An. Şt. Univ. Ovidius Constanţa Vol. 11(1), 2003, 195–206

A STATIC SORTING ALGORITHM FOR P SYSTEMS WITH MOBILE CATALYSTS *

Dragoş Sburlan

To Professor Silviu Sburlan, at his 60's anniversary

Abstract

We propose an integer sorting algorithm using P systems with mobile catalysts. The paper considers the case when the number of components of the vector to be sorted is a fixed number k. The components of the vector are represented as the multiplicities of certain objects present in one membrane. The result of the computation will be given in the outer region by eliminating objects according to their initial multiplicities order. Due to the massive parallel computational feature of P systems (and despite the "unorder" of the multiset or the "weakness of the ingredients" used in computation) the time complexity of the algorithm depends linearly on the maximum of the values to be sorted and not on the number of values to be sorted as in classical sequential cases.

1 Introduction

P systems with symbol objects and mobile catalysts are powerful computational devices inspired by biochemical molecular interactions. Being computationally universal and having a massive parallelism feature, P systems can be used for obtaining better results in solving problems as opposed to classical devices. Many algorithms have been proposed especially for solving intractable problems like SAT or integer factoring problem in polynomial time usually considering strong variants of P systems (with features like context-sensitive rules, priorities among rules, promoters/inhibitors, membrane division/creation). Here, we propose an algorithm for solving a much easier problem for which polynomial solutions are know when using sequential machines. However, obtaining better results, in some respect, than in the classical case and, moreover, using only object rewriting context-free rules and catalytic rules (the "weakest" possible sensitivity) is a challenging task.

Key Words: Static Strong Sorting, Static Ranking, P Systems, Mobile Catalysts
*Research supported by PhD. fellowship from the Spanish Ministry of Foreign Affairs

2 Preliminaries

A P system (of degree m, $m \geq 1$) with symbol objects and rewriting evolution rules is a construction

$$\Pi = (V, C, \mu, w_1, \ldots, w_m, (R_1, \rho_1), \ldots, (R_m, \rho_m), i_0),$$

where:

- V its an alphabet; it elements are called *objects*;
- $C \subseteq V$ is a set of catalysts;
- μ is a membrane structure consisting of m membranes usually labeled $1, 2, \ldots, m$;
- w_i, $1 \leq i \leq m$, specify the multisets of objects present in the corresponding regions $i, 1 \leq i \leq m$ at the beginning of a computation;
- R_i, $1 \leq i \leq m$, are finite sets of evolution rules over V associated with the regions $1, 2, \ldots, m$ of μ, and ρ_i is a partial order relation over R_i (a priority relation); these evolution rules are of the form $a \to v$ or $ca \to c_{\{here,out,in\}}v$ where a is an object from $V \setminus C$ and v is a string over $(V \setminus C) \times (\{here, out, in\})$ (In general, the target indications *here*, *out*, *in* are written as subscripts of objects from V; we will omit the use of the subscript *here*.);
- i_0 is a number between 0 and m and specifies the output membrane of Π (in case of 0, the environment is used for the output).

Now, starting from an initial configuration, the system evolves according to the rules and objects present in the membranes, in a non-deterministic maximally parallel manner, and according to an universal clock. The system will make a successful computation if and only if it halts: there is no rule applicable to the objects present in the halting configuration. For details regarding the used notations we refer to [5].

The language generated by a P system Π is the language $L(\Pi)$ that contains all the strings produced by all successful computations in Π according to the output-mode. In our case, the system will generate one string (the computation is deterministic) in the environment.

3 Ranking and Strong Static Sorting

3.1 Definitions and notations

Let $V = \{a_i \mid 1 \leq i \leq k\}$ be an alphabet. A nonempty word over V is denoted by $w = \prod_{j=1}^{m} a_j$, $m \in \mathbf{N}^+$, where $a_j \in V$ for each $1 \leq j \leq m$.

