Membrane computing is a theoretical model of computation inspired by the structure and functioning of cells. Membrane computing models naturally have parallel structure, and this fact is generally for all variants of membrane computing like kernel P system. Most of the simulations of membrane computing have been done in a serial way on a machine with a central processing unit (CPU). This has neglected the advantage of parallelism in membrane computing. This paper uses multiple cores processing tools in MATLAB as a parallel tool to implement proposed feature selection method based on kernel P system-multiobjective binary particle swarm optimization to identify marker genes for cancer classification. Through this implementation, the proposed feature selection model will involve all the features of a P system including communication rule, division rule, parallelism, and nondeterminism.

1. Introduction

The concept of membrane computing (MC) pursues the basics processes taking place in living cell based on its compartmental structure [1]. In general, the so-called P system represents the MC structure in mimicking reactions and rules. This system mimics the basics of cell-like variants such as cell division, cell death, the transformation of objects via rules, and halt process just when there are no more applicable rules. The nature of MC and its tools as P systems are conducted based on three important features including communication rule, parallelism, and nondeterminism. According to the nature of P system, it is quite suitable to represent biological systems, and all molecular interactions can take place in different locations of living cells [2]. Various variants of P system are considered to solve a wide range of problems. Some of the recent studies are [3] which has used the enzymatic numerical P system (EN P system) as the variant of P system mix with active membrane to solve subset sum problem, [4] which is the application of enzymatic numerical P systems discussed in the area of mobile robots control, [5] that used P system as framework to give a better understanding of the pattern in regenerative biology, [6] which applies membrane computing in google earth application, [7] which applies membrane computing in obtaining optimal values of the thresholds, and [8] which introduces various application of membrane computing in real life.

Recently, new models of P systems have been explored. A kernel P system (KP system) based on the tissue P system (graph-based) has been defined, which consists of a low-level specification language that uses established features of existing P system variants and also includes some new elements. KP system, first because of its coherent set of rules, second flexibility of updating rules in different part of modelling, and third its tissue based model which is more adaptable with particle modelling of optimization algorithms, is selected among other variants of P systems to model the multiple-objective binary particle swarm optimization (MObPSO) model in parallel execution. Importantly, KP systems offer a coherent way of integrating these elements.
into the same formalism [9]. Artificial Intelligence (AI) refers to a machine or algorithm which tries to mimic a cognitive function that human uses for learning or problem-solving. Inspired by AI algorithms—the intelligence that a machine or program demonstrates to address a problem—for feature selection of cancer microarray data, here, we propose a membrane-inspired feature selection method to use the potentials of membrane computing, such as decentralization, nondeterminism, and maximal parallel computing, to address the limitations of AI-feature selection.

Particle swarm optimization has been used to develop feature selection methods in microarray gene expression studies [10–12]. Graph-based MObPSO [13] was modelled through kernel P system for the following reasons: (1) it has ability to model genes (nodes) and define relationships between them (edges); (2) it has a higher accuracy as compared with flat (filter and wrapper) methods, sequential backward elimination (SBE), correlation-based feature selection (CFS), minimum redundancy maximum relevance (mRMR), and sequential forward search (SFS).

Based on the interaction on membrane computing and evolutionary computation, the field of membrane-inspired evolutionary algorithms (MIEAs) has been introduced in the study [14]. After that, more studies have been done to mix the evolutionary algorithms and P systems (e.g., [2, 15, 16]). Some other studies also focused on hybridization of P system with optimization algorithms: for example, [17] combined P system with particle swarm optimization for the aim of minimizing nonlinear optimization problem, [18] combined P system with particle swarm optimization for the objective of enhancing accuracy of particle swarm optimization as well as overcoming the premature convergence, and [19] proposed a particle swarm optimization P system, the so-called PSOPS, and examined the model on seven-bench function optimization problem and concluded that effectiveness of method improved compared to PSO. Moreover, there are other combinations of PSO with membrane computing, for example, [2, 20–22].

These reviews, of previous studies, prove the usefulness of introducing the P systems into EAs and particularly optimization algorithms to improve pure EAs and optimization algorithms. To the best of our knowledge, there is not any work focusing on the use of a kernel P system into optimization algorithms using all characteristics of a membrane system including parallel implementation and particularly in the scope of cancer dataset’s feature selection. In this paper, a hybrid model of a kernel P system with multiobjective particle swarm optimization is proposed to improve the performance measures to compare with pure multiobjective particle swarm optimization. The main contribution of this paper is to parallelize the implementation of the proposed KP-MObPSO model that will lead to building a membrane-inspired optimization model taking full advantage of membrane computing.

This article can be summarized as follows. (1) in Section 2, a brief of basic concepts is brought regarding KP system and MObPSO approaches that are used to build the proposed model. (2) In Section 3, first an overview of the proposed KP-MObPSO is explained and then the four steps of building KP-MObPSO are displayed in detail, namely, initialization, evolution, interaction with client, and collecting the final result. The last two steps define how the proposed KP-MObPSO model is parallelized via multicore. (3) In Section 4, technical explanations of executing KP-MObPSO on multicore are brought to provide a clear overview of parallelizing particles for swarm optimization. (4) Section 5 displays the type of cancer dataset, the methodology of the preprocessing dataset, and simulation of a proposed model via real dataset. (5) In Section 6, the performance of proposed KP-MObPSO model compared with pure MObPSO model is applicable when $g$ is evaluated to true. Its generic form is $r(g)$. KP systems use a graph-like structure (similar to that of tissue P systems) and two types of rules:

(1) Rules to process objects: these rules are used to transform objects or to move objects inside compartments or between compartments. These rules are called rewriting, communication, and input-output rules.

