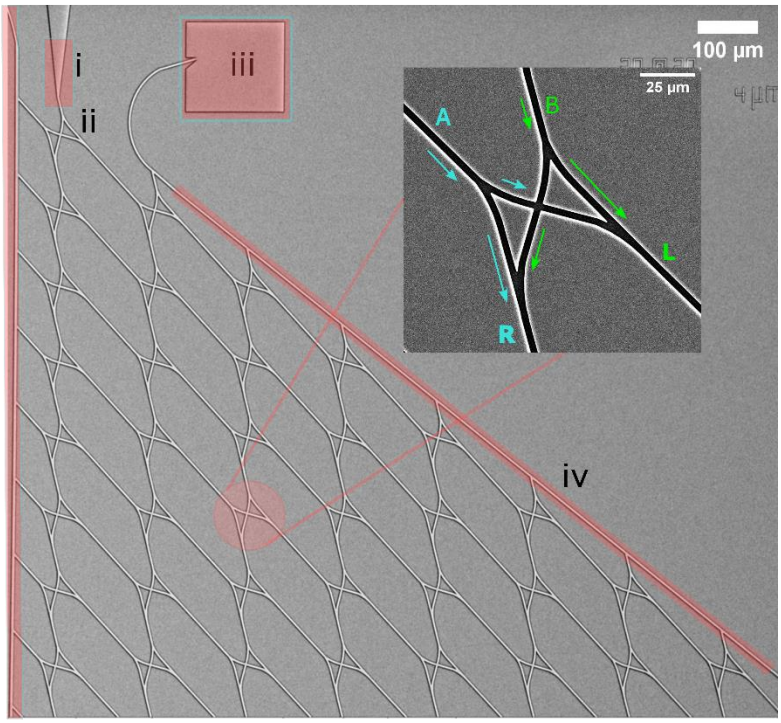
4.2.2. Conditions for Unpredictability in a Stochastic Data Source

4.2.2.1. Perfectly symmetrical junctions.

An optical image of microfluidic network used for this demonstration is illustrated in Figure 15. Two network sizes - comprising of 45 and 190 junctions respectively - were used to generate raw data. The choices of network designs and dimensions were based on the size of the biological

agents (length – width: 2.5 ± 0.6 - 1.0 ± 0.2 µm), which required enough space to move smoothly, but at the same time, enough constraint to avoid U turns [31]. Accordingly, the channel width was set to 4 µm. Agents enter the RNG network from a unique entry (at the top in Figure 15) and move towards the exits (bottom in Figure 15). When crossing a SJ (enforcing a binary path), agents can turn either left or the right with equal probability, as evidenced by experimental data. Figure 15 illustrates details of a SJ with two inlets and their respective binary pathways (resolving to L – left and R-right) marked with blue and green arrows. To enable equal turning probability, junctions were designed to be perfectly symmetrical, as shown in Figure 15. The pattern of repeating symmetrical junctions guarantees that a fair probability condition is maintained throughout the whole network paths, such that each path is equally probable to be crossed, or in other words, such that the maximum entropy is preserved: this is a *sine-qua-non* condition for the non-deterministic behaviour of the system.

I



II

S.No.	Junction route (A_R+B_R)	Junction route (A_L+B_L)
agent count (n)	571	569
directionality preference (%)	50.10	49.90

Figure 15: Microfluidic network with junctions, and 4 µm-wide channels. I. A part of the split junction only network, showing with repetitive split junctions (SJ) (ii) and several other network

components like entry funnel (i), ghost or dirt lanes (iv), traps for forbidden bacterial traffic(iii). All the component marked in transparent red shapes. II. Directionality preference of *E. coli HCB 437*, showing a 50:50 preference for turning left or right. The inset image zooms into a single junction. The junction has two symmetrical inlets labelled A and B, and two outlets labelled R for 'right' – which maps to 0 - and L for 'left' – which maps to 1. Biological agents move through the junction resolving to a 'left' or 'right' direction. II. The directionality preference of E. coli HCB 437 obtained from experimental data is shown for paths $A_R + B_R$ and $A_L + B_L$ (as %).

4.2.2.2. Bacterial stochastic behaviour

Previous work in microfluidics suggested that bacteria offer a broad spectrum of benefits with respect to computing applications, compared to other alternatives [31]. Among bacterial species, *E. coli HCB437* was found to be the most suited candidate to provide stochastic behaviour in the network [33]. *E. coli HCB437* presents smooth swimming phenotype with reduced tumble frequencies and longer run lengths [34]. It presents velocities approaching 20 μm/s [35] using multiple flagella oriented axially along the cell body [34]. Importantly, *E. coli HCB437* present an equal preference (50:50) for turning angles at SJ [35]. Such stochastic behaviour is an essential condition to generate maximum-entropy data.

4.3. Operation

4.3.1. The Entropy Source

The digital stochastic data source. Figure 16 illustrates the architecture of the digital stochastic data source comprising a hardware component (microchip with running bacteria and microscope), and a digitization software for image processing and digital raw data extraction. The microfluidics hardware comprises of a PDMS-on-silicon chip coated with agar gel to facilitate the movement of self-propelled bacteria. The left panel in Figure 16 illustrates the stochastic bacterial movements in the network being captured by a 10X objective microscope taking snapshots with a 0.5 s resolution (data generation). Produced snapshots record the full history of bacterial flow and therefore they encode the necessary information to be digitized. The right panel in Figure 16 illustrates the operating components of the digitization software taking snapshots as input, processing each frame and extracting binary data (data interpretation and digitization). Digitized raw data are then released as output. The maximum output obtained over a single experimental run was 6000 frames using a network of 45 junctions.

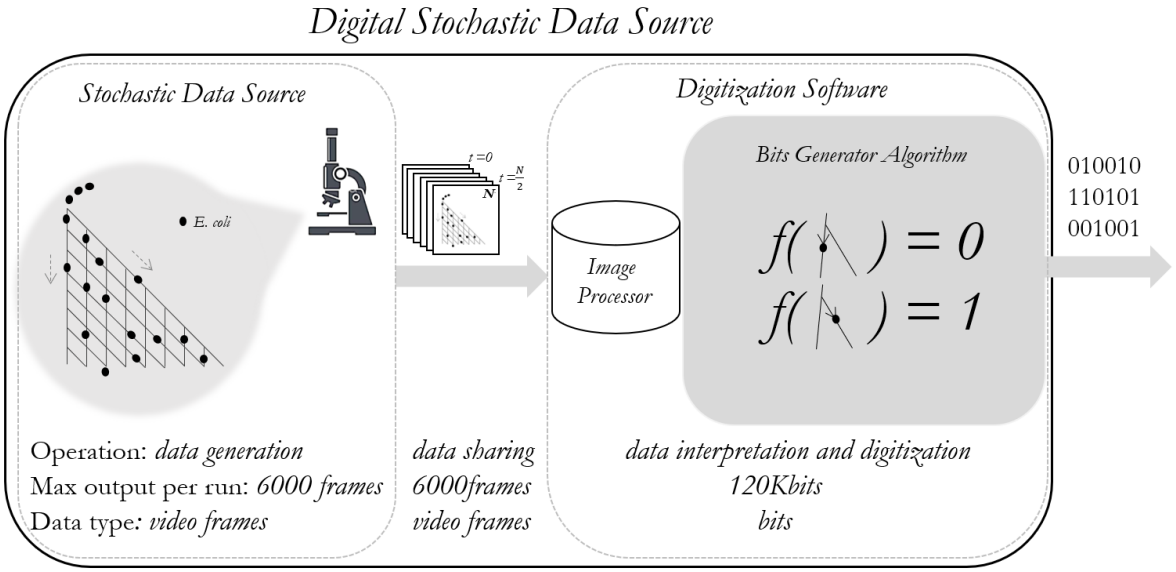


Figure 16: Digital stochastic data source. The steps involved within the digital data source. Shown on the left, the hardware component consists in a microfluidic network with moving bacteria. The hardware is responsible for raw data generation. The network snapshots are then sent to the

digitization software (data sharing) and bits, mapped to the stochastic motion of bacteria, are extracted by the Bits Generator Algorithm (data interpretation and digitization).

Bits generator algorithm: The Bits Generator Algorithm produces stochastic data in a digitized format by processing the incoming snapshots. The core object recognition algorithm was designed to compare two subsequent snapshots, find the displacements of the agents between snapshots (difference between positions), record these displacements corresponding to 'decision' events inside junctions and assign 0 when agents turn 'right' or and a 1 when they turn 'left'. The grey panel in Figure 17 shows the displacement of agents over a time step Δt by comparing each pair of snapshots from the light-green panel. Bacterial positions in the first frame (earlier in time) are marked in light yellow, and their subsequent appearances in the next frame in darker yellow. The red arrow indicates the displacement.

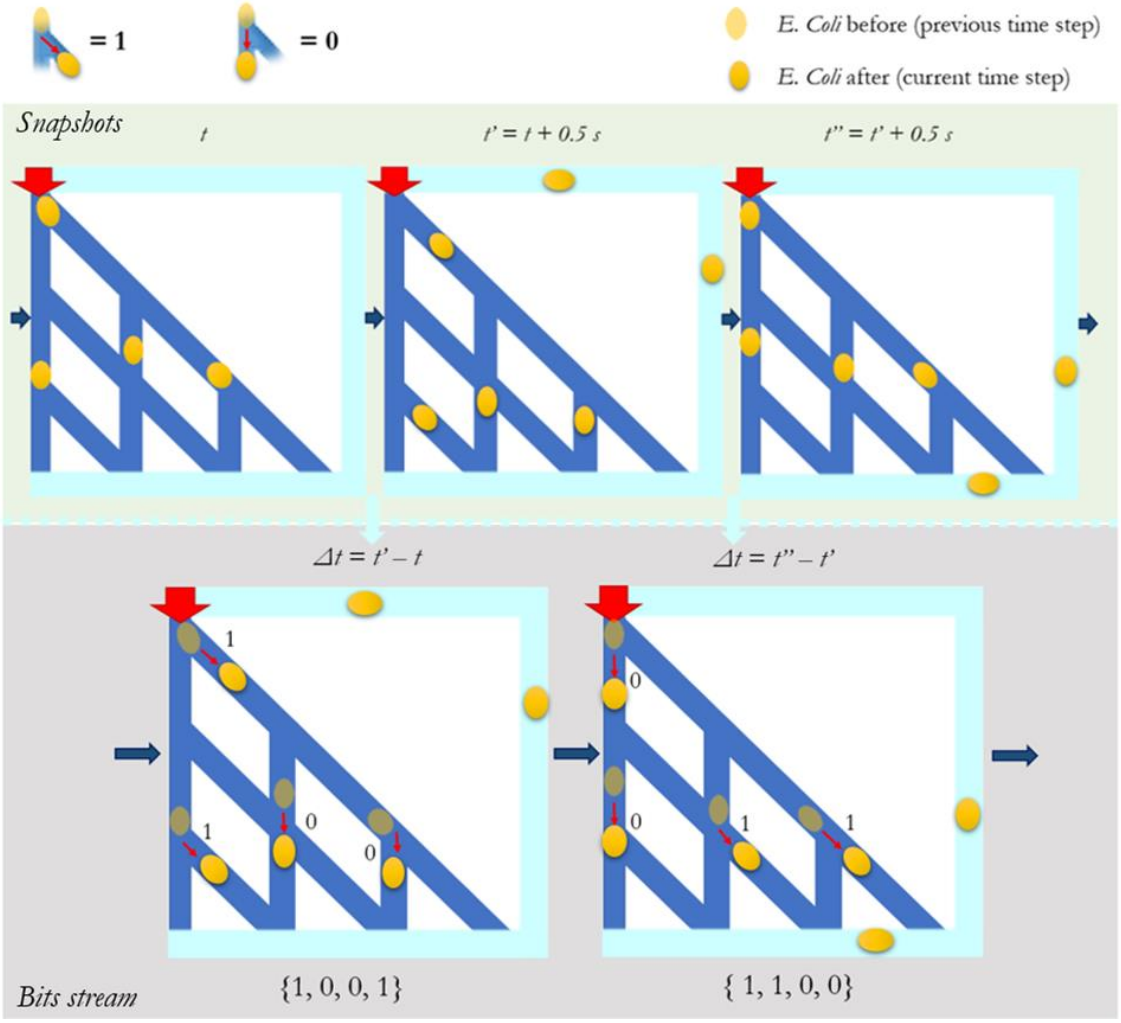


Figure 17: Raw data transformation process. Bits Generator Algorithm digitizes the bacterial displacement data comparing pairs of snapshots at two subsequent time steps. The light-green panel shows how snapshots look once produced every 0.5 s. The grey panel presents the comparison operation performed by the generator to extract bits. For each time step Δt, each bacterium appearance in an earlier frame (light yellow) is comparted to a later frame (dark yellow) as shown by t' – t and t'' – t'. The displacement of the agent (red arrow), resulting in a turn event at a junction, is mapped to a bit.

4.3.2. Performance and Statistics

4.3.2.1. Raw Data

The 45 junctions network provided 6000 frames per single experiment, which coded, on average, for 120K bits. On the other hand, the 190 junctions network provided 2500-3000 frames per experiment, thus coding for 60-80K bits. It was found that the total bits produced is dependent on the density of the agents present in the network, which can be modulated, by e.g., growth conditions, temperature, to achieve an optimum bacterial density (if the network is overpopulated, clogging may occur).

Using both networks, it was possible to generate 1M bits of digital raw data from 10 different experiments taking about 4.5 hours of total experimental time (experiments were carried out in series, not in parallel).

The minimum entropy [36] and Shannon Entropy [37] were calculated for the proposed RNG system, followed by the validation of the entropy source statistics. The minimum entropy provides a measure of how easy it is for an attacker to guess the most likely output of the entropy source [12]. For given a set $A = \{x_1, x_1, \dots x_k\}$ with probability $\Pr(X = x_i) = p_i$ for $i = 1, \dots, k$, it is defined as [29]:

$$H = \min_{1 \le i \le k} (-\log_2 p_i)$$

The Shannon Entropy measures the average information in the data resulting from the frequency distribution of the symbols that comprise the data [12]. It is defined as follows:

$$H = -\sum_{i=1}^{k} p_i \log_2 p_i$$

Table III presents raw data statistics (evaluated over a binary file).