Definition 1 *The word* $w = \prod_{j=1}^{k} a_j \in Perm(V)$, $k = card(V) \in \mathbf{N}^+$, *where* $a_j \in V$ *such that* $M(a_j) \leq M(a_{j+1})$, *for each* $1 \leq j \leq k-1$, *is called the* ranking string *of the multiset M.*

Definition 2 *The word* $w = \prod_{j=1}^{k} a_j^{M(a_j)}$ *is called a* strong sorting string *of* M *if* $\prod_{j=1}^{k} a_j$ *is the ranking string of* M.

Remark 1 *In this case, we are interested in obtaining as the output of computation the objects with the same associated multiplicities, present in a string in the increasing order of their multiplicities.*

Remark 2 *Another weaker definition for strong sorting can be: the word* $w = \prod_{j=1}^{k} a_j^{M(a_j)}$ *is called* strong sorting string *of* M *iff for each* $1 \leq i \leq k$ *we have* $|w|_{a_i} = M(a_i)$ *and for any representation* $w = \alpha a_i \beta a_j \gamma$ *we have* $M(a_i) \leq M(a_j)$. *In the case of equality of some numbers either a decision should be made whether* " $=$ " *is* " $\leq$ " *or* " $\geq$ ".

Example 1 *For the alphabet* $V = \{a, b, c\}$ *and the multiset* $M = \{(a, 20), (b, 10), (c, 30)\}$, *we have:*

- *ranking string :* $b\ a\ c$
- *strong sorting string :* $b^{10}\ a^{20}\ c^{30}$.

The strong sorting is the most difficult static sorting problem. The common approach for solving it is to first obtain the ranking string and then, controlling the output process.

Let us start by considering that the input of the system is placed in one specific region.

3.2 Sorting with mobile catalysts

We will use $2k + 1$ catalysts, from which, the first $2k$ are associated with the objects (two catalysts for a symbol object) whose multiplicities we want to sort, plus one which will be used to "clean" unneeded symbols. With all this, we can reach our goal in an unexpectedly good time with respect to the "weakness" of the ingredients.

But this will give the answer for the ranking problem only, and one can remark that passing from the ranking problem to the sorting problem (using only context-free and catalytic rules) is not a trivial task. This is due to the fact that the rules must be independent of the initial multiplicities of the objects representing the initial vector (no rules of type $ca \to ca_{out}^{n_1}$ or $a \to a_{out}^{n_1}$ are used).

Here is the general algorithm for solving the ranking problem:

Algorithm 1 *Let* $v = \Pi_{i=1}^{k} a_i$.
Consider $\Pi = (O, C, [_1[_2[_3]_3]_2]_1, s^k \Pi_{j=1}^{k} {a_j}^{n_j}, \Pi_{j=1}^{k} X_j, K^k, R_1, R_2, R_3, 0)$, *where*

$$
\begin{aligned}
C &= \{X_i \mid 1 \le i \le k\} \cup \{K\}, \ \ O = C \cup \{a_i, y_i \mid 1 \le i \le k\} \cup \{s, f, t\},\\
R_1 &= \{a_j \to a_{j\,in}, X_j y_j \to X_{j\,out} \mid 1 \le j \le k\} \cup \{s \to f, f \to s_{in}\},\\
R_2 &= \{X_j a_j \to X_{j\,in} a_{j\,in}, X_j s \to X_{j\,out} y_{j\,out} \mid 1 \le j \le k\}\\
&\cup \ \{Ks \to K_{in} f_{out}\}, \ R_3 = \{K a_j \to K_{out} t, X_j t \to X_{j\,out} \mid 1 \le j \le k\}.
\end{aligned}
$$

