Design of network-based biocomputation circuits for the exact cover problem

Till Korten¹, Stefan Diez¹,², Heiner Linke³, Dan V Nicolau Jr⁴ and Hillel Kugler⁵,∗

¹ B CUBE - Center for Molecular Bioengineering, Technische Universität Dresden, D-01307 Dresden, Germany
² Max Planck Institute of Molecular Cell Biology and Genetics, D-01307 Dresden, Germany
³ NanoLund and Solid State Physics, Lund University, Box 118, S-22100 Lund, Sweden
⁴ School of Mathematical Sciences, and ARC Centre of Excellence for Mathematical and Statistical Frontiers, Queensland University of Technology, Gardens Point, Brisbane, Qld, Australia
⁵ Faculty of Engineering, Bar-Ilan University, Ramat Gan, Israel

∗ Author to whom any correspondence should be addressed.
E-mail: hillelk@biu.ac.il

Keywords: network based biocomputation, NP-complete problems, exact cover, biological computation

Abstract

Exact cover is a non-deterministic polynomial time (NP)—complete problem that is central to optimization challenges such as airline fleet planning and allocation of cloud computing resources. Solving exact cover requires the exploration of a solution space that increases exponentially with cardinality. Hence, it is time- and energy consuming to solve large instances of exact cover by serial computers. One approach to address these challenges is to utilize the inherent parallelism and high energy efficiency of biological systems in a network-based biocomputation (NBC) device. NBC is a parallel computing paradigm in which a given combinatorial problem is encoded into a graphical, modular network that is embedded in a nanofabricated planar device. The network is then explored in parallel using a large number of biological agents, such as molecular-motor-propelled protein filaments. The answer to the combinatorial problem can then be inferred by measuring the positions through which the agents exit the network. Here, we (i) show how exact cover can be encoded and solved in an NBC device, (ii) define a formalization that allows to prove the correctness of our approach and provides a mathematical basis for further studying NBC, and (iii) demonstrate various optimizations that significantly improve the computing performance of NBC.

This work lays the ground for fabricating and scaling NBC devices to solve significantly larger combinatorial problems than have been demonstrated so far.

1. Introduction

Many combinatorial problems of practical importance require exploring a large number of possible candidate solutions to check for the existence of a correct solution and—if it exists—to find it. One example is exact cover, which needs to be solved for example when scheduling resources, such as airline fleet planning [1] and allocation of cloud computing resources [2]. Exact cover has been shown to be NP-complete, meaning that it is a difficult problem for which all currently existing algorithms scale exponentially with the cardinality of the problem [3]. In some cases, heuristics can be used to try to solve the problem for a subset of the instance space, using some knowledge about the structure of certain types of instances [4, 5]. However, these approaches cannot provide any guarantees on the runtime performance, and there will always be instances that require very long runtimes. Another approach is to relax the problem and look for approximate solutions [6], with no guarantee to find the optimal solution. An approximate heuristic solution has been demonstrated to have a quantum advantage on a noisy quantum computer [7, 8]. But this solution still suffers from potentially long run times and approximate solutions.

Here, we focus on a computing strategy that explores the entire solution space utilizing an energy-efficient, parallel-computation approach to solve exact cover. Whereas such approaches still require exponential computing resources (energy, memory, computing cores), they do not necessarily require
Figure 1. Explanation of a previously demonstrated NBC device solving a small instance of the subset sum problem, which asks whether a target sum $T$ can be reached by summing up any combination of integers from set $S$. For example, given the set $S = \{2, 5, 9\}$ and the target $T = 7$ we would sum up $2 + 5$, while for $T = 13$ there is no solution.

exponential time, because the computing is performed in parallel. As an added benefit, individual computing steps can be performed relatively slowly, making it possible to use vastly less energy in total compared to serial computing at the necessary high frequency [9]. Specifically, we use network-based biocomputation (NBC), an approach by which a mathematical problem is encoded into a graphical, modular network embedded in a nanofabricated planar device [10]. Exploring the network in a parallel fashion using a large number of independent, biological agents, such as molecular-motor-propelled protein filaments, can then solve the mathematical problem, as demonstrated for subset sum [11] (see figure 1 for an explanation of the subset sum network algorithm). This approach, using molecular motors, has the critical advantages that it uses orders of magnitude less energy per operation than conventional computers and that it has the potential to be scalable with presently available technology. Particularly, the protein filaments that act as both computing core and memory are self-assembling and cheap to produce in large numbers.