(a) Rewriting and communication rule: $x \rightarrow y(g)$, where $x \in A^+ \setminus t_0, y \in A^+ \setminus t_0, g \in \text{finite regular expressions } FE \text{ over } (A \cup \bar{A})$; $y$ at the right side is defined as $y = (a_1, t_1) \cdots (a_h, t_h)$, where $a_i \in A$ and $t_j \in L$, $1 \leq j \leq h$, $a_j$ is an object, and $t_j$ is a target, respectively.

(b) The input-output rule: $(x/y)|g$, where $x, y \in A^+ \setminus t_0, g \in \text{finite regular expressions } FE \text{ over } (A \cup \bar{A})$; it means that $x$ can be sent from current compartment to the environment or $y$ can be brought from the environment to the target compartment.

(2) System structure rules: these rules make a fundamental change in the topology of the membranes, for
example, with division rule on a compartment, dissolution rule on a specific compartment, and making a link between compartments or dissolving the link between them. These rules are described as follows:

(c1) Division rule: \([l_i - l_i] \rightarrow [l_i, g]\), where \(g \in \) finite regular expressions FE over \((A \cup A)\); it means compartment \(l_i\) is replaced with \(h\) number of compartments. All newly created compartments inherit objects and links of \(l_i\).

(c2) Dissolution rule: \([l_i - l_i] \rightarrow \lambda\); it means compartment \(l_i\) does not exist anymore as well as all its links with other compartments.

(c3) Link-creation rule: \([l_i - l_i] \rightarrow [l_i, l_j]\); it means a link will be created between compartment \(l_i\) and compartment \(l_j\). If there is more than one compartment with the label \(l_j\), one of them will have a connection with \(l_j\) nondeterministically.

(c4) Link-destruction rule: \([l_i - l_i] \rightarrow [l_i, l_i]\); it means the existing link between \(l_i\) and \(l_j\) will be eliminated and there will not be any link between them anymore. The same as link creation, if there are more than one compartment which has a link with \(l_j\) then one of them will be selected nondeterministically to apply this rule.

2.2. The MOBPSO Approaches. Optimization problems with multiple goals or objectives are referred to as multiobjective optimization (MOO) problems. Therefore, the objectives may estimate different aspects of solutions, which are partially or wholly in conflict. MOO can be defined as follows: optimize objects \(Z = (f_1(x), f_2(x), \ldots, f_m(x))\), where \(x = (x_1, x_2, \ldots, x_m) \in X\). A multiobjective searching concept is clearly described in [24].

Graph structure is one of the models of feature selection for classification. For example, a graph-based MOBPSO algorithm [13] proposed to optimize average of node weights and edge weights at the same time through making different subgraphs. This algorithm is a feature selection model to highlight relevant and nonredundant genes in microarray datasets. The results of microarray datasets indicated that graph-based MOBPSO produces better performance in comparison to SBE, CFS, mRMR, and SFS methods from the classification accuracy point of view. Multiobjective optimization has been vastly used in evolutionary algorithms [25]. Although execution time in such a technique, which is known as optimization technique, is not efficient, its time complexity is not much higher relative to other comparable methods [13], MOBPSO is designed for maximizing the dissimilarity (negative correlation) and signal-to-noise ratio (SNR); (1) and (2) are represented as edge weight and node weight, respectively. The population is initialized by arbitrarily selected features from the data matrix, and population-fitness values are calculated using dissimilarity and SNR average values. The archive, \(A\), is initialized to the population value after nondominated sorting of the primary population. Velocity and position are updated using (3) and (4). The local best \(P\) is updated after comparing the current and previous fitness values of a particle, and the global best \(G\) is updated according to randomly picking a particle from the archive.

\[
\text{Dissimilarity} = \left(1 - \frac{\text{cov}(x, y)}{\sqrt{\text{var}(x) \text{var}(y)}}\right) \tag{1}
\]

\[
\text{SNR} = \left|\frac{\text{mean}(C1) - \text{mean}(C2)}{\text{s.d.(C1)} - \text{s.d.(C2)}}\right| \tag{2}
\]

\[
V(t+1) = w \ast v(t) + c1 \ast r1 \ast (pbest(t) - x(t)) + c2 \ast r2 \ast (gbest(t) - x(t)) \tag{3}
\]

\[
x(t+1) = x(t) + v(t+1) \tag{4}
\]

3. Proposed KP-MObPSO Model and Multiple Core KP-MObPSO

The entire process of the proposed KP-MObPSO model is summarized in Algorithm 1. It consists of four main phases, including (i) initialization, (ii) evolution, (iii) selecting the minimum \(gBestScore\), and (iv) collecting the marker gene. Each phase is built based on defined objects and rules.