Table III: Statistics of raw data obtained from the digital stochastic data source.

Digitized raw data (1M bits)	
Shannon Entropy (bits per byte)	7.997
Minimum Entropy (bits per byte)	7.791
Serial Correlation (0.0 totally uncorrelated)	0.01
Throughput (bits per second)	256 *

**Seed size – see section 4.4*

4.3.2.2. Conditioned data

Raw data were processed by the AES based Cipher Block Chaining-Message Authentication Code (AES_CBC_MAC) [38] conditioning algorithm, the last entropy source component (Figure 14). Table IV presents the final results of the TRNG.

Table IV: Statistics of processed data obtained from the entropy source.

TRNG output (1M bits)	
Shannon Entropy (bits per byte)	7.998
Minimum Entropy (bits per byte)	7.833
Serial Correlation (0.0 totally uncorrelated)	0.01
Throughput (bits per second)	256 *

**Seed size – see section 4.4*

4.3.2.3. NIST randomness tests of entropy source

The NIST randomness tests were used to examine the stream of TRNG bits and compute statistical confidence of their randomness against a theoretical 100% entropic stream of bits [12]. Results are presented in Table V.

Table V: NIST tests results for 1M bits produced by the entropy source

Test	P-value	Result
monobit_test	0.18242	PASS
frequency_within_block_test	0.04348	PASS
runs_test	0.93023	PASS
longest_run_ones_in_a_block_test	0.43447	PASS
binary_matrix_rank_test	0.58931	PASS
dft_test	0.07317	PASS
non_overlapping_template_matching_test	0.99999	PASS
overlapping_template_matching_test	0.57004	PASS
maurers_universal_test	0.68542	PASS
linear_complexity_test	0.30754	PASS
serial_test	0.09000	PASS
approximate_entropy_test	0.09046	PASS
cumulative_sums_test	0.15215	PASS
random_excursion_test	0.02886	PASS
random_excursion_variant_test	0.25082	PASS

4.4. The Full Model

4.4.1. CTR PRNG Integration

The entropy source throughput (Table IV) positions the TRNG to work best in conjunction with a cryptography secure (CS) PRNG, by providing it with random input blocks called seeds. Figure 18 proposes an integrated architecture where the entropy source periodically provides seeds (reseeding) to the NIST approved Counter Mode Deterministic Random Bytes Generator (CTR DRBG) [39, 40]. The initial seed instantiates the DRBG and determines its initial internal state used to generate the first set of output bits. Reseeding allows restoring the secrecy of the DRBG output in case a seed, or its internal state, becomes known. By enabling periodic reseeding, various threats can be addressed potentially concerning the DRBG seed, entropy input or working state being compromised over time [39].

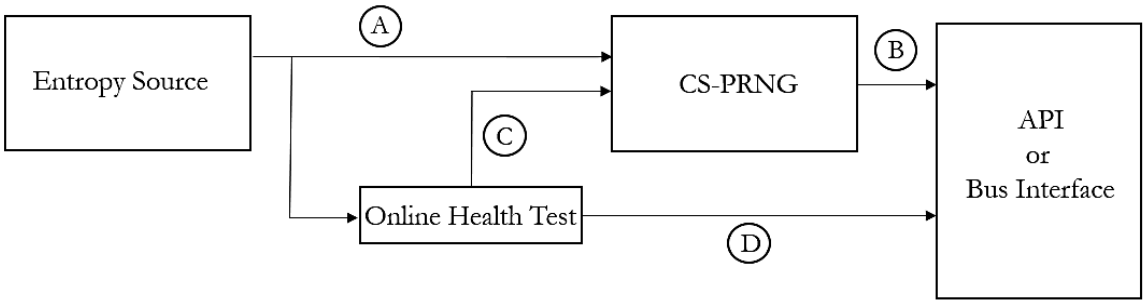


Figure 18: TRNG and CTR DRBG integration. A) The TRNG functions as an entropy source to the CTR DRBG by periodically providing a seed. B) Random numbers are made available to a custom API. C) Online Health Tests monitor the entropy source behaviour and validate each seed. D) The Online Health Test provides up-to-date feedback on the system's status.

An implementation of the CTR DRBG with AES block cipher [9] was adopted due to its security strength advantage compared to other block cipher algorithms [41]. By functioning as a randomness source, the TRNG provides CTR DRGB with random input blocks of 256 bits, (*seedlen = blocklen + keylen* in Table VI). The secrecy of the input seed provides the basis for the security of the DRBG [39]. Ideally, the entropy provided in each seed should be close to ideal, hence a seed length *seedlen* should provide *seedlen* bits of entropy. However, the actual seed length

(in bits) may be increased within some limits as far as the total entropy meets the minimum requirements of the final application [39].

Table VI lists the NIST approved parameters adopted in this proposal using AES-128. The violation to these limits would make the CTR DRGB vulnerable to attackers [39].

Table VI: Parameters provided by the NIST [39] on required bit lengths

Parameters	**AES-128**
Input and Output Block Length *(blocklen*)	128 bits
Security Strength (*security_strength*)	128 bits*[1]
Key Length (*keylen*)	128 bits
Seed Length (*seedlen = blocklen + keylen*)	256 bits
Entropy Input Length	*seedlen*
Max Number of Bits per Request	2^{11}bits*[2]
Maximum Requests Between Reseeds (*reseed_interval*)	2^{48}requests

*[1] [41]

*[2] The security of CTR_DRBG can be improved by limiting the number of requests and bits provided per requests [39]. This value was derived considering the minimum counter field length, *ctr_len* = 4 where the Max Number of Bits per Request, $B = (2^{ctr\_len} - 4) \times blocklen \approx 2^{11}$

The CTR DRBG implementation was run with 7.998 bits/byte of entropy input seeds generated by the entropy source. Given such full entropy condition, the parameter *nonce* – a time-varying value, which has at most a negligible chance of repeating [39], was not required in the DRGB instantiation process, and the use of a derivation function was not needed. A derivation function provides additional internal code to derive the DRBG internal state value or to distribute entropy evenly across the length of a low-entropy bit string [39]. Using a derivation function is optional during instantiation and reseeding and required only if the input *seedlen* does not provide full entropy. In addition, the optional parameter *personalization string* - a string of bits that is combined with the secret entropy input and (possibly) the nonce to produce a seed [39] - was not specified.

4.4.2. NIST randomness tests

A 2.2MB of data (17.6M bits) were generated with 24 reseeding cycles and 256 bits constant *seedlen*. The Shannon entropy of the PRNG output was 7.998 bits/byte. The NIST tests results are presented in the Table VII.

Table VII: NIST test suite results for 2.2M bits produced with CTR DRGB

Test	P-value	Result
monobit_test	0.87785	PASS
frequency_within_block_test	0.40268	PASS
runs_test	0.01880	PASS
longest_run_ones_in_a_block_test	0.07506	PASS
binary_matrix_rank_test	0.59679	PASS
dft_test	0.41124	PASS
non_overlapping_template_matching_test	1.00570	PASS
overlapping_template_matching_test	0.35517	PASS
maurers_universal_test	0.06825	PASS
linear_complexity_test	0.48039	PASS
serial_test	0.03121	PASS
approximate_entropy_test	0.03130	PASS
cumulative_sums_test	0.90362	PASS
random_excursion_test	0.02460	PASS
random_excursion_variant_test	0.01801	PASS

Finally, Figure 19 summarizes the proposed model by displaying fundamental data transformations throughout the RNG chain.

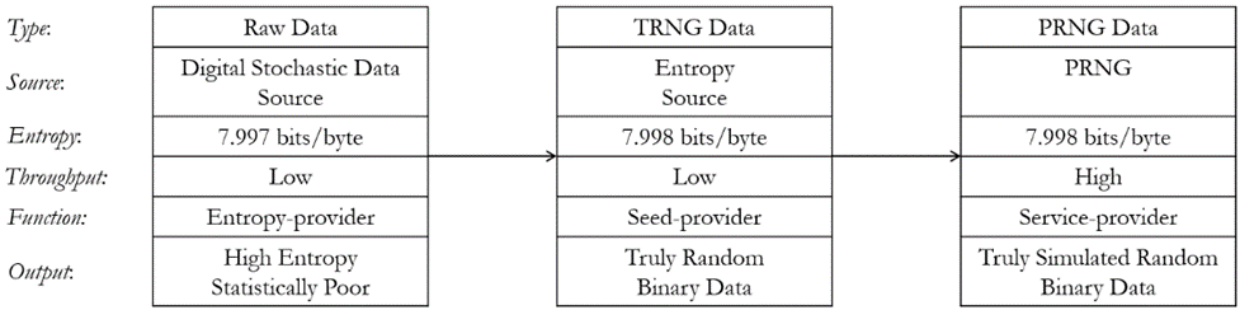


Figure 19: Data transformation model. The raw data generated by the digital stochastic data source exhibits high entropy, but poor statistical properties. The conditioning algorithm adjusts these

statistical properties and returns truly random data as the final TRNG output. At this stage, the cryptography secure PRNG receives true random data as periodic input seeds (256 bits) and returns a high throughput output enabling 2^{48} requests per seed with 2^{11} bits per request (Table VI). By providing a truly random seed, the TRNG-PRNG model behaves as a TRNG.

4.5. Perspectives

This work harnesses the stochastic motion of bacteria in constrained microfluidic channels to generate random numbers. It demonstrates a model which involves generating of raw data by the physical source, subsequently conditioning data with a CS conditioning algorithm and feeding the conditioned output (as a periodic seed) to a CS PRNG. Ultimately, the adoption of the proposed model for cryptographic applications depends on the validity of the entropy source to produce high entropy data with high reliability. Accordingly, statistics, e.g. min-entropy, and randomness tests were run to validate the binary output produced by the entropy source.

In order to develop a bacterial-run entropy source, observing the same frequency of *right* and *left* turns at split junctions was essential and possible thanks to the unbiased motility of the *E. coli HCB437* strain and the symmetrically designed and fabricated junctions. Experimental data demonstrated a slight *right* turn preference (50.10 versus 49.90%), possibly due to the chip fabrication (lithography and/or etching) errors, which resulted in a corresponding slight prevalence of 0s (coded from *right* turns). The CBC_MAC conditioning algorithm was purposely included in the design to reduce eventual bias.

Previous approaches [23, 24, 26] using biological processes for TRNGs require considerable human intervention to extract raw data. For example, electroencephalogram signal which proved suitable to produce random numbers with adequate postprocessing, requires presence of humans to evaluate electrical activity in living brains [26]. Similarly, using DNA technology for RNG purposes requires a long juxtaposition of human-performed laboratory steps to achieve the target objectives [25]. Far from being automated and reliable as efficient data-providing solutions, such methodologies showed limitations in the context of modern technologies that must provide fast and ready-available competitive amount of data in an automated fashion. The proposed TRNG currently requires minimum human manipulation (adding and removing bacterial cells to the chip) and ideally, some level of laboratory supervision; all remaining steps can be performed independently from human input. On this respect, for the purpose of this demonstration, the image processing of recorded snapshots was performed using the ImageJ GUI (Materials and Method – Image Processing). In future work, the ImageJ Java API will be used to automate image processing and integrate it within the digitization software (Figure 16).

There have been previous attempts [42, 43] to generate random numbers from the chaotic movements of mice in free space, which showed successful results but some limitations. Firstly, a free-space scenario requires adequate measures to categorise movement patterns, ultimately, into an equal number of 0s and 1s. A frequency imbalance would in fact break the RNG security. Moreover, mice are relatively large agents, and such system will be difficult to scale up its final throughput. In contrast, a very large number of bacteria running in parallel in constrained channels, and symmetrical junctions enforcing a binary output, provide a number of benefits. First, each binary output is inherent in the hardware itself (an agent physically takes a definite direction out of two possible choices) and the system does not rely on an algorithmic interpretation of movement patterns based on a human-instructed model, potentially weaker in security strength. Second, bacteria offer a broad spectrum of benefits being low-cost, easy to work with and stable in the long run [35]; they facilitate long lasting experimental results as they are capable of handling large network sizes for scaling purposes. A thorough analysis [44] on bacterial dispersal in microfluidic channels demonstrated that *E. coli* bacteria show faster dispersal time in confined channels spaces than if left in unrestricted environment, hence proving the suitability of *E. coli* motility characteristic for the purpose of quickly spanning the whole network area. In addition, bacterial self-propelled motion enables low energy consumption while still providing a high parallelism [30, 31].

Currently, the entropy source producing 2 frames/s is a limiting factor in throughput terms. However, the network ability to scale given its pattern of identical junctions, and possibly to double (become a square rather than a triangle), brings forward possible alternative designs.

Our results showed that the larger the network the lesser manageable it becomes for experimental purposes. In fact, the 45 junctions network provided 6000 frames and 120K bits per single experiment compared to the 190 junctions network which provided 3000 frames and 80K bits. Therefore, we believe the most convenient scaling approach would *not* be increasing the network size under the given FOV, but rather juxtapose smaller networks over a large area, with each network doubled into a square shape (rather than triangle) and possibly, under rotating microscope lenses to capture different network views in a fixed cycle. The Bits Generator Algorithm task of processing snapshots can be parallelized on multiple processors, or even multiple servers, given

that each snapshot is only evaluated with respect of its previous one (at a previous time step) and does not have dependency to others. Finally, the adoption of a TRNG-PRNG architecture (Figure 18), substantially scales up the final throughput to a competitive level providing 2^{48} requests and 2^{11} bits per request for each reseeding (Table VI). Importantly, by producing a random seed, the TRNG-PRNG model actually behaves as a TRNG. To provide a measure of the capability of this approach, Table VIII compares our TRNG, which benefits of the AES-based CTR DRGB, with the fastest TRNGs on market, which happen to exploit the quantum mechanical uncertainty principle to produce random bits.

