Parallel Virus Machines

David Orellana-Martín[1,2]

[1]Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, Avda. Reina Mercedes s/n, 41012, Universidad de Sevilla, Sevilla, Spain

[2]SCORE Laboratory, I3US, Avda. Reina Mercedes s/n, 41012, Sevilla, Spain

Natural Computing is a research field where different computing paradigms arise from the inspiration of processes occurring in Nature. DNA computing [1], Membrane computing [2], artificial neural networks [3], and evolutionary computing [4], among others, are widely studied while looking for alternative methods for solving real-life problems that demand a large amount of resources in more efficient ways.

In 2015, virus machines were introduced as a model of computation inspired by the way viruses spread between hosts and replicate their genetic code by "tricking" the host entities.

A basic virus machine of degree $(p, q), p, q \geq 1$, graphically depicted as in Fig. 1, is a tuple $\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_1, h_{out})$ where $\Gamma = \{v\}$ is the working alphabet, whose unique element is called a *virus*, $H = \{h_1, \ldots, h_p\}$ is the set of labels of the *hosts*, that will contain the viruses, $I = \{i_1, \ldots, i_q\}$ is the set of labels of the *instructions*, that will control the functioning of the system, D_H is the graph of the hosts, connecting them through *channels*, that will be opened by the instructions and will let viruses pass from one host to other one, D_I is the graph of the instructions, that will control the flow of the computation, G_C connects the instructions with the channels they will open, $n_1, \ldots, n_p$ are the initial number of viruses in each host, i_1 is the initial instruction and $h_{out} \in H \cup \{h_0\}$ is the output region, that can be either a host or the environment. If a virus passes through an open channel, then the next instruction will be the one connected to the current instruction by the edge with the higher weight; otherwise, the selected instruction will be connected by the edge with the lower weight. If no instructions are connected to the current instruction, the following instruction is denoted by # and the computation halts.

In previous works, it has been demonstrated that this model of computation is computationally complete; that is, its power is equivalent to the power of a Turing machine. Apart from that, it has been demonstrated to be a good model of computation for solving different types of

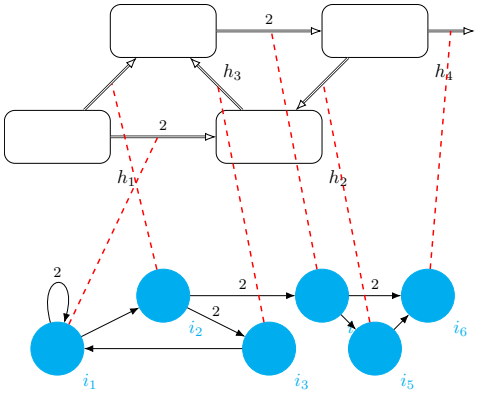

Figure 1: Basic virus machine

computational problems. The basic model is, by definition, a sequential model of computation. Thus, it is easy to see that from the computational complexity point of view, these devices will only be able to solve from the complexity class **P**.

Several bio-inspired models of computation have included different features to increase their computational efficiency, taking inspiration from some elements present in real-life processes. The most basic variant changes the initial instruction i_1 of the tuple by a set of initial instructions I_0, that will be executed at the same time. In this generalization, when two instructions open the same channel, both take the same decision concerning which path to take for selecting the next instruction. Since only one instruction can be selected from another instruction, the number of active instructions will decrease throughout the computation. The computation halts when the set of current active instructions is the empty set.

Another interesting approach is to let instructions control more than one channel; that is, in the graph G_C, one instruction can be connected to more than one channel. Different possibilities arise from this variant. Let us suppose that the instruction i is connected to k channels. When should i select the edge with the higher weight? When at least one virus goes through one channel? When all the channels transport a virus? When the majority of channels have moved a virus? The different choices will lead to different models that work in a very different

ITAT'23: Information technologies – Applications and Theory, September 22–26, 2023, Vysoké Tatry, Slovakia

dorellana@us.es (D. Orellana-Martín)

0000-0002-2892-6775 (D. Orellana-Martín)

way.

Even when these variants are not able to solve **NP**-complete problems efficiently, these new ingredients are really interesting for some applications, improving the running time for previously designed solutions. When a real-life cell replicates its ADN, without realizing it, it replicates also the genetic code of the virus. This behavior can be abstracted as a division of the host entity, duplicating its entire genetic code (the connections and internal elements). Potentially, this type of instruction could lead to presumably efficient virus machines, as it happens in the framework of membrane computing [5].

Acknowledgments

The research described in this work is supported by the Zhejiang Lab BioBit Program (Grant No. 2022BCF05).

References

[1] Gh. Păun, G. Rozenberg, A. Salomaa, DNA Computing, Springer Berlin Heidelberg, 1998.

[2] Gh. Păun, G. Rozenberg, A. Salomaa, The Oxford Handbook of Membrane Computing, Oxford University Press, Inc., USA, 2010.

[3] S. S. Haykin, Neural networks and learning machines, third ed., Pearson Education, Upper Saddle River, NJ, 2009.

[4] A. Eiben, J. Smith, Introduction to Evolutionary Computing, Springer Berlin Heidelberg, 2015.

[5] G. Păun, Computing with membranes: Attacking np-complete problems, in: I. Antoniou, C. S. Calude, M. J. Dinneen (Eds.), Unconventional Models of Computation, UMC'2K, Springer London, London, 2001, pp. 94–115.