Figure 1 shows a network encoding the subset sum instance $\{2, 5, 9\}$. Agents enter the network from the top-left corner. Cyan circles represent split-junctions where it is equally probable that agents continue straight ahead or turn. Unmarked junctions represent pass-junctions where agents continue straight ahead. Moving diagonally down at a split-junction corresponds to adding that integer (numbers 2 and 9 for the purple example path). The actual value of the integer potentially added at a split-junction is determined by the number of rows of junctions until the next split-junction. 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 labeled in green, and incorrect results (where no agents will arrive) are labeled in magenta.

Here we report on the encoding of exact cover into a network format suitable for solving the problem by NBC. Given a collection $S$ of subsets, each containing elements of a target set $X$, exact cover asks whether a subcollection $S^*$ of $S$ exists such that each element in $X$ is contained in exactly one subset in $S^*$. In other words, exact cover has a solution when all subsets in $S^*$ are (i) pairwise disjoint, (i.e. $\forall S_i, S_j \in S^*: S_i \cap S_j = \emptyset$) and (ii) yield $X$ when joined together. In the following, we will consider exact cover as a decision problem, meaning we only want to know if there exists a solution or not. This information can be used to find the exact solution by solving a partially fixed variation of the decision problem $|S|$ times, each time enforcing the use of one set in the solution collection. By checking each fixed variation for whether it has a solution or not, the exact solution can be identified with only linear increase in computing effort [3]. For example, given the target set $X = \{1, 2, 3, 4\}$ and the collection of subsets $S = \{[3, 4], [2, 4], [2, 3], [1, 3]\}$, the subcollection $S^* = \{[1, 3], [2, 4]\}$ would be a solution. Hence, this instance has a solution. On the other hand, the subcollection $S^{**} = \{[3, 4], [2, 4]\}$ would not be a solution because ‘1’ is missing and ‘4’ is in two subsets. A more practical example would be an outgoing airplane that needs $X = \{captain, first officer, purser, cabin crew\}$. The corresponding collection of subsets $S$ would be formed by several incoming airplanes that each have their own varying crew combinations that need to be matched to $X$. 
Figure 2. Encoding exact cover into network format. (A) Schematic rendering of a biocomputation network where kinesin-1 motors coat the bottom of 300 nm wide channels etched in silicon oxide. The channels are explored by microtubules propelled by kinesin-1 motors. (B) Binary encoding principle: elements of the target set $X$ as well as each subset in $S$ are mapped to bits of a binary number. Here, 1 is mapped to the most significant bit and 4 is mapped to the least significant bit. Two examples of exact cover instances ($X_1$, $S_1$ and $X_2$, $S_2$) are given. $X_1$, $S_1$ has a solution (highlighted in green) and $X_2$, $S_2$ does not. (C) Example network block that encodes the subset 0011. Agents arriving at input 0100 (blue) encounter a split junction (cyan rectangle) that allows the agents to choose between the path straight down—not combining the sets 0100 and 0011—and the diagonal path that combines the sets to leave at output 0111. The same is true for input 0000, where agents leave at outputs 0000 or 0011. Because input subsets containing identical elements shall not be combined (as this would violate the rules of exact cover), all inputs that contain elements already present in the encoded subset start with a reset junction (red rectangle). This junction forces the agents to take the path straight down, i.e. not combining the input subset with the encoded subset. For example an agent entering at input 0001 (red rectangle) can only leave at output 0001. The same is true for inputs 0010 and 0011 which would force agents to leave at outputs 0010 and 0011, respectively. The input and output rows are separated by two rows of pass junctions (example denoted by blue rectangle) that force agents to remain on their chosen paths (diagonal or straight down). The number of pass junction rows is always one fewer than the binary value of the encoded subset. Cyan circles mark positions of split junctions, red rectangles mark positions of reset junctions. (D) Scanning electron micrographs of examples for each junction type. The color of the frame corresponds to the rectangles in C.