To implement the exact proposed sequential KP-MObPSO feature selection method on multiple cores, we have defined four stages: initialization, evolution, interaction with client, and collecting the final result. Based on the assumption for the entire model, there are 25 number of particles \((p = 25)\) divided into 4 compartments (compartment 1, compartment 2, compartment 3, and compartment 4) having 6 particles in compartments 1 to 3 and 7 particles in compartment 4. The number of compartments is chosen based on the number of logical cores in the machine used for execution of the model. A total number of iterations is 100 times in evolutionary step.

Objects are defined as \(P\): number of particles, \(\text{Max}_{-c}\): maximum number of genes inside particles, position\(1\): position\(2\): \(n\) number of positions inside each particle, reserve, \(a: a_1 \cdots a_{100}\): data source of all genes, \(\text{Max}_{-c}\): maximum number of genes inside particles, NGENES, NewNGENES, Q, c, C, sum_diss, sum_snr, FIT, Q: selected gene IDs, \(pBestScore\). Rules are defined based on Table 1.

The symbol dictionary and explanation of rules in Table 1 are explained in the following before going through the steps. According to the MOBPSO algorithm, random numbers will be generated first to choose a random number of genes for further process. Thus, in Table 1, \(r1, r2, r3\) and \(r4\) are used to implement the first part of MOBPSO via kp rules. In \(r1\), we have defined \(\text{max}_{-c}\) and \(p\) which are two objects to present the maximum number of genes and a maximum number of particles, respectively. Through \(r1\), \(p\) number of membranes will be generated inside the membrane so-called position. Through \(r2\), multiple sets of
Begin
(i) Initialize
  Run $r_1 > r_2 > r_3 > r_4 > r_5 > r_6 > r_7$ once to initialize $p_{BestScore}$
  $it = 1$
(ii) Evolution
  (a) Run the rules $r_1 > r_2 > r_3 > r_4 > r_5 > r_6 > r_7 > r_8 > r_9 > r_{10} > r_{11} > r_{12} > r_{14} > r_{15} > r_{16}$
  by priority to get fitness value and replace with $p_{BestScore}$ in the case fitness < $p_{BestScore}$
  (b) Converge curve = min( fitness )
  (c) $w = p_{BestScore}$/Converge curve
  (d) if $it \geq 3$ and Converge curve is repeated the same amount at least three times,
      update the binary position of genes from 0 to 1 and from 1 to 0
  till $it = 100$
(iii) $g_{BestScore} = \min(Converge\ curve)$
(iv) collect marker genes

Algorithm 1: The entire process of the proposed KP-MObPSO model.

Table 1: Rules of KP-MObPSO.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>rewriting</td>
</tr>
<tr>
<td>$r_1 \equiv \left[ (p, max_c)[pos] \right]<em>{p_1} \rightarrow \left[ \left[ (pos^1 \cdot \cdot \cdot pos^n) \right]</em>{pos} \right]_{p_1}</td>
<td></td>
</tr>
<tr>
<td>r2</td>
<td>communication</td>
</tr>
<tr>
<td>$r_2 \equiv \left[ (pos^1 \cdot \cdot \cdot pos^n)<em>{pos} \right]</em>{p_1} \rightarrow \left[ \left[ (pos^1 \cdot \cdot \cdot pos^n)<em>{pos} \right]</em>{pos} \right]_{p_1}</td>
<td></td>
</tr>
<tr>
<td>r3</td>
<td>communication</td>
</tr>
<tr>
<td>$r_3 \equiv \left[ \left[ (pos^1 \cdot \cdot \cdot pos^n)<em>{pos} \right]</em>{pos} \right]<em>{p_1} \rightarrow \left[ \left[ (pos^1 \cdot \cdot \cdot pos^n)</em>{pos} \right]<em>{pos} \right]</em>{p_1}</td>
<td></td>
</tr>
<tr>
<td>r4</td>
<td>rewriting</td>
</tr>
<tr>
<td>$r_4 \equiv \left[ \left[ pos, a, max_c \right]<em>{p_2} \rightarrow \left[ \left[ pos, a, max_c \right]</em>{p_2} \right]_{pos}</td>
<td></td>
</tr>
<tr>
<td>r5</td>
<td>communication/rewriting</td>
</tr>
<tr>
<td>$r_5 \equiv \left[ \left[ \left[ pos, a, max_c \right]<em>{p_2} \rightarrow \left[ \left[ pos, a, max_c \right]</em>{p_2} \right]_{pos}</td>
<td></td>
</tr>
<tr>
<td>r6</td>
<td>Link creation</td>
</tr>
<tr>
<td>$r_6 \equiv \left[ \left[ pos \right]<em>{p_3} \rightarrow \left[ \left[ pos \right]</em>{p_3} \right]_{pos}</td>
<td></td>
</tr>
<tr>
<td>r7</td>
<td>communication/rewriting</td>
</tr>
<tr>
<td>$r_7 \equiv \left[ \left[ pos \right]<em>{p_3} \rightarrow \left[ \left[ pos \right]</em>{p_3} \right]_{pos}</td>
<td></td>
</tr>
<tr>
<td>r8</td>
<td>division</td>
</tr>
<tr>
<td>$r_8 \equiv \left[ \left[ [p]<em>{p_3} \rightarrow \left[ \left[ [p]</em>{p_3} \right]_{pos}</td>
<td></td>
</tr>
<tr>
<td>r9</td>
<td>membrane dissolution</td>
</tr>
<tr>
<td>$r_9 \equiv \left[ \left[ [p]<em>{p_3} \rightarrow \left[ \left[ [p]</em>{p_3} \right]_{pos}</td>
<td></td>
</tr>
<tr>
<td>R10</td>
<td>link creation</td>
</tr>
<tr>
<td>$r_{10} \equiv \left[ \left[ [p]<em>{p_3} \rightarrow \left[ \left[ [p]</em>{p_3} \right]_{pos}</td>
<td></td>
</tr>
<tr>
<td>R11</td>
<td>communication/rewriting</td>
</tr>
<tr>
<td>$r_{11} \equiv \left[ \left[ [p]<em>{p_3} \rightarrow \left[ \left[ [p]</em>{p_3} \right]_{pos}</td>
<td></td>
</tr>
</tbody>
</table>
random numbers which are $p$ sets in maximal will be generated. Through $r3$, each set of random numbers will enter the independent membranes so-called Ps.