Table VIII: Comparison of fastest TRNGs

Company	**Model**	**Performance, Mbits/s**	**Dimensions, mm**
ID Quantum	PCIe	16	160 x 100
Quintessence Labs	qStream	8000	80 x 440 x 680 (2U)
ComScire	PQ128MS	128	80 x 54 x 23
Quantum Numbers Corp	QNG2	1000	0.02 x 0.02
Proposed TRNG	Bacterial TRNG-PRNG	58104 *	1.6 x 1.174

*Evaluated on an Intel i7-8700K CPU model

The TRNG performance was evaluated by testing the speed of AES-128 in CTR mode (Materials and Methods – Comparing fastest TRNG). Given a single seed of 256 bits, the number of available bits produced within a CTR_DRGB period corresponds to $5.7646075e^{17}$ bits (2^{48} x 2^{11}). Therefore, exhausting all available bits of a single period would take 115 consecutive days. Each experiment (25-30 mins) provides a set of frames coding for, on average, ~100K bits, from which almost 400 seeds can be extracted. Therefore, as shown in Table VIII, the final throughput of the TRNG is reduced to the dominant factor, the one provided by the CTR_DRGB, which in turns depends on the speed of AES-128 [45].

Differently from previous approaches [23, 24, 26, 46], an interface between hardware, hosting stochastic biological processes, and digital electronic computers was demonstrated for the purpose of generating true random numbers with competitive high throughput. In addition, the bacterial

run stochastic data source provides considerable improvement in energy consumption compared to electron-based solutions [31] given the autonomous motion of bacteria in the network. This property is potentially advantageous for energy expensive applications such as blockchain technologies. The demonstrated TRNG model envisages potential for future commercial applications, e.g. software as a service (SaaS). For the first time, the feasibility of using random bacterial motion as an entropy-providing process was demonstrated, opening the door to a novel stream of research, i.e. the generation of bacterial-run TRNG microchips.

4.6. Materials and Methods

Microchip Fabrication. Microfluidic networks were manufactured using PDMS-replicas casted from a silicon master. The master was fabricated using electron -beam lithography and Reactive Ion Etching. The casted PDMS parts were treated with air-plasma and sealed on to plasma treated coverslips, followed by wetting with LB-media containing 0.5% BSA (bovine serum albumin). Wetting of the surface was performed in vacuum to remove any air bubbles struck in the channels of the PDMS replicas. The detailed fabrication procedures are described elsewhere [30].

Bacterial culture and experimental set-up. Escherichia coli HCB 437 (*E. coli* HCB437) was used in this study. The competent *E. coli HCB437* cells were transformed to constitutively express the plasmid pMF440 mChe. Bacteria were maintained in positive selection pressure of ampicillin (final concentration of 100 µg/ml) to maintain the plasmid and express constitutively the mChe during sub-culturing and experimentation. All video frames were acquired using 10X (NA - 1.7, WD - 13mm) UPLANO objectives with experimental run time ranging 30 minutes to up to 1 hour in Olympus IX 83 microscopes, equipped with mChe filters (excitation/emission). The exposure times were kept a constant 500 ms so that the best acquisition in fluorescence mode was possible. Image acquisition from the microscope was carried-out using the Metamorph Advanced Olympus software [47]. The acquired frames were post-processed using ImageJ FIJI [48] open source image processing tool. The bacterial trajectories were monitored and analysed using ImageJ tracking software with track mate & MtrackJ plugins.

Image Processing. The microscope acquired frames (also referred to as snapshots) were 16-bit, 1024x1024 sized, subsequently converted to 8-bit. The background subtraction was performed using ImageJ tools. The step includes duplicating a stack for n=100 frames as 1-99 frames as Stack 1 and 2-100 frames as stack 2. When stack 1 is subtracted from stack 2, the resulting 99 frames represent the background subtracted frame which only consists of moving pixels in the frame. With this method, unnecessary signals were removed to avoid interference with the object recognition process performed by the Bits Generator Algorithm. The background subtracted stacks were then binarized, with pixels representing either 0s or 1s. The post processed images were fed to the Bits Generator Algorithm. For the present proof of concept, the aforementioned steps were performed using the ImageJ GUI.

Bits Generator Algorithm. The Bits Generator Algorithm digitizes the displacements of bacteria (agents) captured by snapshots of the microfluidic network every 0.5 s.

To achieve its tasks, the algorithm performs the following steps:

i. Takes a video file (tif) and extracts individual frames;
ii. Applies frames filtering if required, finds and extracts the position of agents from each frame;
iii. Assembles the positions of the agents in a common data structure as a list of elements, each one referring to a single agent throughout the frames sequence. Each element records the *frame of appearance* of the agent, the *agent index* with respect to the total agents appearing in the same frame, the centroid location of the agent, as *x*, *y* pixel values, and finally the size of the *agent area;*
iv. Finds pixel coordinates in the network corresponding to junctions;
v. Tracks the motion of the agents by continuously comparing frame pairs at two subsequent time steps;
vi. Evaluates the displacement of the same agent between two frames using a search radius;
vii. Based on knowledge of the coordinates of the junctions, it decides if an agent took a 'left' or 'right' path (turning event) at the point of crossing a junction. Otherwise, it continues searching;
viii. Maps 'right' directions with 0s and 'left' directions with 1s;
ix. Writes the output in a binary file.

The output bits stream plays the role of raw data within the proposed TRNG architecture, as presented in Figure 14. The algorithm is written in Python 3.6.8 and runs on a Linux Ubuntu 18.4.

The AES_CBC_MAC Conditioning Algorithm. The NIST SP800-90B standards [29] approved cryptographic seeded CBC_MAC [38] was used as a conditioning algorithm, also referred to as conditioning algorithm, to reduce the bias of digitized raw data. The algorithm uses AES for encrypting data and guaranties true statistical randomness. The algorithm takes in entropic raw data as a first input, and a uniformly random seed as a second input, the latter comprising of an encryption key and an initialization vector [12]. We generate the key and initialization vector extracting random entropic values form /dev/urandom with *urandom* function available with the python *os* library. We use an implementation of the AES_CBC_MAC proposed by the Python cryptographic library *blowfish* with Python 3.6.8. The code is run on a Linux Ubuntu 18.4.

NIST randomness tests. A Python implementation of the NIST Statistical Test Suite [49], implemented by the author of the book *'Random Number Generator: Principles and Practises'* [12], was used to test randomness of generated bits. All 15 subsets tests were run on 1M bits on an Ubuntu Linux 18.4. An important parameter within the tests is the value α known as the level of significance, which is selected from the [0.001, 0.01] range [26]. The NIST test releases p values as a measure of randomness for each subtest. If a p is equal to 1, numbers are said to have perfect randomness for that particular test. If p value becomes 0, numbers are not random. All tests must be successful ($p > \alpha$) for a sequence of bits to be considered truly random.

Entropy and Statistical Prerequisite Measurements: The Shannon and min-entropy were tested using a Python script developed for the purpose. The Linux command *head -c file_bytes file_name | ent* validated Shannon entropy estimation and provided measurement on base statistics, such as correlation, mean and kai-squared.

The CTR DRGB Integration. The TRNG functions as an entropy source periodically providing bits in chunks to the CTR DRBG defined by the NIST [39] for cryptographic applications. The set of input bits fed to CTR is called *seed*, a value used to initialize or refresh the internal state of a PRNG. The word Deterministic Random Bytes Generator, DRBG is essentially the NIST's specific term for PRNG. The CTR DRGB is built around the CTR (counter) mode algorithm [39] using an underlying block cipher algorithm. In this implementation, AES with a key size of 128 bits was adopted. The key is used to compose part of the seed material, which comprises of a total seed length of 256 bits (see Table VI for *seedlen = blocklen + keylen*). The optional parameters accounting for 'additional input', or 'provided data' were not provided.

The CTR DRBG available with OpenSSL, was implemented in Python 3.6.8 by building a custom wrapper around the AES block cipher algorithm available with the PyCryptodome library [50] which uses AES in CTR mode. Following the NIST recommendations [51], the data provided for encryption purposes was an array of all zero bytes (plaintext).

Comparison of fastest TRNG: According to the SP 800-90A [39], for large generate requests, CTR-DRGB produces outputs at the same speed as the underlying block cipher algorithm encrypts data [45]. Therefore, the DRGB speed corresponds to the AES underlying speed. The CPU tested was an Intel i7-8700K model. To run the test, the LibreSSL (OpenSSL) library [52] was installed. The performance of the AES encryption algorithm was evaluated per single CPU core. The

following command was run to evaluate the speed of AES-128 (Table VI) in CTR mode: `openssl speed -elapsed -evp aes-128-ctr`.

The resulted output provided the amount of data the CPU can process using the cipher specified (AES-128) in *thousands of bytes per second*, i.e. `7263717.65k`. Table VIII shows the corresponding value in *millions of bits per second*.

References

1. Pandya, D., et al., *Brief History of Encryption*. Vol. 131. 2015.
2. Kar, A., *Cryptography in the Banking Industry*. 2010.
3. Al-Bakri, S., et al., *Securing peer-to-peer mobile communications using public key cryptography: New security strategy.* International Journal of the Physical Sciences, 2011. **6**: p. 930-938.
4. Roelofsen, G., *Cryptographic algorithms in telecommunications systems.* Information Security Technical Report, 1999. **4**(1): p. 29-37.
5. Gunathilake, N.A., W.J. Buchanan, and R. Asif. *Next Generation Lightweight Cryptography for Smart IoT Devices: : Implementation, Challenges and Applications*. in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. 2019.
6. Sklavos, N. and I. Zaharakis, *Cryptography and Security in Internet of Things (IoTs): Models, Schemes, and Implementations*. 2016.
7. Russell, B. and D.V. Duren, *Practical Internet of Things Security: Design a security framework for an Internet connected ecosystem, 2nd Edition*. 2018: Packt Publishing.
8. AlDairi, A. and L. Tawalbeh, *Cyber Security Attacks on Smart Cities and Associated Mobile Technologies.* Procedia Computer Science, 2017. **109**: p. 1086-1091.
9. Burr, W.E., *Selecting the Advanced Encryption Standard.* IEEE Security & Privacy, 2003. **1**(2): p. 43-52.
10. Nisha, S. and M. Farik, *RSA Public Key Cryptography Algorithm – A Review.* International Journal of Scientific & Technology Research, 2017. **6**: p. 187-191.
11. Alabaichi, A., F. Ahmad, and R. Mahmod. *Security analysis of blowfish algorithm*. in *2013 Second International Conference on Informatics & Applications (ICIA)*. 2013.
12. Johnston, D., *Random Number Generators—Principles and Practices: A Guide for Engineers and Programmers*. 2018: De Gruyter.
13. Matt, C. and U. Maurer. *The one-time pad revisited*. in *2013 IEEE International Symposium on Information Theory*. 2013.
14. Bellovin, S., *Frank Miller: Inventor of the One-Time Pad.* Cryptologia, 2011. **35**: p. 203-222.
15. Shannon, C.E., *Communication Theory of Secrecy Systems*.* 1949. **28**(4): p. 656-715.

16. Miller, J.S. and A. Nies, *Randomness and Computability: Open Questions.* The Bulletin of Symbolic Logic, 2006. **12**(3): p. 390-410.
17. Kelsey, J., et al., *Cryptanalytic Attacks on Pseudorandom Number Generators.* Lecture Notes in Computer Science, 2000. **1372**.
18. Huang, Z. and H. Chen. *A truly random number generator based on thermal noise*. in *ASICON 2001. 2001 4th International Conference on ASIC Proceedings (Cat. No.01TH8549)*. 2001.
19. Hasan, R.S., et al., *A true random number generator based on the photon arrival time registered in a coincidence window between two single-photon counting modules.* Chinese Journal of Physics, 2018. **56**(1): p. 385-391.
20. Wayne, M.A., et al., *Photon arrival time quantum random number generation.* Journal of Modern Optics, 2009. **56**(4): p. 516-522.
21. Stipčević, M. and Ç. Koç, *True Random Number Generators*. 2014. p. 275-315.
22. Stipčević, M. and R. Ursin, *An On-Demand Optical Quantum Random Number Generator with In-Future Action and Ultra-Fast Response.* Scientific Reports, 2015. **5**(1): p. 10214.
23. Hedayatpour, S., N. Kama, and S. Chuprat, *A Sticker-Based Model Using DNA Computing for Generating Real Random Numbers.* International Journal of Security and Its Applications, 2014. **8**: p. 113-122.
24. Bogard, C., E. Rouchka, and B. Arazi, *Random number generation for DNA-based security circuitry.* BMC Bioinformatics, 2008. **9**.
25. Bridget, M., et al., *A Survey Of Dna Computing.* 2013. **1**: p. 2321-9467.
26. Arslan Tuncer, S. and T. Kaya, *True Random Number Generation from Bioelectrical and Physical Signals.* Computational and Mathematical Methods in Medicine, 2018. **2018**: p. 3579275.
27. Szczepanski, J., et al., *Biometric random number generators.* Computers & Security, 2004. **23**: p. 77-84.
28. Mitchell, J.G. and K. Kogure, *Bacterial motility: links to the environment and a driving force for microbial physics.* FEMS Microbiology Ecology, 2006. **55**(1): p. 3-16.
29. Turan, M.S., Barker E., Kelsey J., McKay K. A., Baish M. L., Boyle M., National Institute of Standards and Technology (NIST),, *SP 800-90B Recommendation for the Entropy Sources Used for Random Bit Generation.* 2018.