Example 2 *Consider the case of sorting 3 numbers. Let us rename variables for simplicity:* $a_1 = a$, $X_1 = A$, $y_1 = a'$, *etc.*
Initial data: $[_1 a^{n_1} b^{n_2} c^{n_3} s^3 [_2 ABC [_3 K^3]_3]_2]_1$
Rules of region 1: $a \to a_{in}$, $b \to b_{in}$, $c \to c_{in}$, $s \to f$, $f \to s_{in}$, $Aa' \to A_{out}$, $Bb' \to B_{out}$, $Cc' \to C_{out}$;
Rules of region 2: $Aa \to A_{in} a_{in}$, $Bb \to B_{in} b_{in}$, $Cc \to C_{in} c_{in}$, $As \to A_{out} a'_{out}$, $Bs \to B_{out} b'_{out}$, $Cs \to C_{out} c'_{out}$, $Ks \to K_{in} f_{out}$;
Rules of region 3: $Ka \to K_{out} t$, $Kb \to K_{out} t$, $Kc \to K_{out} t$, $At \to A_{out}$, $Bt \to B_{out}$, $Ct \to C_{out}$.

The P system that compute the ranking problem for the case of three integers is depicted in Figure 1.

We present now the configurations of the P system during the computation for the case of 3 numbers; one can notice that the first two steps initialize the system and then in rounds of four computational steps, all the components will be decreased up to the time when one component is exhausted (in the configuration table presented the component with objects a is exhausted first). Then the catalyst which corresponds to the exhausted component will leave the system and will arrive in the environment. In the table, in the column representing the computational steps, we denote by $0, 1$ the initialization procedure, by $1, 2, 3, 4$ the behavior of the system when all the components are present, and by $1', 2', 3', 4'$ the behavior of the system when one component is exhausted.

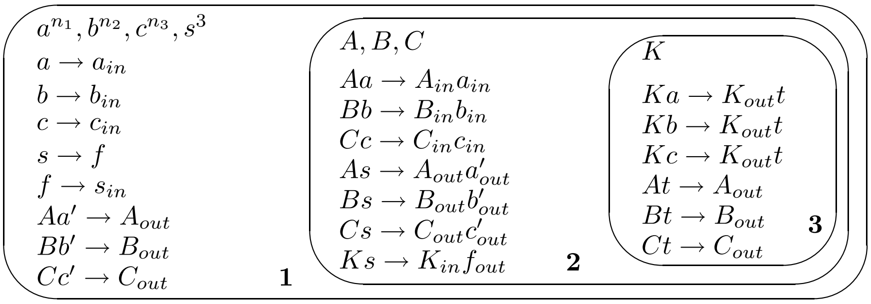

Figure 1: The P system solving the 3 integer ranking problem.

Step	environment	Region 1	Region 2	Region 3
0		$s^3a^{n_1}b^{n_2}c^{n_3}$	ABC	K^3
1		f^3	$a^{n_1}b^{n_2}c^{n_3}ABC$	K^3
1		f^3	$a^{n_1}b^{n_2}c^{n_3}ABC$	K^3
2			$s^3a^{n_1-1}b^{n_2-1}c^{n_3-1}$	$K^3ABCabc$
3			$s^3K^3a^{n_1-1}b^{n_2-1}c^{n_3-1}$	$tttABC$
4		f^3	$a^{n_1-1}b^{n_2-1}c^{n_3-1}ABC$	K^3
$1'$		f^3	$b^{n_2}c^{n_3}ABC$	K^3
$2'$			$s^3Cb^{n_2-1}c^{n_3-1}$	$K^3ABCabc$
$3'$		Aa'	$ssKKb^{n_2-1}c^{n_3-1}$	$KttBC$
$4'$	C	f^2	$b^{n_2-1}c^{n_3-1}BC$	K^3

For the general case the configuration table is:

Step	environment	Region 1	Region 2	Region 3
0		$s^k\Pi_{j=1}^k {a_j}^{n_j}$	$\Pi_{j=1}^k X_j$	K^k
1		f^k	$\Pi_{j=1}^k {a_j}^{n_j}\Pi_{j=1}^k X_j$	K^k
1		f^k	$\Pi_{j=1}^k {a_j}^{n_j}\Pi_{j=1}^k X_j$	K^k
2			$s^k\Pi_{j=1}^k {a_j}^{n_j-1}$	$K^kPi_{j=1}^k X_j\Pi_{j=1}^k a_j$
3			$s^kK^k\Pi_{j=1}^k {a_j}^{n_j-1}$	$t^kPi_{j=1}^k X_j$
4		f^k	$\Pi_{j=1}^k {a_j}^{n_j-1}\Pi_{j=1}^k X_j$	K^k
$1'$		f^k	$\Pi_{j=1}^{k-1} {a_j}^{n_j}\Pi_{j=1}^k X_j$	K^k
$2'$			$s^kX_k\Pi_{j=1}^{k-1} {a_j}^{n_j-1}$	$K^kPi_{j=1}^{k-1} X_j\Pi_{j=1}^{k-1} a_j$
$3'$		X_ky_k	$s^{k-1}K^{k-1}\Pi_{j=1}^{k-1} {a_j}^{n_j-1}$	$Kt^{k-1}Pi_{j=1}^{k-1} X_j$
$4'$	X_k	f^{k-1}	$\Pi_{j=1}^{k-1} {a_j}^{n_j-1}\Pi_{j=1}^{k-1} X_j$	K^k

Now let us see which is the idea for this system. First we have the symbol f (a checker) which goes back and forth from membrane 1 to membrane 2.

It will react in different ways according to what it finds in the membrane 2. In this membrane, by applying the rules of type $X_j a_j \rightarrow X_{j\,in} a_{j\,in}$ we "erase" one symbol from each component in a step of computation. Up to the time when one component is completely erased, the rules of type $Ks \rightarrow K_{in} f_{out}$ will be applied. The catalyst K is also executing a back and forth oscillation between membranes 2 and 3, but with a different timing then f. This is done because we would want that if there are still elements to delete, the rules of type $X_j a_j \rightarrow X_{j\,in} a_{j\,in}$ must be executed before the rules of type $Ks \rightarrow K_{in} f_{out}$. When one component is "removed" from the initial multiset then the corresponding catalyst will not leave membrane 2, and so, the rule $X_j s \rightarrow X_{j\,out} y_{j\,out}$ can be applied. This means that we can send a signal representing the fact that the object with the smallest multiplicity has been reached. In the same time the corresponding catalyst will enter in membrane 1 and it will not participate anymore in further tasks. The process will keep going on up to the time when all components will be exhausted. In this way we obtain a ranking algorithm which will have the time complexity proportional to the maximum multiplicity of the objects from the initial multiset. Now, once we have the ranking problem solved we can go further to solve the sorting. First, one can remark that a signal is a catalyst or a common object (the rule to send signals in membrane 1 is of type $X_j s \rightarrow X_{j_{out}} y_{j_{out}}$). Of course, one can send out a finite set of objects but this will not essentially affect the whole process. Another remark is that, also, before the process begins, we can make a copy of the initial multiset (for example, we replace the rule $a_j \rightarrow a_{j\,in}$ by the rule $a_j \rightarrow a_{j\,in} a_j$ in the membrane 1) for using later when we would like to eliminate objects in the right order. Since a rule of type $X_j y_j \rightarrow Xjy_j^{n_1}$ is not good, because we would like to have a system that dissociates the input from the implementation, we have to construct the output based on the signals that we receive. Moreover, the elements cannot be eliminated all at once because this is implied by the use of a non-cooperative rule; however, having such kind of rules we cannot control the process. So, the only solution is to again use catalysts. No matter how it is done, this will be a difficult task because the signals representing the order are not sent out in the "right" interval of time (the interval from which we can deduce easily the multiplicity of a certain object). Consider the case when the multiplicities of the objects are $n_1 < n_2 < n_3 \cdots$, then the first signal will appear after a time proportional to n_1, the second after a time $n_2 - n_1$ and so on. As it can be seen, if in our example we have $n_1 > (n_2 - n_1)$, then we will have "overlapping" tasks meaning that at a certain moment we will have more than one signal present in the same membrane. To overcame this, one can use the same techniques as were used before, to allow only one catalytic rule to work, up to the time when all elements from a specified component will be exhausted.