In $r4$, with having an array of sample data called “$a$” and the random numbers generated and saved in position objects in the last stage we will choose related gene of random numbers generated and saved in position objects array another set, the so-called “catetowhichsampleeachcolumnbelongs”. This action makes “$Q$” keeps the value of the gene which carries the random number “position.” This processed gene is called NGENES. By adding a row at the end of the chosen genes, we can indicate to which sample each column belongs. This action makes another set, the so-called “NewGENES.” Other objects, like “$Q$” or “$c$,” will help to keep some value, for example, a total number of genes in each set; for example, “$c$” keeps the value of the total number of genes in each set and “$Q$” keeps the genes inside each compartment. $r5$ calculates dissimilarity and signal-to-noise value for each set of genes with the symbolic indications of “sum$_{\text{diss}}$” and “snr,” respectively. Then fitness value for each set will be calculated and shown by the symbol “FIT.” To find the best fitness value among the sets of genes, $r6$ will create a link among the membranes through a master membrane, the so-called “master.” Then through the $r7$, all the newly linked membranes will be able to communicate to extract the minimum fitness value, which will be called “pBestScore” for the first run and “gBestScore” for the second run, and the gene IDs have contributed to make this fitness value saved in the objects called “$Q$.” Through $r8$, a division rule will create new particles called Ps for the evolving session of selected genes to other ones. According to the MOBPSO itself, evolving the genes will produce objects such as $pBest$, $gBest$, Velocity, $c1$, $c2$, $w$, $V_{\text{max}}$, and $s$. Then, $r9$ can dissolve ($\Lambda$) previous Ps and all their connections with other membranes and $r10$ make new connections between the new Ps with the “master” membrane of the last session. In $r11$, after evolving the genes to new sets of genes inside each Ps then a calculation of “fitness” will be done, and it will compare with previous “pBestScore” and the value of “$pBest$” and “$gBest$” will initialize by “1.” Moreover, in the second round, the values of “pBestScore” and “gBestScore” will compare with each other. At the end, the minimum value of “pBestScore” will be the value of the object called “converge.” In $r12$, the position of the genes has been contributed in the particular process and will be saved in the membrane called “Master.” $r13$ basically explains the typical formulation of swarm optimization through the objects of position, $c1$, $c2$, $w$, $pBest$, $gBest$, $p$, max$_{\text{c}}$, rand to produce new velocity and new positions. The new positions will pass to $r14$ to make particles from newly generation positions for random genes and the process will continue again by new genes.

The explanation of objects and how they have evolved based on the rules are defined in different steps as Step 1: initialization, and Step 2: evolution. Moreover, Figure 1 also explains Steps 1 and 2 plus Step 3: interaction with client, and Step 4: collecting the final result.

Step 1 (initialization). Rules from the proposed sequential KP-MOBPSO translate into jobs in the proposed multiple core KP-MOBPSO feature selection method. As Algorithm 1 and Figure 1, first, binary positions of genes should be randomly initialized by the client. Thus, rule numbers $r1$, $r2$, and $r3$ in sequential KP-MOBPSO are defined as function $F_n = \text{Pos}$. To execute $F_n = \text{Pos}$ on four compartments in multiple core KP-MOBPSO, it is divided into four tasks including Pos1, Pos2, Pos3, and Pos4. Pos1 initializes the random position of genes inside 6 particles of compartment 1, Pos2 initializes random position of genes inside 6 particles
Function (Pos) sends result of \( y_1, y_2, y_3, y_4 \) as input for job1 in workers: Compartment 1, Compartment 2, ..., Compartment 4.

\[
\begin{align*}
y_1 &= \text{Pos}1(p, \text{max}_c), \quad y_2 = \text{Pos}2(p, \text{max}_c), \quad y_3 = \text{Pos}3(p, \text{max}_c), \quad y_4 = \text{Pos}4(p, \text{max}_c) 
\end{align*}
\]

### Workers

**Compartment 1**
- Particle 1, Particle 2, Particle 3
- Particle 4, Particle 5, Particle 6

**Rule1** = job1 = subgraph 1 = on particles of compartment 1
- task1 = op1, op2, op3 & c1, c2, c3, c4
- Task2 = op4

**Rule2** = job2 = Mycost1
- Task3 = Mycost2

**Rule3** = job3 = Mycost2
- Task4 = Mycost1

**Compartment 2**
- Particle 7, Particle 8, Particle 9
- Particle 10, Particle 11, Particle 12

**Rule1** = job1 = subgraph 1 = on particles of compartment 2
- task1 = Mycost1,1 = on particles of compartment 1
- task2 = Mycost1,2 = on particles of compartment 2
- task3 = Mycost1,3 = on particles of compartment 3
- Task4 = Mycost1,4 = on particles of compartment 4