30. Nicolau, D.V., et al., *Parallel computation with molecular-motor-propelled agents in nanofabricated networks.* 2016. **113**(10): p. 2591-2596.
31. Delft, F.C.M.J.M.v., et al., *Something has to give: scaling combinatorial computing by biological agents exploring physical networks encoding NP-complete problems.* Interface Focus, 2018. **8**(6): p. 20180034.
32. Nayak, M., et al., *Bacterial motility behaviour in sub-ten micron wide geometries*. 2018. 382-384.
33. Biondi, S.A., J.A. Quinn, and H. Goldfine, *Random motility of swimming bacteria in restricted geometries.* 1998. **44**(8): p. 1923-1929.
34. Swiecicki, J.M., O. Sliusarenko, and D.B. Weibel, *From swimming to swarming: Escherichia coli cell motility in two-dimensions.* Integr Biol (Camb), 2013. **5**(12): p. 1490-4.
35. DiLuzio, W.R., et al., *Escherichia coli swim on the right-hand side.* Nature, 2005. **435**(7046): p. 1271-1274.
36. Cryptography, I. and D. Eth, *Entropy Measures and Unconditional Security in Cryptography.* 2002.
37. Shannon, C.E., *A mathematical theory of communication.* SIGMOBILE Mob. Comput. Commun. Rev., 2001. **5**(1): p. 3–55.
38. Bellare, M., J. Kilian, and P. Rogaway, *The Security of the Cipher Block Chaining Message Authentication Code.* Journal of Computer and System Sciences, 2000. **61**: p. 362-399.
39. Barker, E.B. and J.M. Kelsey, *SP 800-90A. Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. 2012, National Institute of Standards & Technology.
40. Cohney , S., et al, *Pseudorandom Black Swans: Cache Attacks on CTR_DRBG*. 2019, Report 2019/996: Cryptology ePrint Archive.
41. Barker, E.B., et al., *SP 800-57. Recommendation for Key Management, Part 1: General (revised)*. 2007, National Institute of Standards & Technology.
42. Hu, Y., et al., *A true random number generator based on mouse movement and chaotic cryptography.* Chaos, Solitons & Fractals, 2009. **40**: p. 2286-2293.

43. Wang, X., Q. Xue, and T. Lin, *A Novel True Random Number Generator Based on Mouse Movement and a One-Dimensional Chaotic Map.* Mathematical Problems in Engineering, 2012. **2012**.
44. Weber, A., et al., *Rectification of Bacterial Diffusion in Microfluidic Labyrinths.* 2019. **7**(148).
45. Woodage, J. and D. Shumow, *An Analysis of NIST SP 800-90A*. 2019. p. 151-180.
46. Piva, F. and G. Principato, *RANDNA: a random DNA sequence generator.* In silico biology, 2006. **6**: p. 253-8.
47. Olympus, *MetaMorph Advanced Digital Imaging Software from Olympus*. Biocompare.
48. Schindelin, J., et al., *Fiji: an open-source platform for biological-image analysis.* Nature Methods, 2012. **9**(7): p. 676-682.
49. AndrewRukhin, et al., *NIST Special Publication 800-22: A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications*. 2010.
50. Legrandin, *PyCryptodome Documentation*. 2020.
51. Dworkin, M.J., *SP 800-38A 2001 edition. Recommendation for Block Cipher Modes of Operation: Methods and Techniques*. 2001, National Institute of Standards & Technology.
52. Project, T.O., *LibreSSL, Transport Layer Security protocol*. 2014.

Chapter 5

Discussion

The field of network-based computing has recently emerged [30, 52] with the purpose of harnessing self-propelled computing agents to explore solutions of NP-complete problems in a massively parallel fashion, where the problem is encoded in a graph, or network. Other technologies, i.e. quantum and DNA computing, have thrived to demonstrate new computing models for solving combinatorial problems efficiently, but none have scaled so far due to errors, noise and engineering challenges. Such complications in scaling are tied to the challenge of distributing an exponential number of computing resources, e.g. DNA, biological agents, or qubits, in time and space, which is pivotal to solve combinatorial problems; the success of one technology to provide efficient algorithms, i.e., in polynomial time or better, highly depends on its ability to use available computing resources efficiently. The work in Chapter 1 demonstrates that, whether time, space or matter, *at least one* resource must grow exponentially if an exponential number of operations - at best 2^N - is to be computed to solve an NP-complete problem of increasing input size N. In network computing, this fact is expressed by the difference between combinatorial and multiplication run-modes. While a combinatorial mode proceeds in a 'massively parallel manner' taking advantage of the parallel bacterial motion and the increasing available area as agents move from single entry to multiple exits, i.e. the processing of information grows as the computation progresses, its intrinsic parallelism would not be sufficient to provide an exponential speedup over classical computing. Instead, an exponentially growing number of agents is required to efficiently solve the given instances of an SSP with increasing input size N and such scenario is only possible in a multiplication run mode, i.e. agents actively divide inside the network as the computation progresses. In other words, in a multiplication run mode, exponential time is traded with exponential matter. Figure 10 (via numerical calculation) and Figure 13 (via computer simulation) demonstrate the impact of division on the device scaling law. The implications of such result bring back to the question of P vs NP, which, as discussed, has not been practically proved yet. Therefore, a question arises on whether the exponential speedup provided by division (shown in Figure 13) proves the existence of polynomial time algorithm for solving an NP-complete problem. The answer is negative. What Figure 13 shows is rather a demonstration of the opposite, hence

that P ≠ NP: an exponential number of physical computing resources is required to actually provide a polynomial time algorithm for SSP. Differently from problems in P, solving NP-complete problems necessitates at least one exponentially growing available resource.

Scaling matter: Similarly, DNA technologies require an exponential quantity of DNA to generate all possible combinations of oligonucleotide sequences [15]. By enabling random mixing of molecules in a test tube, the computing time is very fast, i.e. oligonucleotides bind almost instantaneously in the ligation reaction, but on the down side, sequences encoding physically impossible solutions are generated, e.g. in the case of an Hamiltonian path problem, the sequence $v_1 \rightarrow v_1 \rightarrow v_1 \rightarrow v_1 \rightarrow v_1$ is an impossible path. Producing all combinations, including impossible ones, requires a much larger quantity of DNA than if only possible solutions were explored. In this scenario, scaling matter, i.e. DNA, to compute an NP-complete problem of increasing input size N would become quickly unfeasible. Consider the Hamiltonian path problem which scales with $N!$ [18] assuming solutions terms are not repeated such that any possible solution is a sequence of the form $v_i \rightarrow v_j \rightarrow v_k \rightarrow v_z \rightarrow v_q$ where all subscripts are different from each other, $i \neq j \neq k \neq z \neq q$. Generating paths of the form e.g. $v_1 \rightarrow v_1 \rightarrow v_2 \rightarrow v_2 \rightarrow v_1$ where $1 = 1 \neq 2 = 2 \neq 1$, would require $N^2!$ oligonucleotides in the mixture to exhaust the complete solution space (and make sure a single Hamiltonian path is formed, if it exists). Adleman [15] used approximately 3×10^{13} copies of associated oligonucleotides *for each of the six edges* in the graph, possibly to account for potential errors; accounting for errors would require even higher quantity of DNA. In addition, it is worth considering that producing oligonucleotides is highly expensive, which makes DNA difficult to commercialize [20].

Network computing shows a number of benefits with respect of scaling matter, i.e. bacteria: i) biological agents can be recycled (an external channel connects network exits to entry), ii) bacteria are easily available and accessible [57] and considerably cheaper than DNA, which makes bacterial-based technologies easier to commercialize, iii) agents divide under appropriate environmental conditions, hence they exponentially increase in number and vi) bacteria are channelled into available network paths of which there are as many as the number of possible solutions to the given problem ($N!$ for the Hamiltonian path problem). Constraining bacteria from randomly and freely moving reduces the solution space of the problem to the space of physically possible solutions.

The polynomial lines marked in green and orange (for *E. coli* and *V. natriegens* bacteria respectively) shown in the logarithmic graph in Figure 13, demonstrate the competitiveness of multiplication mode-based network computing over other solutions. Quantum-based computation is projected in the form of an exponential, even if one growing with half the speed of those of classical counterparts. The power of quantum computers comes from the ability to generate and manipulate subatomic particles e.g. photons or electrons, known as quantum bits or qubits. Generating, manipulating and ultimately scaling qubits (matter of QC), is an engineering and scientific challenge. One common approach it to build superconducting circuits cooled to temperatures colder than deep space. Alternatively, individual atoms can be trapped in electromagnetic fields on a silicon chip in ultra-high-vacuum chambers [58]. In both cases, the goal is to isolate the qubits in a controlled quantum state; by controlling N qubits, we can create 2^N linear combinations of their spin states *simultaneously* (a *superposition* of quantum states). On the contrary, a conventional computer with N bits at any given moment must be in only *one* of its 2^N possible states [56]. The fact that particles can be in multiple states at once, which is in contrast to classical bits, finds profound explanations in the nature of wave-particle duality of matter [59]. It suffices to say that a particular problem, e.g. an NP-complete, can be encoded by the linear superpositions of quantum states of the controlled particles and consequently, an increasing number of qubits would bring an exponential increase in quantum computing power. How is information processed in such a machine? By applying "quantum gates"—the respective of classical logic gates—that change particles spins in a precise and controlled manner [56]. It was estimated that, to compete with a common laptop, a quantum computer would require between 1,000 to 100,000 qubits, meaning that such a machine would sustain being in $2^{1,000}$ states at once (10^{300} which is more than the number of subatomic particles in the universe) [56]. Clearly, scaling qubits is a physical challenge. In addition, one must consider the effects of errors. As discussed in previous sections, errors in classical computers only exist at the input and output level. Any potential misalignment happening at the level of transistors (a transistor is switched off when supposed to be on and vice versa) is accounted by error-correction methods, which make use of some level of redundancy built into the hardware [56]. Error correction has not yet been proven to scale for quantum computing [60], therefore the possibility of maintaining controllably low error rates for 10^{300} simultaneous parameters is not foreseeable in any near future.

Scaling time: The projected performance of QC when solving instances of SSP is rather optimistic (Figure 13). It assumes that the aforementioned scaling-related problems were not an obstacle to quantum computing and tries to calculate how QC would scale if 1,000 or more qubits were working effectively with negligible error rates. A quantum polynomial-time algorithm for an NP-complete problem has not been demonstrated yet [61], and currently Grover's algorithm [29, 62] provides the only projection on quantum speed up over NP-complete problems. Grover demonstrated that, rather than an exponential, QC can only provide a quadratic improvement over classical algorithm. Such improvement is considerable (exponential grows at half the speed of that of classical computers) but, as shown in Figure 13, the quantum computing time still grows exponentially fast.

Figure 13 does not draw a comparison with DNA computing which, for a complete picture, should be addressed in future work. From the perspective of scaling DNA computing times, it can be observed that the computation stage, denoted as t_{comp} (actual build-up of the problem solution), coincides with molecular reaction times. Multiple factors can influence the success of ligation phases, including temperature, time, DNA concentration, amount of Ligase enzymes and length of legated fragments, where the longest the fragments the more challenging the ligation [63, 64]. The booting time, t_{boot} which corresponds to the time to initialize the computation, coincides with the molecular incubation time required for a successful ligation. Depending on the temperature, incubation and consequently ligation, may take between 10 mins to 16 hours. At 16°C incubation happens overnight (~12 hours) while at room temperature it may take even 2 hours. High concentration of DNA Ligase can be used in a 10 minutes ligation [64]. DNA ligation is normally carried out at 12–16°C (~12 hours) to ensure maximal ligation efficiency [65]. Low temperatures generally reduce ligase activity, whereas too high temperatures may reduce cloning efficiencies by melting annealed DNA overhangs and increase overall molecular motion in the ligation reaction [63]. Finally, the readout time, $t_{readout}$ which refers to the time taken to decode and read the computation output, e.g. filtering and sequencing DNA strings, is the most time-consuming process. The series of laboratory operations performed to read DNA solutions require considerable human manipulations, carried out sequentially and with low levels of automation [20] at the cost of exponential time.

The DNA computing time, t_{comp} may be described as relatively efficient from a computational standpoint. The ligase reaction time does not scale exponentially with the concentration of DNA, but rather, in a high concentration solution, DNA strands have better chance to pair relatively fast. Possibly though, multiple ligation phases may be required to handle a large amount of DNA [65]. As previously discussed, random mixing of simultaneously present DNA strands come at the cost of generating a larger solution space than actually needed (including physically impossible solutions) and, consequently, of requiring an impractical amount of DNA (it was estimated that solving a TSP with 200 vertices would exceed $3*10^{25}$ kg! [66]). Contrary, in network computing where, as previously mentioned, bacteria are recycled, they never coexist all simultaneously inside the network: they move in and out, coming back to the entry using external channels. Consequently, network-based calculation uses considerably less mass of agents, but at the expense of a much larger computation time, t_{comp}.

Overcoming traffic density in a network device: One engineering bottleneck in a network device lies at the entry point of the network, or relatively close to it. The restricted available area surrounding the single entry, compared to a much larger area towards the bottom of the network, represents a zone of blocking, or high traffic, which delays bacterial flow and consequently the entire computation. Such bottleneck results from saturation of confined space and typically happens in a combinatorial run mode: as agents are expected to distribute in time and space, they remain blocked if not enough space is available. On the other hand, enlarging the network channels (usually between 2-4 um) would enable bacteria to perform U-turns in low traffic conditions. Given the device does not support subtraction when solving the SSP, U-turns are highly undesirable as they create two-ways traffic without bacteria reaching the bottom exits to fully explore solutions. Under traffic conditions, the booting time, t_{boot} of the device, which coincides with the total time bacteria take to enter the network at the start of the computation, is considerably large and, when summed to t_{comp}, time becomes impractical (Figure 7). Consequently, traffic density must be avoided for a successful computation, a reason why the combinatorial run mode was demonstrated not to be an optimal solution [13]. It is worth noticing that t_{comp} is itself exponential in a combinatorial run mode (Figure 7a), signifying that a higher number of agents to simultaneously coexist in the network and explore the solution in parallel would be needed to provide a more efficient solution. As expected, such possibility is hindered by the existence of traffic density. Figure 7a demonstrates the average relative traffic density versus cardinality when

instances of SSP are solved by the computing device with input numbers belonging to different problem series. Both Figure 10 and 13 demonstrate how computing time scales when instances of SSP with numbers of the *prime number series* are used. This observation has relevance since the size of an SSP network depends on the compactness of the series i.e. the relative distance between numbers in the set, and the more compact the series, the smaller the network. The smallest SSP network solves a binomial expansion series, e.g. Pascal's triangle, while the largest network solves a factorial series. In Figure 7a expected traffic density when networks solve the prime number series (green line) are compared with other network sizes. The larger the network, the lesser the expected traffic density (as shown by the exponential (violet line) and the factorial (brown line) number series). Figure 7b demonstrates the substantial improvement that a multiplication run mode brings to traffic density. Under multiplication conditions, agents are able to divide inside the network, therefore, increasingly populating the bottom and benefiting of enough space at the top. This is because, in practise, by enabling a restricted number of agents to enter the network, the device is able to solve the problem efficiently by experiencing bacterial division as the computation proceeds. As a result of the exponential number of agents exploring the bottom exits simultaneously, the device is able to solve the SSP in polynomial time (Figure 13, green and orange line). One consideration must be made with respect of the division phenomenon, such that, for the first time, a machine is conceived to have its computing resources increasing as the computation progresses. Theoretically, if an exponential number of agents could grow fast enough to map the SSP solution space, computing times could be reduced ever further. The physical constraint of a limited available network area inhibits such growth since, in those conditions, even the bottom of the network could experience traffic density.