In this paper, we will describe an algorithm that takes an instance of exact cover (i.e. a given target set $X$ and a collection of subsets $S$) as input and creates an NBC circuit which allows to solve the underlying decision problem by counting the number of agents leaving the network at predefined exits. Moreover, we will provide the formal semantics for a computational model for capturing the networks used to solve the exact cover problem, that is suitable for studying the computational power of the NBC paradigm and forms the basis for applying formal verification algorithms to prove the logical correctness of NBC circuits.

2. Results

The aim of the network algorithm is to enable agents exploring the network (figure 2(A)) to find the solution by randomly choosing all possible subcollections of $S$ and checking whether any subcollection exactly covers $X$. The result for an exact cover instance encoded by a particular network is then given by whether or not agents arrive at the exit that corresponds to the target set $X$. To achieve this, our algorithm for converting exact cover into network format performs the following steps:
Figure 3. Network representation of exact cover problems. Networks encoding the example exact cover instances \(X_1, S_1\) and \(X_2, S_2\) introduced in figure 2. \(X_1, S_1\) has a solution (A) while \(X_2, S_2\) does not (B). Each network calculates all possible combinations of the subsets in \(S_1\) (A) and \(S_2\) (B), respectively. (A), (B) Blue arrows indicate network entrances. Green arrows highlight the target exit that indicates whether there exists an exact cover or not. Blue lines indicate the legal paths that agents can take through the networks. Cyan circles mark positions of split junctions, red rectangles mark positions of reset junctions.

(a) Map the elements of the given target set \(X\) to bits of a binary number (figure 2(B), left panel). This mapping can be done independently of the values of the elements in \(X\), that is, the order in the binary representation does not need to follow the value order of the members.

(b) Convert the given collection of subsets \(S\) to a set of binary numbers \(a\) using the same mapping rule, i.e. a bit is set to 1 if the respective element in \(X\) is a member of the subset and 0 if the respective element in \(X\) is not a member in the subset (figure 2(B), right panel).

(c) Create network blocks that enable agents entering the block to randomly choose or disregard a subset from \(S\), following the rules defined by exact cover (figure 2(C) including further explanation in the figure caption). Each network block consists of three types of junctions (figure 2(D)): (i) split junctions allow computing agents to randomly choose or disregard subsets from \(S\). (ii) Reset junctions prevent agents from choosing subsets that contain identical elements [12]. (iii) Pass junctions force the agents to continue on a diagonal path, incrementing the value of the binary number and thus defining which elements are contained in each subset.

(d) Combine the network blocks that represent all subsets from \(S\) into a network that enables agents to randomly choose all possible subcollections. If a path exists from an entrance of the network to the binary number \(2^{|X|} - 1\) (i.e. all bits are set to one), then the exact cover instance does have a solution, otherwise it does not. The (non-) existence of this path is found by many biomolecular agents that explore the network in parallel.

A detailed description of how the network algorithm works can be found in figure 2. Two examples of networks encoding an instance that has a solution and one that does not are shown in figure 3.

Because the network is explored stochastically, it should be noted that the answer is always probabilistic, i.e. it certainty depends on the problem size, the number of agents and the probabilities of making choices at the split junctions [13]. For example, if the split junctions split 50/50 and there are no errors at pass junctions, the examples given in figure 3 would require approximately 64 agents to give the correct answer with 95% confidence. If the agents can make wrong turns at pass junctions, the certainty of the answer depends additionally on the associated error rate and on the number of traversed pass junctions [11].

