Insertion Sorter in P Systems

Davood Pour Yousefian Barfeh*1,2,3 Jonalyn G Ebron1 and Jaderick P Pabico2
1Lyceum of the Philippines University-Laguna, Calamba City, Philippines
2University of the Philippines-Los Baños, Laguna, 4031, Philippines
3Adamson University, 1000 Manila, Philippines
*davoodpb@gmail.com

Abstract. In this study researchers pay attention to the essence of Insertion Sort and propose a sorter in Membrane Computing. This research shows how a theoretical computing device same as Membrane Computing can perform the basic concepts same as sorting. In this regard, researchers introduce conditional reproduction rule such that each membrane can reproduce another membrane having same structure with the original membrane. The researchers use the functionality of comparator P system as a basis in which two multisets are compared and then stored in two adjacent membranes. And finally, the researchers present the process of sorting as a collection of transactions implemented in four levels while each level has different steps.

1. Introduction
Membrane computing (MC) is a scope within computer science which explores to apperceive new computational models from the study of biological cells, particularly cellular membranes.

In 1998 when Gheorghe Păun initiated membrane computing [1], nobody was able to conceive this type of computing grows swiftly. The initial goal of membrane computing was to learn from biology especially cells something possibly useful to the field of computer science.

Păun officially proposed membrane computing in 2000 [2] and after that various types of membrane systems – known as P systems – were defined, all of them inspired from biological facts. Then numerous applications reported in several areas such as biomedicine, linguistics, computer graphics, economics, approximate optimization, and cryptography. Accordingly, several software products for simulating and implementing P systems were developed such as SNUPS (simulator of numerical P systems) and SRPSUMGPU (simulation of recognizer P systems by using Manycore GPU) [3–4].

2. Cell-like P systems
Cell-like P system – or for convenience we call them P system in this study – consists of many membranes arranged hierarchically. These membranes bound compartments. The compartments are the area where multisets of abstract objects are placed. The multisets are sets of objects (symbols) with multiplicities, while the objects are the chemicals in the compartments, swimming in some liquid such as water [5–6]. In other words, a multiset can be seen as a string in which multiplicity (number of each symbol) is significant, not the order of symbols. Since the objects are swimming and moving freely inside compartment, permutation or order of objects is not important [6–8]. For example consider multiset M1: abbbac, multiset M2: bacabb, and multiset M3: cbbaab. All of these multisets are the same, because in all of these multisets, number of object a is 2, number of object b is 3, and number of
object c is 1. As it can be observed the orders of objects in multisets are not important since the objects are inside water and can freely move.

It is noticeable that, in each compartment there are some rules and the objects inside the compartment evolve according these rules. In other words, number of objects may change based on rules. The rules direct how the objects change or how the objects communicate across membranes [6–8]. Although, Gheorghe Păun and others proposed some desirable rules based on behaviour of cells, all of these rules pay attention to the objects inside membrane, while membrane itself is left behind. More specifically, there are situations in which a cell may reproduce another cell or a membrane may reproduce another membrane. This is why; researchers propose some rule regarding membrane reproduction. Totally, in this study, there are three types of rules. One of these types of rules makes changes in number of objects, while another type of rules changes the positions of the objects, and the last type of rules allows a membrane to reproduce. In the following, these three types of rules are explained in details.

2.1. Rules regarding change in number of objects
These rules change some object to some other object. For example, based on rule R1: \(ca \rightarrow d\), our previous multiset (as mentioned, all of 3 multisets M1, M2 and M3 are the same) changes to \(abbbd\). As seen, one “a” and one “c” are transmuted to one “d”. Now, based on rule R1, number of objects changed. Number of object “a” is 1, number of object “b” is 3 and number of object “d” is 1 (we do not talk about object “c” any more, since its number is zero).

2.2. Rules regarding change in position of objects
Some rules change the position of the object. In fact, this type of rules transfers the object from some membrane to some other membrane. Simple example about how the objects communicate across membranes, based on rules R2: \(a \rightarrow ein\) and R3: \(d \rightarrow foutg\), our previous multiset (abbbd) changes to \(bbbg\), since object “a” is converted to object “e” and object “e” is sent to a membrane located inside current membrane and object “d” is converted to object “f” and object “g”, while the object “f” transfers to a membrane that covers the current membrane.

2.3. Rules regarding membrane reproduction
At some situations, membrane M can reproduce another membrane \(\bar{M}\) such that the structure of new membrane \(\bar{M}\) is exactly same as structure of the initial membrane M. Although we can have two types of reproduction - in and out, in this study we use the rule regarding reproduction-out.

When we use production-out rule for membrane M, another membrane \(\bar{M}\) having same structure with the initial membrane M is generated out of the initial membrane M. Here the new membrane \(\bar{M}\) may include the initial membrane M or not. In this study, new membrane \(\bar{M}\) includes the initial membrane M. For example, there exists a membrane \(m0\) containing two membrane \(m1\) and \(m2\) as shown in figure 1. If the rule reproduction-out is implemented on \(m0\), then another membrane \(\bar{m0}\) containing \(m1\) and \(m2\) is generated out of initial membrane \(m0\) while the new membrane \(\bar{m0}\) includes the initial membrane \(m0\) as shown in figure 2.