The halting condition: The multiplication run mode brings noticeable improvements on both the booting time t_{boot} and the computing time t_{comp} of the device. Previous considerations provided clues that the total time to compute a problem instance is not only a function of the input size, but also a function of the number of computing agents exploring the network. Nevertheless, some agents may be able to make 'errors', thus not providing defined solutions. The ability of bacteria to make errors underlines the stochastic nature of the computing device. Errors arise at pass junctions (PJ), designed to force a given bacterium to maintain its current direction. In other words, if an agent comes from a straight path, it should continue going straight after crossing the junction. Potentially, agents can succeed in taking a 'difficult to reach' route, thus entering a forbidden path

and exploring an exit which is not a solution (in Figure 3, exits marked with magenta index are not solutions). One must remember that, when solving an SSP, knowledge of correct and incorrect solutions is only derived from the ability to distinguish between explored exists (correct) and not explored ones (incorrect). Therefore, if the pass junction error rate is high, results cannot be discerned. In addition to errors performed by bacteria, chip fabrication errors may be a cause of erroneous computation. Due to the occurrence of errors, one question arises. When are we going to be confident enough of computed solutions, or in other words, when do we to stop the computation? Such question partially resembles Turing's Halting Problem [67], but with some qualifications. Certainly, the minimum number of bacteria required to explore all exists depends on the PJ error; with high error rates, all exits will be explored with some magnitude and it would be impossible to recognize correct and incorrect solutions. Ideally, if errors were not an option, it would become very clear and very fast which solutions are correct based on which exists are visited. Given the sophisticated level of precision in the fabrication of the chip, experimental PJ error was observed to be at 0.01%. With such error rate, the formula of the coupons' collector problem [53] provides a suitable measure for scaling the minimum number of agents to solve instances of SSP. In this context, the coupons' collector problem asks, given a problem of input size *N*, what is the minimum number of agents required to consistently explore all 2^N combinations with >95% probability. In the attempt to minimize the number of agents (which reduces computing time) while still enabling agents to provide a reliable solution, a factor of two (x2) was added to the coupons' collector formula (CCF). The formula shown below was adopted as a halting condition (with results presented in Figure 10 and 13).

$$\textit{Halting Condition} = 2 \times CCF$$

$$CCF = ((2^N \times \ln(2^N) + 0.5572) + 0.5 - 1) \times \frac{0.5}{0.45}^N$$

where *N* is the input size (cardinality) of the problem instance and 2^N are the possible solutions.

Reducing error rates: It is important to note that, once the computation gets to a halt, identifying errors for discerning the true solution (readout) is not a trivial task. Some exits are naturally more explored than others, depending on the problem instance to be solved and the motility characteristics of the running bacteria e.g. *V. natriegens* has a preference with turning left. This uncertainty has consequences on the interpretation of results. Some exiting regions of the

network may show a much higher concentration of agents compared to others e.g. in the case of prime number series, the output distribution resembles a bell shape, with tails being less heavily explored. Consequently, it is possible that incorrect exists positioned in highly visited regions e.g. centre of the distribution, will be explored by a higher number of agents than correct ones positioned in poorly visited regions e.g. tails of the distribution. Clearly, from a scaling perspective, reducing if not removing PJ errors would be ideal. Performed simulations showed that an error rate of >2% would be detrimental for computation efficiency. To address this issue, a proof of principle of 3D-SSP junction networks composed of bridges and tunnels was demonstrated in recent (not yet published) work. The 3D crossings physically separate the bacterial traffic at the pass junctions, inhibiting them from taking forbidden turns and errors. This approach, aimed at an error-free computation, was tested using an SSP network with 16 exits and cardinality equal to 3, i.e. SSP (2,5,9) using *E. coli* agents. Results successfully demonstrated a zero-error output in the 3D network compared to the 2D network. Analysis of 3D networks will be further perused in future work.

Defining solutions: The combinatorial nature of SSP is expressed by the distinction between Complexity Class I and II (CC-I and CC-II) problems, described in Figure 3. In CC-II, some exits may be reachable through multiple routes, while in CC-I each exit has only one path to it. The importance of the Complexity Class distinction comes apparent when different facets of the original SSP definition are considered. Informally, the simplest one, here called Q1, asks, if there exist any subset sums of a target value T given the set of input elements E. In principle, experimental and simulated demonstrations of bacteria exploring an SSP network suggest that the counts of bacteria at each exit contain enough information to solve a CC-II problem even without any representation of the network (density maps), any prior knowledge of the set employed and any knowledge of the individual routes taken by bacteria. Therefore, in order to answer Q1, the readout time $t_{readout}$ is instantaneous (observing the counts of agents at the exits). To give a concrete example, consider a CC-II network with T = 42 and input elements E such that E = (2, 5, 9, 11, 15). Judging from counts of agents, it can be discovered that valid subsets S are as follows, S = (0, 2, 5, 7, 9, 11, 13, 14, …, 39, 41), where S provides all subset sums of T for the given problem instance. On the other hand, Q2 asks what are the precise components of each solution in S. Consider the same example network above where T = 42. Let us pick 11 as one of the solutions in S which is a CC-II solution, hence it has multiple routes taking to the same exit 11: Q2 wants to

know what are the subsets of 11 (and other CC-II exits), not only 42! This problem, which aims to know what are the exact CC-I subsets of CC-II exits, can only be solved by tracing back the possible routes to every CC-II exit. The given example is a trivial case and, with few calculations one may quickly realize that, among the solutions in S, S' = (0, 2, 9, 11) are subsets of 11, where in fact 11 = 11+0 = 9+2, while 5 and 7 are not. Nevertheless, when complexity increases, discerning the exact subsets of all CC-II solutions becomes increasingly (computationally) hard. Clearly, Q2 adds another layer of information but with the resulting answer (what are the subset sums of 42) being essentially the same as Q1. The variation between Q1 and Q2 may have relevance depending on the practical application of the SSP.

Integrating the readout. To answer Q2 correctly, a considerable number of operations must be carried out and, at the same time, the readout time $t_{readout}$ should also be minimized as much as possible. One simple method for reading out the computation output is the use of density maps of the whole network area (cumulative sum of network images) showing the most visited routes and exits. Density maps were produced to compile experimental results shown in Figure 13 (green and orange lines). This method provides visual clues rather than a numerical solution for the different routes and requires multiple microscopes and stitching images to visualize very large networks. One technology discussed in Chapter 2 which could potentially allow to discriminate between agents taking different routes in CC-II problems is dynamic fluorescence tagging (Figure 11). In such scenario, agents are given a fluorescent colour label when taking a route. When agents exit the network, their colours represent the particular routes taken by each of them. A counter would increment values by counting agents out of each route, rather than each exit. In practice, bacteria must be genetically modified to express fluorescent proteins which get excited and emitted with selected wavelengths [68, 69]. Therefore, the engineering challenge of this approach is non-trivial, in particular in the context of scaling the computing device and consequently its number of paths, meaning that an exponential number of colours must be provided, one for each path. An alternative solution would be to monitor bacterial motion with a tracking software assigning IDs to individual agents. As for tagging, a counter would increment values per each route as the computation proceeds. The feasibility of this approach is higher compared to tagging, and possibly doable at relatively modest cardinalities, C (up to ~30C). Multiple microscopes covering the full chip area would be required to record the history of bacterial flow. Nevertheless, it must be noted that, at higher cardinalities, the number of bacteria to be tracked in parallel would become impractically

large (possibly an even harder engineering challenge than tagging) and it would also become impossible to include the whole chip area under the available field of views with a resolution still allowing identification of individual agents. As shown in Figure 8 in Chapter 2, the *E. coli* cell widths can only be resolved by a high-resolution optical microscope, while the benefit of a larger bacteria, e.g. *E viridis*, would come at the cost of a proportionally larger network size.

Scaling area: Figure 13 demonstrates the ability of the computing device to solve instances of SSP in polynomial time given the combinatorial and independent nature of bacterial division. Until now, some concerns have been expressed with respect to scaling the network areas. As shown in Figure 13, *V. natrigens*, which multiplies at a rate almost three times faster than *E. coli*, would be able to undertake an AMD Rayzen past cardinality ~45, and in theory, a Pentium Pro operating as a QC past cardinality 90. Nevertheless, such high cardinality networks cannot be fabricated without an innumerable amount of errors using e-beam lithography and chemically assisted HF-cold etching. As a result, scaling the area, and effectively the network, is a fundamental challenge that requires additional future research. One point must be made with respect of the different network sizes based on the structure of the number series. Solving increasingly hard instances of SSP using numbers of the factorial and exponential series requires the network area to scale exponentially fast while using elements of the prime number series (a more compact series) only requires it to scale polynomially. From Figure 9, it can be deducted that by using *E. coli*, or an agent with similar size, the area of a device solving SSP with primes numbers and a cardinality of 30 is slightly larger than a 6 inches wafer while, for the same wafer size, only a problem of cardinality 15 can be attempted with numbers of the exponential series. One potential solution of scaling the network area would be to run multiple networks in parallel. If a prime numbers network with cardinality 25 is used (close to the maximum that could be physically attempted), which has 1060 exits, one would need 1061 networks running in parallel, 1 running a 25-cardinality network and the other 1060 functioning each one as a continuation of an exit of network 1. Such operation would require some computational redundancy as it is not known a-priori what are the valid exits of network 1. Therefore, among the 1060 networks, some will be excluded only at the end of the computing time.

Dynamic programming: It was argued [70] that the computing device as described by Nicolau et al. [52], if surely a remarkable piece of engineering, does not circumvent the problem of

exponential time bound solutions when solving the SSP. It was also pointed at dynamic programming (DP) algorithms which solves the problem for subsets of increasing size using the known result from previous steps, thus avoiding to compute operations multiple times; from a network computing perspective, essentially, they skip all pass junctions to only compute operations at split junctions. In particular remarkable is the DP algorithm proposed by Pisinger et al. [71] which proposes the use of a technique called *balancing* aimed at discovering a problem solution by mean of 'some reasoning' e.g. if a possible solution found is excessively small to hit a target value, than smaller numbers composing even smaller solutions would certainly not be valid either and would not need to be computed. Essentially Pisinger [71] formalises such reasonable assumptions and successfully delivers an algorithm for SSP which scales in $O(NW)$, where N is the input size (cardinality) and W is the largest (in magnitude) integer in the set. Certainly, solving SSP with a DP algorithm would yield a much faster result. In previous work [52] a *brute force* algorithm was demonstrated, which by definition, explores all possible combinations, even if repeating operations must be performed. Therefore, as it was observed [70], the biological computer would scale exponentially fast following a similar trend to electronic computers. What was not observed, as in fact a recent result, is the ability of the device to run in polynomial time with bacterial division *while still solving a brute force algorithm*. These results, shown in Figure 10 and 13, demonstrate the advantage of a the proposed novel computing model: here the biological device, still a its infancy, is directly compared with various models of electronic chips, some still at their infancy (Intel i286, i385, i486) from an operation and computing time perspective. Introducing clever algorithms at the level of DP techniques shall be addressed in future research.

Justifying the choice of bacteria as computing agents: Chapter 3 provides a glimpse of additional work performed on SSP networks in the last year, which includes proof of concept of 3D networks and an experimental demonstration of division in the networks. Differently from previous work [52] which used self-propelled motion of Myosin II and kinesin-1 molecular motors, bacteria were employed for the latest demonstrations due to (i) the existing large variety of candidate agents with detailed information on velocity, flagellar arrangements, sizes, morphology, and dividing rates; (ii) bacteria require low-cost and uncomplicated experimental procedures, for which well-known protocols exist; (iii) a wide range of fluorescent tagging and gene editing tools and programmable traits are readily available; (iv) bacterial cell width are large

enough to be clearly distinguishable by different imaging techniques and importantly (v) bacteria divide, thus providing self-replicating computing power.

Algorithms is network computing: A computing device must be able to solve different algorithms. In network computing, an algorithm is essentially the network provided as input, with its geometry encoding a particular problem. As a first alternative to the SSP, the computing device was used as an entropy-providing resource of a TRNG, discussed in Chapter 4. Differently from the SSP network having both split and pass junctions, for this application, the network has only split junctions (designed to let agents take either path with 50-50% chance). Clearly, the presence of pass junctions would deny the possibility of generating bits of maximum entropy (which is based on a fair probability split of ½ between two possible choices) and would therefore reduce the overall entropy per byte provided by the entropy source. Increasing the density of split junctions on a given network area represents a possible solution to maximize the rate of output bits and it would be worth attempting it in future work. This solution is expected to be successful as it would not have impact on the computing time of the Bits Generator Algorithm, which in fact, only depends on the number of frames (images), f and the number of agents in each frame squared, n^2 and not on the number of split junctions. Given the large number of bacteria to iterate through in each frame, the initial performance of the Bits Generator Algorithm $O(fn^2)$ was substantially improved using parallelisation of matrix operations (enabled by the Python `NumPy` library [72]) and multiprocessing computational techniques.