**Compartment 3**
- Particle 13, Particle 14, Particle 15
- Particle 16, Particle 17, Particle 18

**Rule1** = job1 = subgraph 1 = on particles of compartment 3
- task1 = Mycost1,1 = on particles of compartment 1
- task2 = Mycost1,2 = on particles of compartment 2
- task3 = Mycost1,3 = on particles of compartment 3
- Task4 = Mycost1,4 = on particles of compartment 4

**Compartment 4**
- Particle 1, Particle 2, Particle 3
- Particle 4, Particle 5, Particle 6

**Rule1** = job1 = subgraph 1 = on particles of compartment 4
- task1 = Mycost1,1 = on particles of compartment 1
- task2 = Mycost1,2 = on particles of compartment 2
- task3 = Mycost1,3 = on particles of compartment 3
- Task4 = Mycost1,4 = on particles of compartment 4

Function (Mycost1) result of job2: SUM1, SUM2, SUM3, SUM4 as input for job3

Function (Mycost2) result of job3: fit1, fit2, fit3, fit4 initialize pBestScore1, pBestScore2, pBestScore3, pBestScore4 in client

---

**Figure 1:** Continued.
Function (Pos) sends result of: $y_1, y_2, y_3, y_4$ & pBestScore1, pBestScore2, pBestScore3, pBestScore4 as input for job1 in workers: Compartment 1, Compartment 2, Compartment 3, Compartment 4

$y_1 = pos_1, y_2 = pos_2, y_3 = pos_3, y_4 = pos_4$ & pBestScore1 = fit1, pBestScore2 = fit2, pBestScore3 = fit3, pBestScore4 = fit4

Figure 1: Continued.
KP-MObPSO. The outputs of job1 are translated from sequential KP-MObPSO to multiple core jobs as other rules.

3 jobs on the initialized particles of 4 compartments. These compartments combine as $C$, and all parameters in all 25 particles after each iteration. Minimum values have fallen in a local trap or not. If the value of the convergence curve variable is the minimum of $pBestScores$ in all 25 particles after each iteration repeatedly is the same value for three times, then all the positions of genes will reverse in client and new positions will launch new iteration continuously.

Step 3 (interaction with client). Totally evolutionary steps will be executed 100 times. However, after each iteration, results of workers should combine together in client section to initialize new iteration. To do so, according to Figure 1, in client section all the results of updated positions in 4 compartments combine as $yy$, all the results of updated $pBestScores$ in 4 compartments combine as $pBestScore$, and all the results of a random number of genes inside each particle combine as $C$. By these combinations second and third iterations will launch. For iteration number 3 and above, the client checks whether the production of $pBestScores$ has fallen in a local trap or not. If the value of the convergence curve variable which is the minimum of $pBestScores$ in all 25 particles after each iteration repeatedly is the same value for three times, then all the positions of genes will reverse in client and new positions will launch new iteration continuously.

Step 4 (collecting final result). After 100 times the iteration, value of the convergence curve variable gathers a minimum of $pBestScores$ in all 25 particles after each iteration. Minimum value in the convergence curve matrix is the record of 100 minimum $pBestScores$ calls $gBestScore$. The set of the gene numbers has already produced such a minimum value of $gBestScore$ which are marker genes. These marker genes are the final result of KP-MObPSO feature selection method.

4. Execution of KP-MObPSO on Multicore Processing

To implement multiple core processing of KP-MObPSO, we integrate the two types of parallel programming. First, all the

Client


$$pBestScore (1,1:6) = pBestScore1 (1,1:6), pBestScore (1,7:12) = pBestScore2 (1,7:12), pBestScore (1,13:18) = pBestScore3 (1,13:18), pBestScore (1,19:25) = pBestScore (1,19:25)$$

$$W_1 (1:6,1) = pBestScore1 / convergence curve (1,1), W_2(7:12,1) = pBestScore2 / convergence curve (1,1), W_3(13:18,1) = pBestScore3 / convergence curve (1,1), W_4(19:25,1) = pBestScore4 / convergence curve (1,1)$$

$$C (1:6,1) = c_1(1:6,1), C (7:12,1) = c_2(7:12,1), C (13:18,1) = c_3(13:18,1), C (19:25,1) = c_4(19:25,1)$$

If trapped in local optimization

Change all the binary position of genes from 1 to 0 and from 0 to 1 and update $yy1 \ldots yy4$ with new positions

End


Figure 1: KP-MObPSO on multicore processing.
functions are defined in sequential MObPSO and rewritten based on Parfor independent loops. All the functions including Pos, subgraph, Mycost1, and Mycost2 are rewritten based on 25 independent Parfor loops. Second, each function is defined as a job based on createJob method to be executable on the particles. Particles are assumed to be located inside four compartments, and each job consists of four tasks each running on one compartment.