![Figure 1. Initial membrane m0 before reproduction-out](image-url)
As figure 2 shows, the structure of new membrane m0 (red color) is exactly same as the structure of the initial membrane m0 (black color). Here the structure means inclusion of other membranes, not inclusion of multisets. This is why; new membrane m0 does not include the multiset aaaa.

![Figure 2. After reproduction-out; initial membrane m0 (black color) and new membrane m0 (red color) (Image)](image)

The rules – belonging to all three types, can be implemented and applied in many ways. In general, rules emulate the biology in which chemical reactions performed in parallel as much as possible. In other words, transactions (rules) have maximal parallelism. There are another modes such as sequential, minimal parallelism and bounded parallelism. In sequential; one rule is used in each step, while in minimal parallelism; when a rule can be used, then at least one rule must be used, and in bounded parallelism; there is a restriction about the number of membrane or the number of rules to be used. In all modes mentioned, there is a common trait that is the objects and the rules regarding objects are chosen non-deterministically [1–6].

In membrane computing, a collection of transitions creates a computation. A computation generates a result as long as it halts that is to reach to a configuration where no rule can be applied [1–6].

3. Comparator P system
A comparator P system is able to sort two multisets as long as each multiset contains only one object, such as multisets s1: aaaaa and multiset s2: bbb. The comparator P system contains three membranes; m0, m1 and m2 while m0 covers m1 and m2 as shown in figure 3 [5].

At the beginning of the computation, two multisets which are a^x and b^y – where x is the number of object “a”, and y is the number of object “b” – are in m0 while m1 and m2 are empty. Then all transactions are performed in two steps in order as following;

step 1:
- m0: ab → abin-m1

step 2:
- m0: a → bin-m2
- m0: b → bin-m2
- m1: b → bout

In step 1, the equal number of a’s and b’s transfer to m1, the rest of objects stay in m0. In step 2, three rules are performed in parallel. The rest of objects in m0 are converted to “b” and transfer to m2. Moreover all b’s in m1 move out from m1. These b’s transfer to m0, and then because of rule b → bin-m2 in m0, they transfer to m2. Now, halting state happens since no rule can be applied. In this case, the larger number (longer string) is in m2 and the smaller is in m1 [5].
4. Insertion sort

Insertion sort is a comparison-based algorithm in which the elements of the input list are sorted one at a time. In this algorithm, the sorted sub-list is always maintained in the lower position of the list. Then the new item is inserted into the previous sorted sub-list such that the new sub-list is also sorted.

We consider the list with one item as a sorted list. Then we iterate, considering one item of the list each repetition and growing the sorted sub-list. In other words, at each iteration, we work on an item from the input list and find its position in sorted sub-list comparing and swapping (if needed) the item with the elements of sorted sub-list started from the end. The iteration stops when swapping stops. The new sub-list is sorted and ready for the next iteration and new item from the list [9–12].

Insertion sort is slow compared to the advanced sorting algorithms such as quicksort, merge sort and heapsort [9]. Based on time complexity and number of comparison, insertion sort is as slow as bubble sort is; the worst case and the average case for both of insertion sort and bubble sort is $O(n^2)$ while the best case is $O(n)$ where $n$ is the number of elements of the input list. Although regarding time complexity, insertion sort is weak as bubble sort is, it has some strengths that make it bright despite bubble sort that is disregarded by some computer scientists [10–11]. The strengths of insertion sort can be listed as:

a) Simplicity: Jon Bentley used C programming language and implemented insertion sort only in three lines. He also implemented the optimized version of insertion sort in five lines [12].

b) Adaptiveness: insertion sort efficient for lists in which the elements are already considerably sorted. In this case the time complexity of insertion sort is $O(nm)$ when each element in the input list is no more than $m$ places far from its sorted position [9–12].

c) Stability: insertion sort does not change the relative order of items in the input list with equal indices [9–12].

d) In-placement: insertion sort needs only one extra unit memory space for swapping of two elements of the input list. The extra unit is small as size of elements of input list.

e) Online-ness: insertion sort sorts a list as it receives the elements of the list one-by-one [10–12].

Researchers appreciate these strengths as well as natural characteristic of insertion sort that is comparing two adjacent items then swapping them (if needed), and then implement insertion sort in comparator P system to propose a sorter in the field of membrane computing.

5. Insertion sorter in P systems

Now, researchers propose a membrane sorter containing comparator P system that uses insertion sort as algorithm to sort a list of multisets.

All membranes used in this sorter are in the form of comparator P system. When some comparator P system is reproduced, it is exactly a comparator P system which has 3 membranes; $m_0$, $m_1$ and $m_2$ while $m_0$ covers $m_1$ and $m_2$. Moreover all rules of new comparator P system are inherited from initial comparator P system.