Reliability of an entropy source: The proposed TRNG generator aims to produce truly random numbers for cryptographic applications. In cryptography, the unpredictability of secret keys is essential. Therefore, the TRNG must comply with the strict requirements given by the National Institute of Standards and Technology (NIST). According to the NIST Recommendation for Entropy Sources, [73], the minimum entropy, rather than the commonly used Shannon's entropy, represent the most accurate description of the reliability of a given key and its security to potential attackers. The minimum entropy produced by the entropy source is shown in Table IV, Chapter 4. The security of the TRNG relies on this value, which in this case, is sufficiently high (the digitized raw data have 7.791 out of 8 bits/byte and the entropy source data have 7.833 out of 8 bits/byte). Therefore, the non-deterministic random laws inherent in the bacterial-run computing device demonstrated evidence of validity in the generation of high entropy data. In order to further assess

the entropy source reliability and optimality, the entropy source must be validated by accredited laboratories [73]. Working to meet additional validation requirements, which include providing a command line interface to extract entropy and integrating health tests, shall be the next steps in future work.

Chapter 6

Conclusion

This contribution defines the mathematical principle and current state of the art in the network computing field and estimates its potential to solve computationally hard problems efficiently. In addition, a novel application of a network computing device is proposed and demonstrated in the context of generating true random numbers.

Solving NP-complete problems efficiently and generating truly random numbers are both fundamental problems in the modern society which ‘von Neumann’ based computers fail to address given their strictly deterministic and sequential computing model. New technologies have emerged to solve such problems and among them, network computing provides innovative and promising approaches to handle combinatorial operations in a massively parallel fashion and with high energy efficiency. The question of whether a network-based device would be able to scale is pivotal for the success of this novel approach, hence a research question addressed in Chapter 2 and 3. This contribution identifies the scaling related key technological challenges in terms of chip fabrication, readout, reliability and energy efficiency in the field, thus providing a road-map to future developments. The research clearly illustrates that, among available resources such as time, space and matter (computing agents), a trade-off always exists in the attempt to scale the computing device: to achieve an exponential reduction in computing time, an exponential number of computing agents and an equally growing space variable must be managed. On this respect, this contribution identifies potential ‘auxiliary’ technologies and solutions that, if adopted, would enable the device to scale successfully. In particular, while the presence of bacterial error and fabrication error (increasing with larger wafer sizes) may limit scaling to very large cardinality, the use of 3D junctions and parallel computation of multiple independent networks (max 6 inches wafer) would be potential solutions worth exploring in future work. The competitiveness of the novel approach is demonstrated using computer-aided simulations and experimental data which, for the first time, demonstrated the impact of bacterial division, actually proved by experimentalists. A comparison with DNA technologies shall be integrated in future work to provide a complete picture of different computing models and their scaling laws.

Network computing does not harness the ‘intelligence’ of bacteria, but rather their random exploration of the network and, importantly, their ability to divide. Consequently, an innovative design of a high throughput TRNG for cryptographic applications was conceptualized and demonstrated, for the first time exploiting random bacterial motion inside a microfluidic network. The random processes inherent in the biological nature of *E. coli* HCB437 provide the unpredictability required to generate high entropy data demonstrating the i) suitability of network-based computing in applications other than NP-complete solvers, ii) the possibility to generate non-deterministic data using the *E. coli* HCB437 strain in confined spaces. If further research is needed to fully assess the entropy source validity for cryptographic uses, this demonstration opens the future possibility to design bacterial run biological chips for novel cryptographic tools.

Based on these conclusions, network-based computing with self-propelled and dividing agents represent a promising avenue for solving computational problems of modern societies. The novel results discussed provide a reference for new developments and applications in the field.

Appendix A

The following Appendix with the Supplementary Information of the paper *Something has to give: Scaling combinatorial computing by biological agents exploring physical networks encoding NP-complete problems* published in Interface Focus.

- Supplementary Information –

Interface Focus

Something has to give: Scaling combinatorial computing by biological agents exploring physical networks encoding NP-complete problems

Falco C.M.J.M. van Delft, Giulia Ippoliti, Dan V. Nicolau Jr., Ayyappasamy Sudalaiyadum Perumal, Ondrej Kaspar, Sara Kheireddine, Sebastian Wachsmann-Hogiu, Dan V. Nicolau

SI-1 Mathematical formulation of network encoding

The ‘hardware’ component of the computational network-based devices, i.e., networks whose design reflects a particular problem of interest, could be mathematically formulated as follows. An arbitrary decision, i.e., yes/no, problem (D) whose instances, i.e., specific cases of the problem, are denoted as D_1, D_2, … can be represented by a class of networks of nodes and connections, E, that captures the structure of the decision problem. This representation is made in such a way that, for every instance D_i of D, one can find a directed graph, E_i, in which a path from any of a predefined set of input nodes to any of a predefined set of output nodes exists if — and only if — the decision problem on D_i (as defined above) has an answer in the affirmative. Conversely, we need to guarantee that such a path cannot be found if the decision problem on D_i has a negative answer. If the NP-Hardness Assumption is true, then the number of unique paths through the networks encoding the instances will, in the worst case, grow exponentially with instance size, thus necessitating the use of a number of agents larger than this number of paths, in order to comprehensively explore the solution space. Consequently, the engineering future of computing with networks rests on finding a physical implementation strategy that is able to marshal potentially very large numbers of agents.

SI-2 Nomogram for Field-of-View and Cardinality in case all agents move through the network channels in singular queues (no overtaking).

The expected chip size is shown in the nomogram in Figure SI-1 (left hand side) as a function of the channel widths (dependent on the agent sizes) for the "prime number" SSP devices for various cardinalities. The horizontal black dashed lines delimit the sizes of 4, 6 and 8 inch silicon wafers – the standards in the semiconductor industry. The vertical yellow bars indicate the line widths for devices running molecular motors-driven cytoskeletal filaments, i.e., actin filaments, microtubules, as well as small (*E. coli*) and large (*E. viridis*) microorganisms. As examples, the green, red and black arrows indicate the chip sizes of, respectively, the cardinality 15 network for microtubules, the cardinality 5 network for *E. coli*, and the cardinality 5 network for *E. viridis*.

Because of the competition between resolution and the FoV, the whole imaging of the computing area requires the employment of the maximum useable pixel size (MUPS) that can still discriminate the individual agents, as well as the legal (and illegal) turns they take. Regarding the latter, the middle panel in Figure SI-1 presents the SSP chip split junction design with a given channel width (top). Also, the split junction design is shown with an overlay of pixels 5 times larger than the channel width, which is the limit that would still allow the system to follow vertical traffic (in blue), diagonal traffic (in yellow), and whether the splits are being used for changing traffic direction (in purple), or if the central pass crossing (in green) is used. The legal pass and split traffic is indicated by the groups of three squares underneath the overlaid design. A typical example of an illegal turn that would be detectable is yellow-green-blue. Apart from being 5 times the channel width, the MUPS value should also allow discernment of the individual agents. Hence, the MUPS value should also be smaller than the agent length.

The right-hand side of Figure SI-1 represents the square root of FoV versus the MUPS value. The black crosses indicate the intersection of the largest attainable FoV (as a square root) with the minimum attainable MUPS for various optical imaging technologies, i.e. their resolution limit. The useable optical range is obtained by the intersection of sqrt(FoV)-MUPS range with the diagonal black line indicated as 'Unity'. At the point where the top horizontal border meets the 'Unity' line, the MUPS value is equal to the total FoV, meaning that only one pixel fits in the frame, obviously far from any reasonable application. To fully exploit the frame size available, the

MUPS value should be as close as possible to the resolution limit. The vertical blue bars indicate the MUPS for devices running molecular motors-driven filaments, and small-and large microorganisms. Similarly to the left panel in Figure SI-1, the horizontal green, red and black arrows copy the chip sizes of respectively the cardinality 15 network for microtubules, the cardinality 5 network for *E. coli*, and the cardinality 5 network for *E. viridis*. If the spot where the vertical green, red, or black arrows, respectively, meet their equivalent horizontal arrows is inside a 'technologically-achievable' sqrt(FoV)-MUPS triangular area, the corresponding optical technique is, in principle, useable for monitoring the computation process using a single FoV. It follows that a cardinality 5 network for *E. viridis* can be monitored by a macro-lens equipped camera, whereas the cardinality 5 network for *E. coli* and the cardinality 15 network for microtubules can be monitored by a lens-less microscope.

Note that this scenario assumes that there are no agents overtaking each other in the channels either laterally or by crawling over each other.

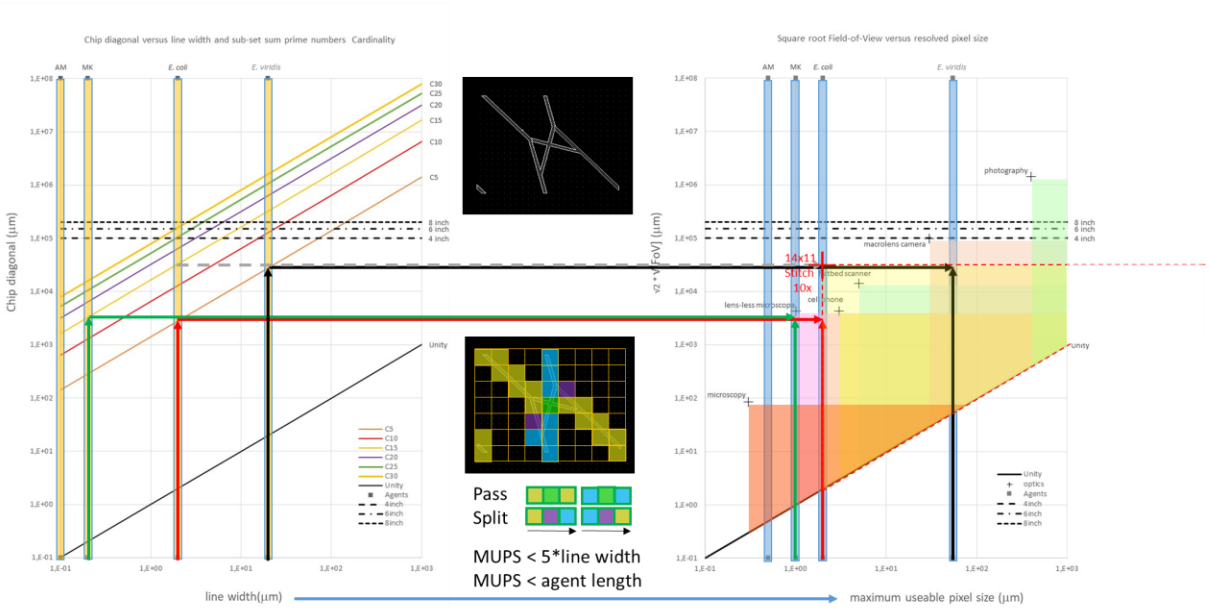


Figure SI- 1: Nomogram for network size versus line width and cardinality, and Field-of-View versus Maximum Useable Pixel Size and resolution. Also shown is an example of an enlarged work window by image stitching (red cross); the limits of this method are discussed briefly in the main text and in detail in SI-3.

SI-3 Stitching FoV's while keeping track of all agents

When the area to be imaged exceeds the Field-of-View (FoV) of the imaging system, image stitching can be performed to 'bridge the gap', although not without some loss of information. Of course, this loss can be minimised through faster switching speeds, which in turn are limited by the mechanical capabilities of the microscope stage. To perform these calculations, the following factors should be taken into account:

- Speed of biological agent used
- Exposure time of each partition of the image

For the sake of simplifying our sample calculations, we are assuming no horizontal or vertical image overlap, negligible exposure times (which in reality can go as low as 1-10 ms for bright field imaging and as high as 500 ms for fluorescence imaging), perfect functioning of pass and split junctions in our subset sum networks (SSN), and unidirectional movement of our biological agents with no U-turns. Factoring in the density of the agents necessitates choosing the optimal switching speed based on dependency on body length in case of high agent density, and based on distance between adjacent junctions in case of low agent density.

For the scenarios described below, numerical examples are shown for E. *coli* K-12, where body width (BW) = 0.5 μm, body length (BL) = 2.5 μm, and speed v = 10 μm/s. Vertical and Diagonal Junction Distances (i.e. between two adjacent junctions) for the subset sum problem network (SSP) fabricated with 2 um-width channels for E. *coli*, are 60 μm (VJD) and 100 μm (DJD), respectively.

- In the case of high agent density (scenario where **more accurate** tracking is required)
  1. Speed of agent v = 10 μm/s
  2. Speed v_b of agent in body lenghts/s: $v_b = v / BL = 4$ bl/s
  3. Time period to return to original image partition (in case we wish the agent to have moved only ½ a body length) $t_r = 0.5*BL/v = 0.5bl / v_b = 125$ ms
  4. The average switching time between partitions t_s is determined by $t_s = t_r / n*m$, where n and m are the number of partitions in x and y respectively (e.g. 2x2, 3x4). Hence, for a 2x2 stitching mode, $t_s = 125ms / 4 = 31.25ms$

- In case of low agent density (scenario where **less accurate** tracking is acceptable)
    1. Speed of agent v = 10 μm/s
    2. Speed of agent (v_v or v_d) in vertical- or diagonal junction steps per second
       v_v = v / VJD = 0.17 vjs/s and v_d = v / DJD =0.1 djs/s
    3. Time period to return to original image partition (in case we want the agent to have moved by 1 vjs or 1 djs) t_v = VJD / v = 1vjs/ v_v = 6s and t_d = DJD / v = 1djs/ v_d = 10s
    4. The average vertical switching time between partitions t_{sv} is given by t_{sv} = t_v / n*m , where n and m are the number of partitions in x and y respectively (e.g. 2x2, 3x4); the average diagonal switching time between partitions t_{sd} is given by t_{sd} = t_d / n*m
       Hence, for a 2x2 stitching mode t_{sv} = 6s/4 = 1.5s and t_{sd} = 10s/4 = 2.5s.

The calculations shown above describe two extreme cases in terms of agent density, and as a result the necessary corresponding tracking accuracy. However, in real-life situations, we will mostly have intermediate scenarios that fall between both, and it would be up to our discretion to choose the appropriate values to use.