4.1. Parallel Computing. Computing in a parallel way is applicable through various methods such as clusters, multicores, and GPUs. The main aim in all parallel computing methods is to increase the speed of computing through using more processors and more memory compared with a single machine. From the programmers’ point of view, languages such as C/C++/FORTRAN and MPI (Message Passing Interface) are hard to use, and languages such as C/FORTRAN and MPI are time consuming. That is the reason programming via MATLAB is highly preferred by programmers due to applying high-level language and use of mix features that can produce productive code. Using MATLAB for parallel computing support may lead to decrease in computing time through options such as parameter sweeping. In fact, parallel computing in MATLAB can take advantage of two types of parallelism, first implicit multiprocessing which uses built-in multithreading and second explicit multiprocessing which uses toolboxes and distributed computing on clusters. The explicit multiprocessing method is employed in this paper. According to Figure 2 that categorized parallel computing options based on ease of use and greater control opportunity, we will use Parfor and createJob to take advantages of greater control simultaneous with having ease of use.

Parfor: A Parallel FOR Loop. The key word, “Parfor,” as in Figure 3, is defined to use instead of “for” keyword where a parallel loop is needed to parallelize an instruction. The difference is that each iteration in Parfor divided between different “workers” and the execution order of iterations is not known, although the results of different executions will be gathered and will be sent to MATLAB. Parfor is the simplest way to parallelize a loop statement. As usual execution starts by executing client codes and when it reaches Parfor, the iterations automatically divide between workers, and after all executions are done, the result will be collected in client again. As a prerequisite, it is necessary that all iterations run as an independent loop due to some limitation in access to the stored data. Thus, every loop in parallel computing makes an opportunity to distribute a series of independent tasks on a series of workers. Detection of workers and exchanging whatever necessary data between client and workers will be done automatically through MATLAB itself.

CreateJob: Life Cycle of a Job. The process of executing job from client side to workers is illustrated in Figure 4, which is called job's life cycle. As it is shown in Figure 4, the life cycle of a job is a process that starts with creating job on client side and finishes by running a job on workers. Between starting and finishing point there are two other stages, the so-called pending phase and queueing phase. In pending phase, the job is already created on scheduler through the command createJob function and tasks are already added to the jobs. In queueing phase, the pending job is already submitted to the queue and according to the sequence of submitting scheduler will send jobs for execution until all queued jobs are finished. The last stage for a job is when it reaches the top of the queue and will be sent for execution phase. In running phase scheduler will distribute job’s tasks to different workers to run.
In this case, more workers are available compared with the number of existing job’s tasks; the scheduler will send another job from the queue to running phase. It means more than one job will be in running phase at a time. When the job’s execution finishes, the result will be gathered with the function “fetchOutputs.” Any problem occurs when the scheduler attempts to run a job’s tasks will lead to a fail error; moreover, in the case of deleting a job, its status will be updated to delete one and its state will be available all the time the job objects are still in the client.

Configuration. We define parallel configuration on MATLAB by cluster, and we have defined jobs and jobs’ tasks in the MATLAB client. As predefined, the number of physical cores on system indicates the maximum number of workers in MATLABPOOL. In this study, there are 4 cores in our machine; therefore, we change the configuration as follows:

Parallel > Manage Cluster Profiles > local > Edit and change the value of “Number of workers to start on your local machine” to 4

The proposed KP-MObPSO feature selection methods assume to have 25 particles. To configure 25 particles on 4 cores of the system, we assume to have 4 compartments including 6 particles, 6 particles, 6 particles, and 7 particles, respectively. Each particle consists of a random number of gene objects from 1 to 100 and sets of rules will be executed on objects. Since then, we will assign each rule to a specific job and each job will assign to each compartment through a set of 4 tasks repeating execution of rules on each object inside a particle. Thus, the multicore KP-MObPSO model will be initialized by

(i) 4 compartments including 6, 6, 6, and 7 particles,
(ii) 4 jobs/rules inside each particle,
(iii) 4 tasks assigned to each rule/jobs to be executed on particles inside the 4 compartments,
(iv) priority of rules Pos > subgraph > Mycost1 > Mycost2 > compare > velocity being effective in the priority of tasks.

5. Material and Method

In this paper, cell line datasets of colorectal cancer and breast cancer are downloaded from publicly available datasets in the Gene Expression Omnibus (GEO) repository (https://www.ncbi.nlm.nih.gov/gds).

5.1. Methods and Data. Totally, six samples of colorectal cancer are used according to Tables 1 and 2, respectively, explaining the specification of samples. When samples of a microarray dataset represent normal (benign) and cancer (malignant) tissue, classifying such samples is referred to as binary classification. Otherwise, when samples represent various subtypes of cancer, classification is known as multiclass cancer classification. In both cases, genes with significantly different expression in the two categories (normal and tumor or two different subtypes of cancer) are labeled as differentially expressed genes. In this paper, our aim was to find these indicator genes or marker genes, to distinguish normal and tumor genes (binary classification) in colorectal dataset through multicore KP-MObPSO method faster than its counterpart sequential KP-MObPSO method.