3. Network optimization

A main factor determining the speed and cost of the calculation is the physical size of the network. In this section we will introduce several optimization steps that each reduce the size of the network that needs to be built, particularly the number of rows. All of these optimizations improve the performance of the NBC devices by reducing the time, energy and number of agents needed for exploring the network and solving the respective exact cover instance.

**Bit-mapping optimization.** The mapping between the elements in the problem sets and the binary numbers is arbitrary from a mathematical point of view, but affects the value of the binary number, which
Figure 4. Network size and performance optimization. (A) Network without optimizations. (B) Network after optimising element-to-bit mapping. Elements in $X$ were sorted by their frequency in $S$ in order to minimize the values of the binary numbers after element-to-bit mapping. (C) Network where numbers corresponding to sets containing the most frequent element are used as entrances. (D) Network where the most frequently appearing element is mapped to the highest value bit and used as entrances. Remaining sets are optimized as described in (B). (E) Entrances that are further from the target exit than the total sum of the numbers in the network (i.e. from which the target exit cannot be reached) are removed. (A)–(E) Blue arrows indicate network entrances. Green arrows highlight the target exit that indicates whether there exists exact cover or not. Cyan circles mark positions of split junctions, red rectangles mark positions of reset junctions.
in turn determines the physical size of the resulting network. Therefore, we can reduce the size of the network (and thus the calculation time and energy) by optimising the mapping between bits and elements in $X$ (figure 4(B)): we ordered the elements in $X$ by how frequently they appear in the subsets in $S$. The least frequent elements were assigned to the highest value bits (i.e. bits resulting in large decimal numbers) in the binary number and the most frequent elements were assigned to the lowest value bits. Figure 3 shows an example where this bit-mapping optimization reduced the size of the network from 48 rows to 26 rows (compare figure 4(A) and (B)), decreasing the time and energy needed to solve the network by approximately 50%.

**Entrance optimization.** Another factor determining the time needed to solve an exact cover instance is the number of filaments that can be fed into the network simultaneously. Therefore, we devised a method...
to increase the number of entrances into the network. We begin by noting that the main purpose of the network is to calculate combinations of subsets. These combinations are calculated at split junctions. Subsets that cannot be combined with each other (because they contain identical elements) are connected by reset junctions. If this happens at the start of the network, the respective subsets only share a split junction with the empty set \( 0 \) (this is the case for the sets 0001, 0101, 0011 and 1011 in figure 4(B)). Consequently, the result of the network does not change if we remove the network sections encoding these subsets and use the numbers corresponding to the respective subsets as entrances into the network (figure 4(C)). The resulting network has only six rows and five entrances. Thus this ‘entrance optimization’ reduces the time needed to solve an exact cover instance by roughly a factor of twenty (a quarter of the rows and five times the entrances).

**Entrance and bit-mapping optimization.** We can combine entrance optimization with bit mapping optimization: because the entrances do not contribute to the number of rows, we rearrange the bit mapping such that the most frequently appearing element in \( X \) is encoded not by the lowest—but the highest value bit. Thus, the corresponding large decimal numbers are used as entrances, further reducing the number of rows in the network to 3 (figure 4(D)). The optimal combination of bit-mapping and entrance optimization reduces the time needed to solve a network by a factor of 80 (5 entrances times 16/3 rows).

**Accessible entrance optimization.** The target exit can only be reached from entrances which have a value larger than the target exit minus the total sum of numbers encoded in the network. In the example given in figure 4, the cutoff is 1100. Any entrance that is further away from the target exit than this cutoff can be removed. In case of the example given in figure 4, this leaves us with a tiny network that has only three rows and three columns.

To better illustrate the power of our optimizations, we picked substantially more difficult problems with \( |S| = 9 \) and \( |X| = 6 \) (figure 5), one that has a solution (figure 5 left column) and one that does not (figure 5 right column). The optimizations reduced the two network sizes from 268 and 252 rows and columns to 25 rows and 24 columns, respectively (figures 5(A) and (B)). At the same time, the number of entrances increased from one to five. Overall, the time needed to solve these instances is reduced roughly by a factor of 50 (a tenth of the rows and five times the entrances). The correct paths (figure 5 blue lines) indicate that agents arrive at the target exit of the network that encodes an instance that has a solution (figure 5(A)) but no agents arrive at the exit of the network that encodes an instance that does not have a solution (figure 5(B)).