As such, the overall parametric equation to calculate switching time (t_s) can be described as:

$$t_s = \frac{t_r}{nm} = \frac{\mathrm{d}}{v^* nm}$$

where t_s is the switching time between image partitions, t_r is the time period to return to original image partition, n is the number of image partitions in the form ‘n x m’, d is the displacement we would like our agent to have moved by the time we return to the original image partition expressed in terms of BL/VJD/DJD etc., and v^* is the agent speed expressed in terms of BL/VJD/DJD etc. rather than in unit length per second.

The switching time (t_s) encompasses two values, namely translation time (t_t) and exposure time (t_e), where t_t can be expressed as:

$$t_t = \frac{\text{FoV}}{v_{ss}}$$

where *FoV* is the diagonal field-of-view of the system, and v_{ss} is the speed of the scanning microscope stage. Moreover, t_e is dependent on illumination technique (e.g. bright field), system magnification (inversely proportional), and when using fluorescence, the fluorophore used. It follows from this that $t_s \geq t_t + t_e$ in order to achieve the required agent tracking accuracy.

In terms of what the current technology can offer, the fastest motorised translational stage, to the best of our knowledge, offers a maximum speed of 250 mm/s, with an accuracy < 3 um. This stage utilises a servo motor as its actuator, and is available from Thorlabs Inc.

Once again, using two extreme scenarios for the FoV can help shed light on what this system could help us achieve. If we use a 100x objective (NA=1.4), which gives us a FoV ~ 60 um, we get t_t = 240 us. When pairing this with a reasonably low t_e, one can achieve high accuracy with a large number of partitions, and a high resolution (~ 0.5-1 um), but the overall FoV covered would be quite small. On the other hand, if we use a 2x objective (NA=0.3), with FoV ~ 1.5 cm, we get t_t = 60 ms. Based on our previous calculations, regardless of the t_e used, we will not be able to use this system to scan even the smallest (2x2) partition matrix for the most accurate tracking scenario (t_s <= 31.25 ms). However, we can still use this for a slightly less accurate tracking scenario while covering a larger overall FoV with a fairly good resolution (~ 3-5 um).

Typically, for observing E. *coli* in the SSN, we use a 4x objective (NA=0.16) under fluorescence, with t_e = 500 ms. In order to observe the largest chip we have with cardinality 30 (C30), which has an area of ~ 13x10 cm, and using the same 4x objective with FoV ~ 1 cm, we would need 13x10 partitions. For this set-up, and using the more accurate tracking approach, we would need a t_s = [125 ms / (13*10)] = 0.962 us = ~ 1 ms, which is unattainable given the large t_e requirement. Moreover, using the less accurate tracking approach, we would need a t_s = [10 s / (13*10)] = ~ 0.077 s = 77 ms, which once again is unfeasible. If, however, we disregard t_e, we would need a v_{ss} = 10 m/s, and v_{ss} = ~ 13 cm/s respectively. Since the SSN has a triangular profile, and only half of the rectangular partition grid will be traversed by the moving stage, the calculated t_s should be multiplied by 2, which would result in half the speed requirement. This, even with the improved requirements, is nowhere near feasible given the current available technology. While

ignoring t_e is unrealistic for tracking E. *coli*, it could potentially work for bacterial species that provide good contrast, such as cocci.

Table SI-I shows which chips can be observed with image stitching using the specific objectives available on an Olympus IX83 microscope. This comparison was made using appropriate exposure times for each magnification, in addition to the translation time using the stage speed available (13 mm/s).

It shows that -with stitching- up to C15 can be observed with a 10x objective and up to C20 with a 4x objective, compared to C4 and C11 respectively without stitching (i.e. one FoV). However, just to relax to speed requirements and other elements that were not taken into account (such as areas of image overlap, precision issues, etc.), the limit will be more likely at C15 when using the 4x objective.

Table SI- 1

Objective	**NA**	**Resolution (μm)**	**FoV (μm)**	**Cardinality 1 FoV**	**Stitching limits**	**Cardinality Stitched FoV**
2x	0.3	5	15000	11	4 x 3	21
4x	0.16	3	10000	9	6 x 4	20
10x	0.4	2	2000	4	14 x 11	15
20x	0.75	2	1000	3	24 x 18	14
40x	0.96	2	500		27 x 20	11
60x	1.35	1	200		32 x 24	8
100x	1.4	0.5	60		39 x 30	5

SI-4 Method 1: Sub-set Sum Problem calculation by various electronic chips

A computer program to solve the Subset Sum Problem was developed in the C programming language to enable low-level memory access, efficient mapping to machine instructions and flexibility. Out of several algorithms to solve the SSP, a naïve approach was adopted to emulate the Biological Chip logic and operations. Therefore, it was possible to establish a comparison between electronic and biological devices in terms of performance times with increasing size of the problem (Cardinality).

The SSP algorithm was designed to recursively explore all N-elements in the set and to log the k winning combinations according to the N chooses k subsets $\sum_{0\leq k\leq N}\binom{N}{k} = 2^N$. Its running time falls in the order $O(2^N N)$ due to the SSP combinatorial nature and the highest computed sum of N-elements to identify subsets. Similarly, the bio-device was designed to find all possible k subsets sums by exploring 2^Npossible combinations, times the N-elements considered for solution identification.

The algorithmic naïve approach, combined to its lack of any pre-existing knowledge and bias of problem clauses, enabled us to benchmark the Biological chip performance analysis with performances of historical Intel processors. Being RAM Memory and Clock Speed the major factors affecting CPU speed, we replicated computing resources of Intel382 SX and DX, Intel486 DX, Intel DX2, Intel Pentium and Intel Core Quad 2Processor versions by simulating part of their computer hardware with Virtual Machines. Thus, we were able to scale down RAM and clock speed and record performance times on SSP.

SI-5 Method 2: Simulation of Sub-set Sum Problem calculation by a hypothetical biological chip

The performance of electronic computers was compared to hypothetical bio-computer performance simulated for the following agents: *E. coli*, *V. natriegens*, *M. janaschii*, and *M. villosus*, as well as Microtubules and Actin filaments. For including cell division in the simulation of the bacterial species used, the following doubling times have been used: *M. janaschii*: 74 min., *M. villosus*: 45 min., *E. coli*: 30 min., and *V. natriegens*: 15 min.

The independent and parallel behaviour of a large number of self-propelled biological agents was reproduced and scaled taking into consideration various factors: network and channels geometry varying with increasing SSP size, agents' random walk behaviour over network exploration and physical rules of motion and logic given by network structures and agents sizes, speed and characteristic behaviour.

SI-6 Electronic reference device

By 'reverse engineering' starting from the pass- and split junctions for the sub-set sum bio-computation device described in this paper, an electronic equivalent device was designed. Originally, an analog circuit was conceived, but simulations showed that already with a limited number of nodes, leakage currents would render the readout of such a device unreliable. Hence, a digital electronic demo device (shown in Figure SI-2) with LED readout and IC HEF4019BP containing AND and OR gates (shown in Figure SI-3) was constructed; this particular DEMO device has a total sum of 6 only, but it has flexible nodes, which can be switched from pass- to split junctions by activating bus lines.

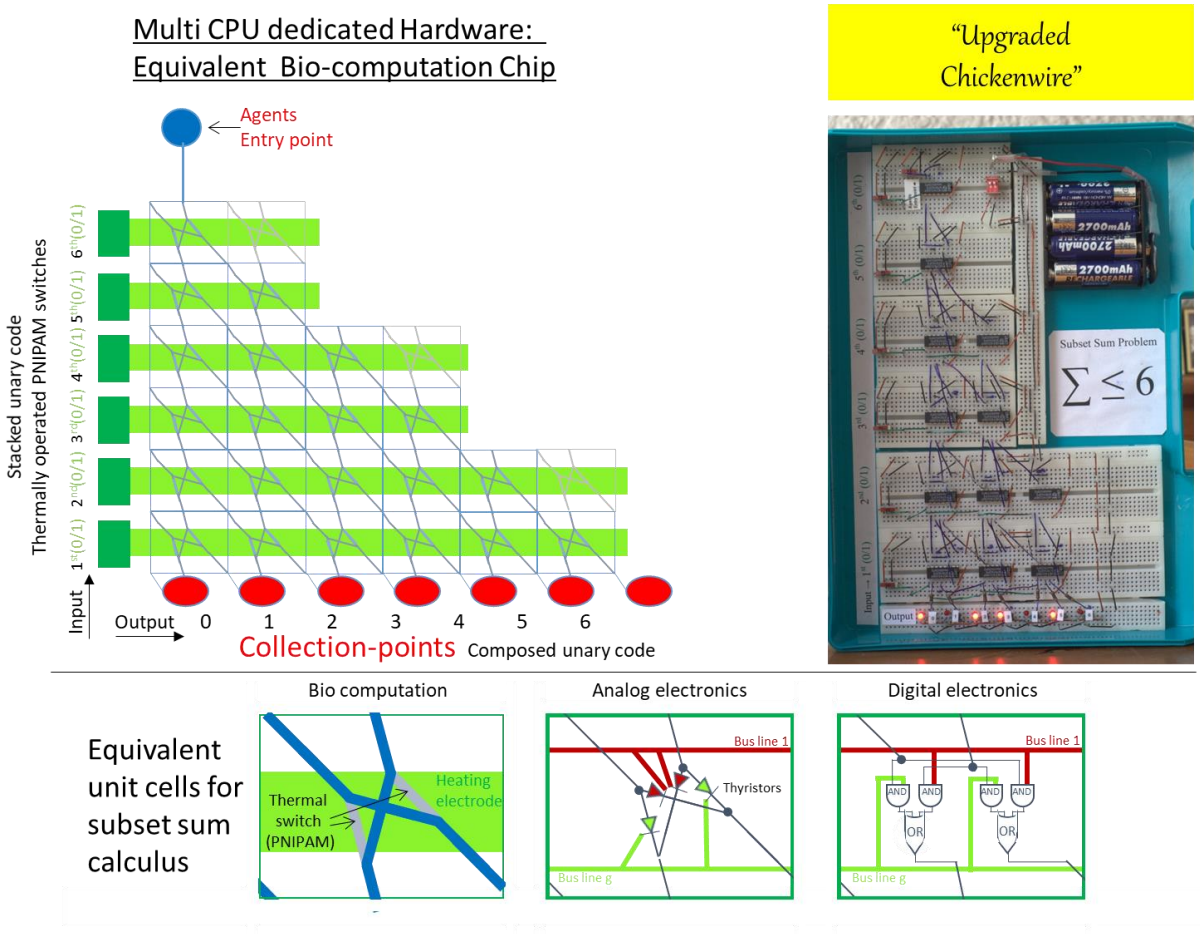
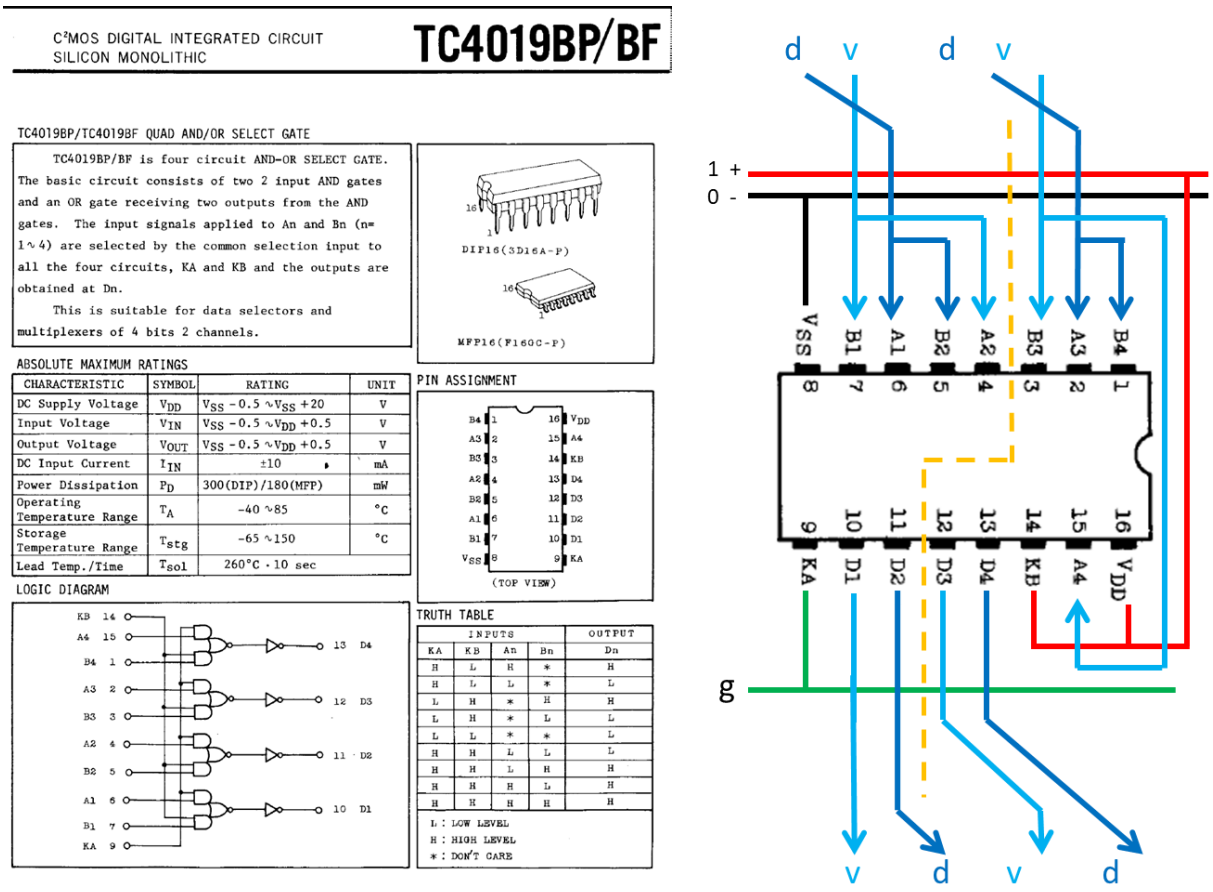


Figure SI- 2: Electronic demo device - Operation Upgraded CHicken wire (OUCH!)