5.2. Colorectal Data. According to Table 2, samples were provided from two subtypes including subtype 1.1 and 2.1 with platform IDs: GPL570 (Affymetrix U133 Plus 2.0) and series GSE35896 to compare between 62 colorectal cancers. First three samples, including GSM877130, GSM877141, and GSM877142, were from the subtype 1.1, and the last three samples, GSM877127, GSM877138, and GSM877140, were from subtype 2.1. Before evaluating the model, we preprocessed the data. The number of genes in the raw data was 54676. A list of significant genes, including normal and tumor genes, was highlighted according to the results of a study [8] where the clinical, pathological, and biological features of different subtypes of colorectal tumors were compared. As a result, 382 tumor genes were selected according to the subtypes 1.1 and 2.1 in the initial cell line dataset. The SNR was calculated according to (5) for both normal and tumor genes. For each gene, the first three sets of samples were categorized as C1 (subtype 1.1), and the rest were classified as C2 (subtypes 2.1). The SNR values were sorted in descending order, and 100
Table 2: Specification of colorectal cancer dataset.

<table>
<thead>
<tr>
<th>Samples</th>
<th>Cell line</th>
<th>Tissue</th>
<th>Disease state</th>
<th>Genotype/variation</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSM877130</td>
<td>CRC_42</td>
<td>Large intestine</td>
<td>Adenocarcinoma</td>
<td>microsatellite.status: MSS</td>
</tr>
<tr>
<td>GSM877141</td>
<td>CRC_45</td>
<td>Large intestine</td>
<td>Adenocarcinoma</td>
<td>microsatellite.status: MSS</td>
</tr>
<tr>
<td>GSM877142</td>
<td>CRC_61</td>
<td>Large intestine</td>
<td>Adenocarcinoma</td>
<td>microsatellite.status: MSS</td>
</tr>
<tr>
<td>GSM877127</td>
<td>CRC_01</td>
<td>Large intestine</td>
<td>Adenocarcinoma</td>
<td>microsatellite.status: MSI</td>
</tr>
<tr>
<td>GSM877138</td>
<td>CRC_02</td>
<td>Large intestine</td>
<td>Adenocarcinoma</td>
<td>microsatellite.status: MSI</td>
</tr>
<tr>
<td>GSM877140</td>
<td>CRC_19</td>
<td>Large Intestine</td>
<td>Adenocarcinoma</td>
<td>microsatellite.status: MSI</td>
</tr>
</tbody>
</table>

maximum SNR values (50 maximum SNR values for normal genes and 50 maximum SNR values for tumor genes) were selected for further analysis. Then the signal-to-noise ratio (SNR) value (node weight) corresponding to each feature is calculated using mean and standard deviation (s.d.) of class 1 ($c_1$) and class 2 ($c_2$).

During the final stage of preprocessing, the 100 genes were normalized according to (6). For normalization, minimum and the maximum value of each gene (column) are calculated first. Then normalization is done, where $g_{ij}$ presents gene expression value of the $i$th sample of the $j$th gene and $j$ presents gene expression values of $j$-th gene. Therefore, all the values of the dataset (100 rows of genes times six sets of samples) were transformed to a value between 0 and 1 [0, 1].

\[
|SNR| = \left| \frac{\text{mean}(c_1) - \text{mean}(c_2)}{\text{s.d.}(c_1) + \text{s.d.}(c_2)} \right| \quad (5)
\]

\[
\text{Normalize}(g_{ij}) = \frac{g_{ij} - \text{minimum}(g_{ij})}{\text{maximum}(g_{ij}) - \text{minimum}(g_{ij})} \quad (6)
\]

The prepared data for colorectal dataset are evaluated by KP-MObPSO and MObPSO to search for gene markers. MATLAB R2014a was used for object-oriented coding of the entire proposed model. Then, Weka 3.6.9 is used to classify the marker genes resulting from KP-MObPSO and MObPSO. Classification accuracy and ROC by Weka are used for comparison. Using the Weka software, SMO (Implements, John Platt's sequential minimal optimization algorithm for training a support vector classifier) works based on support vector machine theory. A linear SMO ($K(x, y) = \langle x, y \rangle$) works to find the largest margin for separating hyperplane. It is defined according to the aggregation of distances from hyperplane to the most nearest positive or negative points. It is projected that as the margin gets larger the classifier will be more general. When the case is not separable, linear support vector machine looks for a trade-off that maximizes the margin and on the other hand minimizes errors.

6. Evaluation and Discussion

The aim of our proposed model (KP-MObPSO) was to build a more robust feature selection method about MObPSO. Therefore, to evaluate the performance of our proposed feature selection method for classification, we needed to consider the classification accuracy, which is the number of correct predictions made divided by the total number of predictions, multiplied by 100 (7). The numbers of true negatives (TN), false negatives (FN), true positives (TP), and false positives (FP) are used to compute the efficiency of the classifier.

\[
\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \times 100\% \quad (7)
\]
Table 3: Performance metrics for classification accuracy for KP-MObPSO and MObPSO.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>ROC area</th>
<th>F-measure</th>
<th>Sensitivity</th>
<th>Specificity</th>
</tr>
</thead>
<tbody>
<tr>
<td>KP-MObPSO</td>
<td>87.50%</td>
<td>0.90</td>
<td>0.877</td>
<td>0.875</td>
<td>0.906</td>
</tr>
<tr>
<td>MObPSO</td>
<td>78.57%</td>
<td>0.72</td>
<td>0.785</td>
<td>0.786</td>
<td>0.792</td>
</tr>
</tbody>
</table>

To evaluate the performance of our proposed feature selection model for binary classification,

\[
\text{Precision} = \frac{TP}{TP + FP}
\]

\[
\text{Recall} = \frac{TP}{TP + FN}
\]

\[
F = 2 \times \text{Precision} \times \text{Recall} \quad (8)
\]

To evaluate the accuracy of the proposed feature selection model, we use support vector machines. We divided 70% of the dataset into training and testing sets, and the proposed approach was applied to the training set. A candidate solution was obtained, with which the corresponding testing set was used for validation.