**Reverse exploration optimization.** Since we know the target exit, we can split the network up into a top and a bottom part, each encoding only half the total number of sets represented in the network (figures 5(C) and (D)). That way, the network is explored both from the entrances and—in reverse—from the exit. We termed this ‘reverse exploration’ optimization. In our current example, this means that the top network encodes the sets \( \{1, 5\} \) and \( \{6, 5, 4\} \) (corresponding to the binary numbers 0011001 and 0001111), while the bottom network encodes the sets \( \{5, 4\} \) and \( \{6, 4\} \) (corresponding to the binary numbers 0010100 and 0000111) (figures 5(C) and (D)). The top network still has the same entrances as the whole network, while the bottom network has only one entrance (corresponding to the target exit 1111111). This optimization reduces the total number of paths that need to (potentially) be explored from 80 possible paths \( 5 \times 2^4 \) to 24 possible paths \( 6 \times 2^2 \). However, it comes at the cost that more exits (16 instead of one) need to be monitored. The benefit of reduced agent number (and thus reduced time needed to feed agents into the network) scales exponentially with \( |S| \), while the cost of increased exit monitoring scales approximately linear. Furthermore, the reduced number of rows in each network will exponentially reduce the number of agents that make an error in case of imperfect pass junctions.

4. Discussion

We successfully encoded exact cover into a network format suitable for solving the problem by NBC. Exact cover is the second NP-complete problem successfully encoded into network format, demonstrating that NBC can solve a range of different problems.

The size of the networks (i.e. the number of rows) of the network encoding of exact cover presented here scales approximately linearly with the number of sets in \( S \) (see below for more detailed scaling considerations). However, it is important to note that the networks scale exponentially with the number of elements in \( X \). This is a consequence of the unary encoding of the binary numbers in \( a \) representing each subset in \( S \). For the placement of reset junctions, only split junctions (i.e. a fraction of all junctions in the network) need to be tested locally, indicating that the algorithm for placing reset junctions scales linearly with the size of the network.

We demonstrated how automated optimizations can be used to make the network design more efficient, while preserving the correctness of the circuit. Each optimization step (bit mapping, entrance, reverse...
exploration) reduces the network size (and the run time) for random problems on average by a factor of 2. The agent number is reduced exponentially (the problem size is effectively halved) by both entrance and reverse exploration optimizations.

Crucially, the algorithm that designs the network scales itself only polynomially, not exponentially, with problem size. It requires counting the frequency of each element of $X$ in $S$, which scales with $|X|^p |S|$ as well as sorting of the resulting counts, which scales with $|S| \log |S|$ (see [14]). Overall, the algorithm that encodes exact cover into optimized network format scales with $2^{|X| (|X| + |S|) \log |S|}$ for randomly generated instances.

The run time scales with $2^{|X||S|}$ and the number of agents scales with $2^{|S|}$. Thus, as long as $|S| > |X|$, the majority of the computation is done by the agents traversing the network. In this case, the computation will benefit from the fact that many agents can perform calculations in parallel. However, if $|X| > |S|$, the majority of the computation will be performed designing the network. In that case, it will be more efficient to solve the respective instance on an electronic computer.

The design of the exact cover network presented here is more complex than the design for the subset sum problem network introduced in [11], which raises a challenge of ensuring logical correctness of the exact cover network and new networks that will be designed in the future. In order to formally verify the correctness of our exact cover network encoding, we defined the formal semantics of NBC circuits for exact cover networks [15] using a transition system [16], which provides a mathematical model for the computation of the circuit. Transition systems are used as a basic computational model for defining dynamic behavior of both software and hardware systems. Constructing transition systems for an NBC circuit provides a formal definition of the computations enabled by this new paradigm, and forms the basis of a formal verification method and tool that allowed us to mathematically prove the correctness of exact cover circuit instances using model checking algorithms [15].