C²MOS DIGITAL INTEGRATED CIRCUIT
SILICON MONOLITHIC

TC4019BP/BF

TC4019BP/TC4019BF QUAD AND/OR SELECT GATE

TC4019BP/BF is four circuit AND-OR SELECT GATE. The basic circuit consists of two 2 input AND gates and an OR gate receiving two outputs from the AND gates. The input signals applied to An and Bn (n= 1∿4) are selected by the common selection input to all the four circuits, KA and KB and the outputs are obtained at Dn.

This is suitable for data selectors and multiplexers of 4 bits 2 channels.

DIP16(3D16A-P)

MFP16(F16GC-P)

ABSOLUTE MAXIMUM RATINGS

CHARACTERISTIC	SYMBOL	RATING	UNIT
DC Supply Voltage	V_{DD}	V_{SS} -0.5 ∿ V_{SS} +20	V
Input Voltage	V_{IN}	V_{SS} -0.5 ∿ V_{DD} +0.5	V
Output Voltage	V_{OUT}	V_{SS} -0.5 ∿ V_{DD} +0.5	V
DC Input Current	I_{IN}	±10	mA
Power Dissipation	P_D	300(DIP)/180(MFP)	mW
Operating Temperature Range	T_A	-40 ∿ 85	°C
Storage Temperature Range	T_{stg}	-65 ∿ 150	°C
Lead Temp./Time	T_{sol}	260°C · 10 sec	

PIN ASSIGNMENT

(TOP VIEW)

LOGIC DIAGRAM

TRUTH TABLE

INPUTS				OUTPUT
KA	KB	An	Bn	Dn
H	L	H	*	H
H	L	L	*	L
L	H	*	H	H
L	H	*	L	L
L	L	*	*	L
H	H	L	L	L
H	H	L	H	H
H	H	H	L	H
H	H	H	H	H

L : LOW LEVEL
H : HIGH LEVEL
* : DON'T CARE

Figure SI- 3: IC HEF4019BP,

Note that this IC can accommodate
two pass- or two split junctions simultaneously

Adapted from [1]

References

1. Datasheet4U. *TC4019BP Datasheet*. [cited 2019; Available from: https://datasheet4u.com/datasheet-pdf/Toshiba/TC4019BP/pdf.php?id=530848.

References

1. Ogban, F., I. Arikpo, and I. Eteng, *Von Neumann Architecture and Modern Computers.* Global Journal of Mathematical Sciences Vol. 6, No. 2, 2007, Page 97: ISSN 1596-6208 Indexed and abstracted on AJOL (UK): http://www.ajol.info, 2007. **Vol. 6, No. 2, 2007, Page 97**.
2. Baba, A., *Von-Neumann Architecture Vs Harvard Architecture*. 2019.
3. Patterson, D., Hennessy, J, *Computer Organization and Design, 2nd ed*. 1998, San Francisco, CA: Morgan Kaufmann.
4. Filsinger, M.D., *Understanding CPU Pipelining Through Simulation Programming*, A.S.f.E. Education, Editor. 2005, University of Cincinnati.
5. Esmaeilzadeh, H., et al., *Power challenges may end the multicore era.* Communications of the ACM, 2013. **56**(2): p. 93.
6. Shen, H. and F. Pétrot, *Using Amdahl's Law for Performance Analysis of Many-Core SoC Architectures Based on Functionally Asymmetric Processors*. Vol. 6566. 2011. 38-49.
7. *International Technology Roadmap for Semiconductors (ITRS).* 2013.
8. Dennard, R.H., et al., *Design of ion-implanted MOSFET's with very small physical dimensions.* IEEE journal of solid-state circuits, 1974. **9**(5): p. 256-268.
9. Markov, I.L., *Limits on fundamental limits to computation.* Nature, 2014. **512**(7513): p. 147-54.
10. Sipser, M., *Introduction to the theory of computation*. Third edition. ed. 2013, Boston, MA: Cengage Learning.
11. Turing, A. and B.J. Copeland, *The essential Turing : seminal writings in computing, logic, philosophy, artificial intelligence, and artificial life, plus the secrets of Enigma*. 2004, Clarendon Press ; Oxford University Press: OxfordNew York.
12. Cook, S.A., *The complexity of theorem-proving procedures,* in *Proceedings of the third annual ACM symposium on Theory of computing*. 1971, Association for Computing Machinery: Shaker Heights, Ohio, USA. p. 151–158.

13. Delft, F.C.M.J.M.v., et al., *Something has to give: scaling combinatorial computing by biological agents exploring physical networks encoding NP-complete problems.* Interface Focus, 2018. **8**(6): p. 20180034.
14. Anderson, S., *The Limits of Quantum Computers.* Scientific Amarican Inc., 2008.
15. Adleman, L.M., *Molecular Computation of Solutions to Combinatorial Problems.* Science, 1994. **266**(5187): p. 1021-1024.
16. Rozen, D.E., S. McGrew, and A.D. Ellington, *Molecular computing: Does DNA compute?* Current Biology, 1996. **6**(3): p. 254-257.
17. Feynman, R.P., *Theres plenty of room at the bottom.* RESONANCE -BANGALORE-, 2011. **16**(9): p. 890-905.
18. Rahman, M. and M. Kaykobad, *On Hamiltonian cycles and Hamiltonian paths.* Information Processing Letters, 2005. **94**: p. 37-41.
19. Parker, J., *Computing with DNA.* EMBO reports, 2003. **4**(1): p. 7-10.
20. Bridget, M., et al., *A Survey Of Dna Computing.* 2013. **1**: p. 2321-9467.
21. Zhang, C., et al., *DNA Computing for Combinational Logic*. 2018.
22. Shor, P.W., *Algorithms for Quantum Computation: Discrete Logarithms and Factoring.* Annual Symposium on Foundations of Computer Science [papers]. 1994. **35**: p. 124.
23. Gordon, D.M., *Discrete Logarithms in $GF (P)$ Using the Number Field Sieve.* SIAM Journal on Discrete Mathematics, 1993. **6**(1): p. 124-138.
24. Lenstra, H.W. and P. Carl *A rigorous time bound for factoring integers*. Journal of the American Mathematical Society, 1992. **5**, 483-516 DOI: 10.1090/S0894-0347-1992-1137100-0.
25. Guo, H., *Lecture 10: NP-intermediate candidates*, in *INFR11102: Computational Complexity*. 2019.
26. Rivest, R.L., A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems.* Commun. ACM, 1978. **21**(2): p. 120–126.
27. Montanaro, A., *Quantum algorithms: an overview.* npj Quantum Information, 2016. **2**(1): p. 15023.
28. Amico, M., Z.H. Saleem, and M. Kumph, *Experimental study of Shor's factoring algorithm using the IBM Q Experience.* Physical Review A, 2019. **100**(1): p. 012305.

29. Grover, L.K., *A fast quantum mechanical algorithm for database search*, in *Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing*. 1996, Association for Computing Machinery: Philadelphia, Pennsylvania, USA. p. 212–219.
30. Chiu, D., et al., *Using three-dimensional microfluidic networks for solving computationally hard problems.* Proceedings of the National Academy of Sciences of the United States of America, 2001. **98**: p. 2961-6.
31. Pardalos, P. and J. Xue, *The Maximum Clique Problem.* Journal of Global Optimization, 1994. **4**: p. 301-328.
32. Bajić, V.B. and T.W. Tan, *Information processing and living systems*. 2005, Imperial College Press ; Distributed by World Scientific Pub.: London Singapore ;.
33. Newton, I., A. Motte, and F. Cajori, *Sir Isaac Newton's Mathematical principles of natural philosophy, and his System of the world*. 1962, Berkeley: University of California Press.
34. Hedayatpour, S., N. Kama, and S. Chuprat, *A Sticker-Based Model Using DNA Computing for Generating Real Random Numbers.* International Journal of Security and Its Applications, 2014. **8**: p. 113-122.
35. Al-Bakri, S., et al., *Securing peer-to-peer mobile communications using public key cryptography: New security strategy.* International Journal of the Physical Sciences, 2011. **6**: p. 930-938.
36. Johnston, D., *Random Number Generators—Principles and Practices: A Guide for Engineers and Programmers*. 2018: De Gruyter.
37. Gearheart, C.M., B. Arazi, and E.C. Rouchka, *DNA-based random number generation in security circuitry.* Biosystems, 2010. **100**(3): p. 208-14.
38. Gong, L., et al., *True Random Number Generators Using Electrical Noise.* IEEE Access, 2019. **PP**: p. 1-1.
39. Stipčević, M. and Ç. Koç, *True Random Number Generators*. 2014. p. 275-315.
40. Beenakker, C.W.J. and M. Büttiker, *Suppression of shot noise in metallic diffusive conductors.* Physical Review B, 1992. **46**(3): p. 1889-1892.
41. Pironio, S., et al., *Random numbers certified by Bell's theorem.* Nature, 2010. **464**(7291): p. 1021-1024.
42. Bird, J.J., A. Ekárt, and D.R. Faria, *On the effects of pseudorandom and quantum-random number generators in soft computing.* Soft Computing, 2020. **24**(12): p. 9243-9256.

43. Fürst, H., et al., *High speed optical quantum random number generation.* Optics Express, 2010. **18**(12): p. 13029-13037.
44. Gabriel, C., et al., *A generator for unique quantum random numbers based on vacuum states.* Nature Photonics, 2010. **4**(10): p. 711-715.
45. Guo, H., et al., *Truly random number generation based on measurement of phase noise of a laser.* Phys Rev E Stat Nonlin Soft Matter Phys, 2010. **81**(5 Pt 1): p. 051137.
46. Qi, B., et al., *High-speed quantum random number generation by measuring phase noise of a single-mode laser.* Optics letters, 2010. **35**: p. 312-4.
47. Collins, M.J., et al. *Quantum random number generation using spontaneous raman scattering*. in *2014 Conference on Lasers and Electro-Optics (CLEO) - Laser Science to Photonic Applications*. 2014.
48. Stipčević, M. and B. Rogina, *Quantum random number generator based on photonic emission in semiconductors.* The Review of scientific instruments, 2007. **78**: p. 045104.
49. Wayne, M.A., et al., *Photon arrival time quantum random number generation.* Journal of Modern Optics, 2009. **56**(4): p. 516-522.
50. Jennewein, T., et al., *A Fast and Compact Quantum Random Number Generator.* Review of Scientific Instruments, 2000. **71**: p. 1675-1680.
51. Herrero-Collantes, M. and J.C. Garcia-Escartin, *Quantum Random Number Generators.* Reviews of Modern Physics, 2016. **89**.
52. Nicolau, D.V., et al., *Parallel computation with molecular-motor-propelled agents in nanofabricated networks.* Proceedings of the National Academy of Sciences, 2016. **113**(10): p. 2591.
53. Flajolet, P., D.l. Gardy, and L.s. Thimonier, *Birthday paradox, coupon collectors, caching algorithms and self-organizing search.* Discrete Applied Mathematics, 1992. **39**(3): p. 207-229.
54. Walker, S., *PCem*. 2007.
55. Valgrind Developers, *Valgrind*. 2012.
56. Dyakonov, M., *The Case Against Quantum Computing*, in *IEEE Spectrum*. 2018.
57. Vazquez, A., et al., *Bacterial Cellulose from Simple and Low Cost Production Media by Gluconacetobacter xylinus.* Journal of Polymers and the Environment, 2013. **21**(2): p. 545-554.

58. Giles M., *Explainer: what is a quantum computer?*, in *MIT Technology Review*. 2019.
59. Penrose, R., *The emperor's new mind : concerning computers, minds, and the laws of physics*. 1991, New York, N.Y., U.S.A.: Penguin Books.
60. Clarke, J., *An Optimist's View of the 4 Challenges to Quantum Computing*, in *IEEE Spectrum*. 2019.
61. Lipton, R.J. and K.W. Regan, *Quantum algorithms via linear algebra : a primer*. 2014, The MIT Press: Cambridge, Massachusetts.
62. Grover, L.K., *From Schrodinger's equation to the quantum search algorithm.* AMERICAN JOURNAL OF PHYSICS, 2001. **69**(Part 7): p. 769-777.
63. Lund, A.H., M. Duch, and F.S. Pedersen, *Increased cloning efficiency by temperature-cycle ligation.* Nucleic acids research, 1996. **24**(4): p. 800-801.
64. Kresge, N., R.D. Simoni, and R.L. Hill, *Insights into DNA Joining: I. Robert Lehman's Work on DNA Ligase.* The Journal of biological chemistry., 2007. **282**(2): p. e1.
65. Suzuki, M., et al., *Efficient DNA ligation by selective heating of DNA ligase with a radio frequency alternating magnetic field.* Biochemistry and biophysics reports, 2016. **8**: p. 360-364.
66. McCloskey, S., *Complexity & Computability*.
67. Butt, S., *Halting problem.* Journal of Problem Solving, 2016. **01**: p. 06.
68. Toseland, C.P., *Fluorescent labeling and modification of proteins.* Journal of chemical biology, 2013. **6**(3): p. 85-95.
69. Thorn, K., *Genetically encoded fluorescent tags.* 2017. **28**(7): p. 848-857.
70. Einarsson, J., *New biological device not faster than regular computer.* 2016. **113**(23): p. E3187-E3187.
71. Pisinger, D., *Linear Time Algorithms for Knapsack Problems with Bounded Weights.* Journal of Algorithms, 1999. **33**(1): p. 1-14.
72. Walt, S.v.d., S.C. Colbert, and G. Varoquaux, *The NumPy Array: A Structure for Efficient Numerical Computation.* Computing in Science & Engineering, 2011. **13**(2): p. 22-30.
73. Turan, M.S., Barker E., Kelsey J., McKay K. A., Baish M. L., Boyle M., National Institute of Standards and Technology (NIST), *SP 800-90B Recommendation for the Entropy Sources Used for Random Bit Generation.* 2018.