According to Table 3, regarding classification accuracy according to (7), there were 87.50% correctly classified genes from the KP-MObPSO method, which was higher than the 78.57% reported from the MObPSO method. Measures like ROC area, F-measure, sensitivity, and specificity of KP-MObPSO which are greater than MObPSO indicate much more reliable model.

The proposed KP-MObPSO model is parallelized on multicore. In general, the main memory in parallel computer hardware can be classified into two kinds: shared memory and distributed memory. Shared memory is all processors interconnections with significant logical memory. Distributed memory refers to the fact that each processor has its own local memory. Indeed, accesses to local memory are typically faster than accesses to nonlocal memory [26, 27]. Figure 5 illustrates architectural differences between distributed and shared memory in the trace of objects based on the process of rules in KP-MObPSO as discussed before in Table 1.

Multicore chips do more work per clock cycle, are able to run at a lower frequency, and may enhance the performance of computation time. MATLAB includes the tic and toc functions which set a starting time and then return the elapsed time. The same command can be used in parallel. The MATLAB pool command itself can take some time. Typically, we want to ignore that and focus on the parallel computation. In the proposed multicore KP-MObPSO method, after launching workers through "yy," the outputs of jobs like "op, c, SUM, fit, gBest, pBest, and yy" need to be saved in client to send to the next jobs as input arguments. This frequent interaction between clients and workers takes time and does not lead to more improvement in timely execution of multicore KP-MObPSO.

7. Conclusion and Future Work

Due to the inherent large-scale parallelism feature of membrane computing, any membrane computing inspired model can fully represent this computation model only in the case of using the parallel platform. From the beginning of introducing this model, it was a big concern in all membrane related studies. For instance, to fully implement parallelism of such membrane computing model and to support an efficient execution [28] used a platform based on reconfigurable hardware. Without parallelism, all subsequent studies face a challenge of how to make rules available in all steps of computation. In [29] with sequential computing of membrane computing, the authors just had an option of using one membrane and made the rules periodically available based on time-varying sequential P system. Even by using minimal parallelism of rules more efficient model of membrane computing is achievable. This is because of the trading space for time. For example, in the study of [30] better performance has already achieved which is due to the parallel implementation of at least one rule from a set of rules to solve NP-complete problems in polynomial time.

Recently, several studies attempted to utilize membrane computing to improve intelligent algorithms. For instance,
multicore processing used in the study of [31] utilized a membrane computing inspired genetic algorithm and [32] has highlighted parallelism in membrane computing in the case of solving the N-queens problem.

Regarding the modelling of particle swarm optimization via kernel P system, in the one hand, useful features of KP system, for example, its capability for changing the structure within modelling, and the rules such as division, dissolution, and link creation make variation of a P system more powerful to compare with those having only rewriting and communication rules. However, on the other hand, the most challenging part is the implementation of division rule when new membrane born to facilitate particle creation in PSO and even the multiple membranes born through division rule to facilitate multi-PSO.

The kernel P system as a variant of a P system is used to apply various rules of the kernel P system including communication/rewriting, division, input/output, link creation, and link dissolution. Thus, the model is not static; it generates compartments by division rule and makes links between compartments and dissolves them by time. The MObpPSO model is built based on kernel P system rules and the proposed model called KP-MObPSO. To examine the performance of hybrid KP-MObPSO model to compare with pure MObpPSO, a real dataset of colorectal cancer dataset is tested on both models including the hybrid and the pure one. According to the methodology, preprocessing steps include filtering the initial dataset based on SNR factor and then normalization performed on the raw dataset. The results came out from executing preprocessed datasets on the proposed KP-MObPSO model. The results of performance measures indicate that empowering the MObpPSO by kernel P system leads to improving accuracy, recall, and F-measure. It proves KP-MObPSO is a more robust model in comparison with pure MObpPSO and the first objective of the paper is fulfilled through that.

The second objective of the paper is taking full advantage of membrane computing via parallel implementation of kernel P system rules. The proposed KP-MObPSO model is implemented on multicore and it made all the rules available at the same time for \( n \) number of particles and swarm optimization for all the objects inside the \( n \) number of particles can run in parallel. The architectural differences between CPUs and GPUs cause CPUs to perform better on latency-sensitive, partially sequential, single sets of tasks. In contrast, GPUs performs better with latency-tolerant, highly parallel, and independent tasks. Executing KP-MObPSO on multicore GPU will be considered as a solution for time improvement in future work.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper. The mentioned received funding in "Acknowledgments" did not lead to any conflicts of interest regarding the publication of this manuscript.

Acknowledgments

This work has been supported by the Science Fund of the Ministry of Science, Technology, and Innovation (MOSTI) (Malaysia; Grant code: 01-01-02-SFI104) and FRGS/2/2015/ICT04/UKM/02/3.

References


Submit your manuscripts at https://www.hindawi.com