This work lays the ground for fabricating and scaling NBC networks solving exact cover instances that are significantly larger than the proof of concept instance [11] solved by NBC so far. Compared to the instance with three numbers and eight possible solutions solved in [11] (see also figure 1), the optimizations enable us to encode an instance with 9 subsets (512 possible solutions) with a network that is only $\sim 50\%$ larger (see also figure 5). This demonstrates that it is possible to implement optimized algorithms in NBC, extending the range of problems that can be solved in practice.

Acknowledgments

This project has received funding from the European Union’s Horizon 2020 research and Innovation programme under Grant Agreement No. 732482 (Bio4Comp)

Data availability statement

No new data were created or analysed in this study.

ORCID iDs

Stefan Diez https://orcid.org/0000-0002-0750-8515
Heiner Linke https://orcid.org/0000-0003-4451-4006
Hillel Kugler https://orcid.org/0000-0001-7924-5665

References

[1] Grönkvist M 2005 The tail assignment problem PhD thesis Chalmers University of Technology and Göteborg University
[2] Ba C 2016 An exact cover-based approach for service composition 2016 IEEE Int. Conf. Web Services (ICWS) pp 631–6
[3] Garey M R and Johnson D S 1979 Computers and Intractability: A Guide to the Theory of NP-Completeness 1st edn (New York: Freeman)
[4] Dong W, Xu Z-H, Liu X-H, Wang L-S-B and Wang L-S-B 2020 Irregular subarray tiling via heuristic iterative convex relaxation programming IEEE Trans. Antennas Propagat. 68 2842–52
[5] Hu R, Sha T, Van Kaick O, Deussen O and Huang H 2020 Data sampling in multi-view and multi-class scatterplots via set cover optimization IEEE Trans. Visual. Comput. Graphics 26 739–48
[6] Guruswami V and Lin P 2019 Parameterized inapproximability of exact cover and nearest codeword (arXiv:1905.06503v3)
[7] Bengtsson A et al 2020 Improved success probability with greater circuit depth for the quantum approximate optimization algorithm
[8] Vikstål P, Grönkvist M, Svensson M, Andersson M, Johansson G and Ferrini G 2020 Applying the quantum approximate optimization algorithm to the tail-assignment problem Phys. Rev. App. 14 054009
[9] Konopik M, Korten T, Lutz E and Linke H 2021 Fundamental energy cost of finite-time computing (arXiv:2101.07075[cond-mat])
[10] Nicolau D V, Nicolau D V, Solana G, Hanson K L, Filipponi L, Wang L and Lee A P 2006 Molecular motors-based micro- and nano-biocomputation devices Microelectron. Eng. 83 1582–8
[11] Nicolau D V et al 2016 Parallel computation with molecular-motor-propelled agents in nanofabricated networks Proc. Natl Acad. Sci. USA 113 2591–6
[12] Note 1. The placement of split- and reset-junctions is pre-calculated during network creation. However, because the calculations are done locally at each input junction (which is a small subset of all junctions in the network), the algorithm that places these junctions scales linearly with the size of the network (as detailed in the discussion).
[13] Flajolet P, Gardy D and Thimonier L 1992 Birthday paradox, coupon collectors, caching algorithms and self-organizing search Discr. Appl. Math. 39 207–29
[14] Hoare C A R 1961 Algorithm 64: quicksort Commun. ACM 4 321
[15] Aluf-Medina M, Korten T, Raviv A, Nicolau Jr D V and Kugler H 2021 Formal semantics and verification of network-based biocomputation circuits January 17-19, 2021 Copenhagen, Denmark Verification, Model Checking, and Abstract Interpretation 12597 pp 464–85
[16] Manna Z and Pnueli A 2012 The Temporal Logic of Reactive and Concurrent Systems: Specification (Berlin: Springer)