Theorem 1 *The time complexity for the ranking algorithm with P systems with mobile catalysts is linear with respect to the maximum of elements to be ranked; it does not depend on the number of elements.*

Proof. The assertion is a consequence of the facts presented above. We can remark only that the system depends on the number of elements to be ranked (the number of catalysts is $k+1$ where k is the number of elements).

Now, once we have obtained the ranking solution of the problem in linear time, the next step is to expel the symbols to the environment, according to the ranking solution. One can see that when we compute the ranking problem, the catalysts leave the system according to the corresponding objects' multiplicities.

Let us come back to the example with three numbers. In this example, the first catalyst leaves the system after n_1 steps, the second catalyst after an extra $n_2 - n_1$ steps and finally the third catalyst after other $n_3 - n_2$ steps of computation. Since we cannot use context-free rule for eliminating symbols in the right order as we mention before we have to use catalytic rules. But in this case, if we use directly rules of type $Aa \rightarrow Aa_{out}$ we can have mixed objects in the output (consider for example that $n_1 > n_2 - n_1$; then catalysts C and B will eliminate objects simultaneously).

In order to overcome this, we will design a P system which will be responsible for the elimination of symbols in the right order. Let us start with a three integer sorting example. First let us denote by *Rank* the module presented above. In order that the elimination of a certain symbol not to interfere with the elimination of an another we have to distribute each catalyst which represents a signal in different, consecutive membranes in such a way that the first signaling catalyst arrives to the outermost membrane, the second signaling catalyst to the second outermost membrane and so on. This can be achieved very easily by a construct like the following (let us consider for simplicity the case of three signaling catalysts that leave the rank module; moreover for the sake of simplicity we will show only the rules for the signal catalysts A – all the other rules can be constructed in a very similar way):

Recall that in the *Rank* example $n_1 < n_2 < n_3$ and so the signal catalysts were eliminated in the order A, B and finally C. This means that the catalyst A will leave the *Rank* module first and will arrive in region 4. There, by the rule $At \rightarrow A_{out}$, will consume an object t and will go to region 5. From there, by using a rule of the same type as before, it will finally arrive in membrane 6. One can see that if another signal catalyst (say B) leaves the *rank* module, it will arrive to region 5 and will not go further because all the objects t have

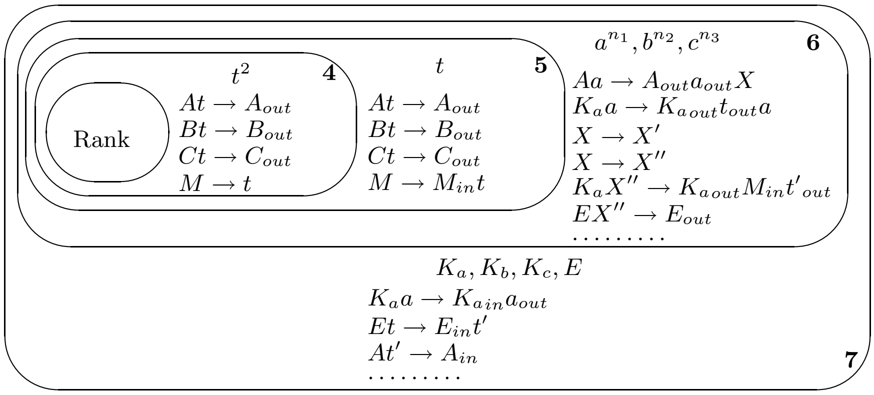

Figure 2: The P system solving the 3 integer strong sorting problem.

been already consumed. In this way, signal catalysts are separated in different membranes according to the multiplicity they represent.