The rules in insertion membrane-sorter has 4 levels which are performed in order inside each comparator P system, and each level has different steps which are also performed in order. At the
beginning of the computation there is only one comparator P system while first and second multisets are inside m0 and following rules are applied;

level 1:
   step 1:
      m0: ab → ab_{in-m1}
   step 2:
      m0: a → b_{in-m2}
      m0: b → b_{in-m2}
      m1: b → b_{out}

level 2:
   step 1:
      m2: b → b_{out}
   step 2:
      m0: reproduction_{out} | if there is no outer membrane
   step 3:
      m0: b → b_{out}

level 3:
   m1: a → b_{out}
   m0: b → a_{in-m0} | if there is inner m0

level 4:
   outside: new multiset → a-multiset_{in}

At the start point; there exists only one comparator P system while two empty multisets m1 and m2 are inside m0.

Level 1 contains exactly the rules of a single comparator P system. At the end of this level, the greater number (multiset) is in m2 and the smaller is in m1 while m0 is empty.

In level 2, at step 1; the greater number transfers from m2 to m0. At step 2; if current m0 is the most outer membrane, then a new comparator P system is reproduced out of current m0 while new comparator P system includes current m0. At step 3; the greater number that contains only b’s, transfers to outer membrane. At the end of level 2, the greater number is in m0 of outer comparator P system, while the smaller number is still in m1 of current comparator P system while m0 of current comparator P system is empty.

In level 3, the smaller number transfers from m1 to m0 while converting to b’s. Moreover, if there is an inner m0, the number goes inside inner m0 to be compared with another number inside another comparator P system.

In level 4, if there is still some multiset in the list, it goes inside m0 of the outer comparator P system to be compared with the number that is inside m0 of the outer comparator P system.

It is notable that all levels can be performed in parallel but in different comparator P systems. For example, some comparator is working on level 2, and some other comparator is working on level 3, while another comparator might be idle.

The sorter system stops when there is no rule to be done. In this case, there is no multiset to go inside m0 of the outer comparator P system that is the rule in level 4, and all multisets are already sorted that are rules in levels 1, 2 and 3. When the system stops, the greatest number is in m0 of outer-most comparator and the smallest is in m0 of the inner-most comparator P system. Moreover, all m1’s and all m2’s are empty.

The proposed sorter uses natural characteristics of insertion sort in which compares two adjacent numbers (multisets) and then swaps them (if needed) in levels 1, 2 and 3. These comparisons and swaps are performed until the sub-list is sorted. Moreover, the sorter is working online same as insertion sort; when the new item arrives in level 4, the item finds its position with comparing and then
swapping (if needed) started from the end of sub-list that is the outer-most comparator P system. In this way, at the end of each iteration, the sub-list is sorted that is the greatest number is in m0 of the outer-most comparator P system and the smallest number is in m0 of the inner-most comparator P system.

Acknowledgments
The researchers would like to extend profound gratitude and appreciation to the people from Lyceum of the Philippines University – Laguna, who contributed assistance to the fulfilment of this research paper, especially Dr. Peter Laurel – President, engineer Ricky Bustamante – Dean, College of Engineering and Computer Studies, and Mrs. Gerby Muya – Director, Research and Statistics Center.

References
[1] Păun G 1998 Introduction to Membrane Computing
[2] Păun G 2000 Computing with membranes Journal of Computer and System Sciences 61(1) pp 108–143, and TUCS Report No 208.
[3] Gutierrez–Naranjo M A, Perez–Jimenez M J and Riscos–Nunez A 2006 Available membrane computing software Applications of Membrane Computing Springer–Verlag, Chapter 15 pp 411–436
[4] Martinez–del–Amor M A, Perez–Hurtado I, Perez–Jimenez M J, Cecilia J M, Guerrero G D and Garcia J M 2009 Simulation of Recognizer P Systems by Using Manycore GPUs Membrane Computing: 10th International Workshop
[5] Ardelean I, Ceterchi R and Tomescu A I 2008 Sorting with P Systems: A Biological Perspective. Romanian Journal of Information Science and Technology 11(3) pp 243–252
[6] Tingfang W, Zhiqiang Z, Păun G and Pan L 2016 Cell-like spiking neural P systems. ELSEVIER: Theoretical Computer Science 623 pp 180-189
[7] Singh K, Deep K and Nagar A K 2014 Cell-like P-systems based on rules of Particle Swarm Optimization. ELSEVIER: Applied Mathematics and Computation 246 pp 546-560.
[8] Chen K, Wang J, Li M, Ming J, and Peng H 2015 Cell-Like Fuzzy P System and Its Application of Coordination Control in Micro-grid Springer: Bio-Inspired Computing - Theories and Applications
[9] Bender M, Farach - Colton M and Mosteiro M 2006 Insertion sort is O(n log n) Journal of Theory of Computing Systems 39(3) pp 391–397
[10] Knuth D 1998 The Art of Computer Programming Volume 3: Sorting and Searching, Second Edition. Addison-Wesley
[11] Owen A 2003 Bubble sort: an archaeological algorithmic analysis ACM SIGCSE Bulletin
[12] Bentley J 2000 Programming Pearls ACM Press Addison–Wesley