Computational Requirements for Nano-Machines

Melanie Badura
Universität zu Lübeck
Lübeck
melanie.badura@student.uni-luebeck.de

ABSTRACT

Researchers are studying many factors to develop nanonetworks. So far, they omit computational requirements. The premise of this paper is to define the complexity of scenarios done at a nanoscale. We mainly use scenarios from the medical field. Out of these scenarios, we filter problems, which sort into the complexity classes AC^0, NC^1, L.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; *Sensor networks*; • **Networks** → Network reliability.

KEYWORDS

nanonetworks; computational complexity; space-complexity

ACM Reference Format:
Melanie Badura. 2021. Computational Requirements for Nano-Machines. In *Proceedings of AzuNet Seminar (CR for Nano'21)*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

For years, researchers are talking about using nano-machines to create solutions in medicine and other areas. Such a machine should be able to communicate and sense/act. Thus, computational power is also a big issue. Because nano-machines are small, one question is how to implement these capabilities. While many researchers already deal with communication technology [1], they usually leave out computational capabilities. In this paper, we attempt to provide a general analysis of the computational capability of nano-machines. Since the capabilities of nano-machines vary widely, from nanoparticles with no computational capabilities to microprocessors [2], nano-machines divide into three groups according to complexity theory. We define the three groups by analyzing the tasks that nano-machines handle.

CR for Nano'21, July 2021, Lübeck, Germany

ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

2 PROBLEMS AND THEIR COMPLEXITY CLASSES

We can apply nano-machines to many scenarios. A few examples are detecting or treating cancer cells and supporting the human immune system. Those scenarios need a specific amount of computational power.

Most scenarios, and therefore most problems a nano-machine has to solve, need basic arithmetic. The addition ADD, subtraction SUB, multiplication MULT, AND/OR build the basis for value aggregation and routing schemes. We compare values with SIGN, EVEN/ODD, THRES to guide conditional behavior and support sensing strategies.

Pattern matching REG_α can detect antibodys and PARITY verifies message integrity.

Protocols, such as forwarding and routing, solve the communication problem. A nano-machine can handle several types of messages by solving these protocols in different ways. Communication is a perfect example of the use of basic arithmetics, pattern matching, storage needs. They need memory for the storage of routing information and for other values with which they must work.

Nanobots should also be able to perform more complex operations, like implementing a neural network. They use graph algorithms to search for differing sensor readings by depth-first DFS or breadth-first search BFS or to monitor the nanonetwork functionality with the reachability REACH.

Most nano-machines need memory. The size of the memory depends on the utility of the machine. For example, a nano-machine with a routing algorithm must have enough RAM to store multiple node labels LABEL. While nano-machines that observe the reaching of a drug threshold with THRES must be able to store at least two values.

All these problems divide into complexity classes. As the name suggests, the problems divide into classes that have approximately the same complexity. The reduction of operations chooses the complexity class. For example, subtract can be reduced to add. We do this by changing the sign of the second input number with a negation gate.

Table 1: Problems of nano-machines and their corresponding complexity class. An L-machine could implement all this problems.

Machine	Problems	
AC^0:	Add	Even/Odd
	Sub	Div_2
	Sign	Mod_2
	Inc	Log_2
	And/Or	Inv
NC^1:	Mult	Min/Max
	Div	Parity
	Exp	Reg_α
	Major	Mod_2
	Thres	Avg
L:	Label	Dfs
	Log mem	Bfs
	Reach	Median

Table 1 shows the three different classes and the problems they may solve. An L-machine can solve problems like a Turing machine. It also has a small amount of integrated memory. The class AC^0 describes boolean circuits with polynomial-size and a constant depth, whereas the class NC^1 may have a logarithmic depth and two inputs per gate.
Problems can lower their class if researchers can find new reductions for them. However, problems exist that need more powerful machines or global knowledge about the network than an L-machine can offer. Thus problems, like addressing, routing broadcasting/forwarding, could not be divided into the three complexity classes. They are therefore not included in Table 1.
Figure 1 shows the relationship between the complexity classes. A nano-machine that solves the problems of the highest complexity class can also use the operations of the lower classes. However, this does not work the other way around. An L-machine can do logarithmic calculations as well as multiplications. Though an NC^1-machine cannot do logarithmic calculations except it is Log_2.
The classification of the applications according to the computational power provides insights into the feasibility of Nanao-machines. Researchers may consult Table 1 to implement a protocol or an algorithm in a nano-machine. The algorithm must decompose into its basic operations. Afterward, the most complex operation gives the decision for a complexity class. Just because a nano-machine may perform only a few easy operations does not mean that it is simple

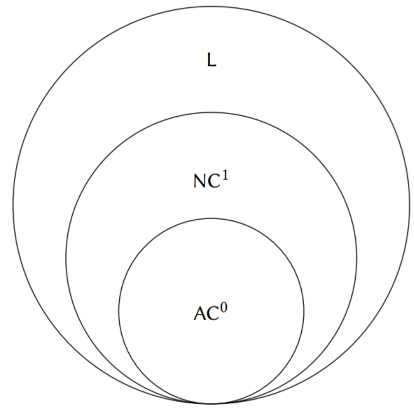

Figure 1: The relationship between the complexity classes. [3]

to build, but the choice of complexity class can help.
Considering the AC^0 class, if a nano-machine can add, it may do all other operations of the complexity class. Let us take the complexity class NC^1 for a practical example. A nano-machine observes the concentration of a marker with Avg in the human body. Since Thres is also in the same class, this nano-machine can also check if a defined value exceeds. We can do this without increasing the complexity nor the memory.

3 CONCLUSION

This paper tries to close the gap of the insufficient specified computational requirements. It does so by analyzing different scenarios. The analyzed problems divide into three complexity classes: AC^0, NC^1, L. A nano-machine can then solve all the problems of his class and the class problems under him.
However, additional components, such as storage, may mean that the classes do not represent the actual implementation of the nano-machines.
Furthermore, we have not addressed the combination of operations, so the power of the complexity class may exceed. In addition, the definition of extra complexity classes may specify the operations in more detail.

REFERENCES

[1] Ian F Akyildiz, Fernando Brunetti, and Cristina Blázquez. 2008. Nanonetworks: A new communication paradigm. *Computer Networks* 52, 12 (2008), 2260–2279.
[2] Luis C Cobo and Ian F Akyildiz. 2010. Bacteria-based communication in nanonetworks. *Nano Communication Networks* 1, 4 (2010), 244–256.
[3] Florian-Lennert Adrian Lau, Florian Büther, and Bennet Gerlach. 2017. Computational requirements for nano-machines: there is limited space at the bottom. In *Proceedings of the 4th ACM International Conference on Nanoscale Computing and Communication.* 1–6.