Now, we have the catalyst A in the region 6 and also the multiset $a^{n_1}, b^{n_2}, c^{n_3}$ ($a^{n_1}, b^{n_2}, c^{n_3}$ can be transported here from the input membrane only using context-free rules). As we said before for the sake of simplicity we will consider only the rules involving object a (the others are very similar, namely, for instance, just add the rules obtained by replacing A with B, a with b and K_a with K_b in membranes 6 and 7).

In region 6 we want to eliminate objects corresponding to the signal that we receive. Once we finish this task we also would like for the next signal catalyst (which stays in a queue) to enter in the region 6 and start the elimination process again. Also, all other signal catalysts must "advance" one region up.

Since in our case the catalyst A enters in region 6, the rule that can be applied is $Aa \to A_{out}a_{out}X$ which will decrease by one the number of objects a. As an effect, an object X will be produced and both the catalyst A and the corresponding object a will arrive into region 7. There, by the rule $K_a a \to K_{a\,in}a_{out}$ the object a will be sent to the environment, while the catalyst K_a (a "checker" – this catalyst will check if there are still objects a present in region 6) will arrive into region 6; if there are still present objects a the catalyst K_a will go to its initial position (region 7). Also, the catalyst A will come back to region 6 and the process will start over again. If the catalyst K_a does not find any object a in membrane 6, an object M ("message") will be sent to the inner membranes. This object will be responsible for the "advancing" in the "queue" of the signaling catalysts. The catalyst E present initially in the region 7 is responsible for "cleaning" operations of undesired symbols, while the objects X, X', X'' are used for synchronizing different tasks. For a better understanding we present the table of configurations: The case when $n_1 \geq 2$:

time	region 5	region 6	region 7	environment
1	$B?$	$A, a^{n_1}, b^{n_2}, c^{n_3}$ $Aa \rightarrow A_{out}a_{out}X$	K_a, K_b, K_c	
2	$B?$	$a^{n_1-1}, b^{n_2}, c^{n_3}X$ $X \rightarrow X'$	K_a, K_b, K_c, E, A, a $K_a a \rightarrow K_{a in} a_{out}$	
3	$B?$	$K_a, a^{n_1-1}, b^{n_2}, c^{n_3}, X'$ $K_a a \rightarrow K_{a out} a t_{out}$ $X' \rightarrow X''$	K_b, K_c, E, A	a
4	$B?$	$a^{n_1-1}, b^{n_2}, c^{n_3}, X''$	K_a, K_b, K_c, E, A, t $Et \rightarrow e_{in}t'$	
5	$B?$	$a^{n_1-1}, b^{n_2}, c^{n_3}, X'', E$ $EX'' \rightarrow E_{out}$	K_a, K_b, K_c, A, t' $At' \rightarrow A_{in}$	
6=1'	$B?$	$A, a^{n_1-1}, b^{n_2}, c^{n_3}$ $\ldots$	K_a, K_b, K_c, E $\ldots$	

The case when in region 6 remains only one object a:

time	region 5	region 6	region 7	environment
1	$B?$	A, a, b^{n_2}, c^{n_3} $Aa \rightarrow A_{out}a_{out}X$	K_a, K_b, K_c	
2	$B?$	$b^{n_2}, c^{n_3}X$ $X \rightarrow X'$	K_a, K_b, K_c, E, A, a $K_a a \rightarrow K_{a in} a_{out}$	
3	$B?$	$K_a, b^{n_2}, c^{n_3}, X'$ $X' \rightarrow X''$	K_b, K_c, E, A	a
4	$B?$	$K_a, b^{n_2}, c^{n_3}, X''$ $K_a X'' \rightarrow K_{a out} M_{in} t'_{out}$	K_b, K_c, E, A	
5	$B?, M$ $M \rightarrow M_{in}t$	b^{n_2}, c^{n_3}	K_a, K_b, K_c, E, A, t' $At' \rightarrow A_{in}$	
6	$B?, t$ $Bt \rightarrow B_{out}$	A, b^{n_2}, c^{n_3}	K_a, K_b, K_c, E	
7	$C?$	$A, B?, b^{n_2}, c^{n_3}$ $Bb \rightarrow B_{out}b_{out}X$?	K_a, K_b, K_c, E	
8	$\ldots$ $\ldots$	$\ldots$ $\ldots$	$\ldots$ $\ldots$	$\ldots$ $\ldots$

For the k integer sorting problem we will have a number of membranes proportional to k. The mechanism used in the example for advancing in the queue of signal catalysts will be preserved. Practically we will have a system with 3 (ranking problem) + k (the queue device) +1 membranes and a number of $2k + 2$ catalysts.

The fact that we consume an object from each component at a time because we use only catalytic rules (the context free rules cannot be applied since with them we cannot control the computation), and that we transport catalysts to a precise position in a queue by using catalytic rules, and finally that we eliminate objects also with the help of catalysts, gives a hint on the time complexity of the hole algorithm.

Remark 3 *The time complexity for the strong sorting algorithm with P systems with mobile catalysts is linear with respect to the maximum of the elements to be sorted. It also depends on the number of components to be sorted and on the sum of elements.*

The result is a consequence of the facts presented in the previous sections. One can see that the ranking solution of the problem is given in linear time, the transporting of catalysts to their right positions depends on the number of membranes (and so, depends on the number of components). The elimination process depends linearly on the sum of elements.

4 Conclusion

We have studied the possibility to solve the sorting problem using P system with mobile catalysts. One interesting result concerning this topic is that starting with objects that do not have any order and are mixed together in what formally we call a multiset, we compute, and after a specified time we obtain an ordered string. The characteristic of this algorithm is that we sort by "carving" meaning that we consume objects iteratively, one symbol from all the components at a time, and signaling when a modification occurs in the system (usually we trigger a signal when a certain component was eliminated). Sorting problems are among the most important problems in computer science theory. Beside them there are a lot of other problems which are waiting to be solved in the framework of P systems. The reason is that we can obtain better results in time complexity than the classical algorithms by using the massive parallelism feature of the P systems. We believe that some techniques (the comparison method in the case of mobile catalysts and non-cooperative rules, the synchronization methods used) developed in the paper can be also used to construct systems that solve such kinds of problems. The improvements of current algorithms (by reducing the number of membranes when this is the case, reducing the number of catalysts and so on) are also left open.

Acknowledgements. The author is very grateful to all the GRLMC staff from "Rovira i Virgili" University for the interesting discussions regarding this topic.

References

[1] A. Alhazov, D. Sburlan, Static Sorting Algorithms for P Systems, *Workshop on Membrane Computing*, Tarragona 2003, accepted.

[2] J.J. Arulanandham, Implementing Bead-Sort with P Systems, *Unconventional Models of Computation 2002* (C.S. Calude, M.J. Dineen, F. Peper, eds.), LNCS 2509, Springer-Verlag, Heidelberg, 2002, 115?-125

[3] R. Ceterchi, C. Martín-Vide, *P Systems with Communication for Static Sorting*, GRLMC Report 26 (M. Cavaliere, C. Martín-Vide, Gh. Paun, eds.), Rovira i Virgili University, 2003.

[4] R. Ceterchi, D. Sburlan, *Simulating Boolean Gates with P Systems*, *Workshop on Membrane Computing*, Rovira i Virgili University, 2003, accepted.

[5] Gh. Păun, *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, Heidelberg, 2002.

[6] Gh. Păun, G. Rozenberg, *A Guide to Membrane Computing*, Theoretical Computer Science, 287, 1 (2002), 73–100.

"Ovidius" University
Faculty of Mathematics and Informatics,
Bd. Mamaia 124,
8700 Constantza,
Romania
Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tárraco 1,
43005 Tarragona,
Spain
e-mail: dsburlan@univ-ovidius.ro