International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:13, No:10, 2019

A Bacterial Foraging Optimization Algorithm Applied to the Synthesis of Polyacrylamide Hydrogels

Florin Leon, Silvia Curteanu

***Abstract*—**The Bacterial Foraging Optimization (BFO) algorithm is inspired by the behavior of bacteria such as *Escherichia coli* or *Myxococcus xanthus* when searching for food, more precisely the chemotaxis behavior. Bacteria perceive chemical gradients in the environment, such as nutrients, and also other individual bacteria, and move toward or in the opposite direction to those signals. The application example considered as a case study consists in establishing the dependency between the reaction yield of hydrogels based on polyacrylamide and the working conditions such as time, temperature, monomer, initiator, crosslinking agent and inclusion polymer concentrations, as well as type of the polymer added. This process is modeled with a neural network which is included in an optimization procedure based on BFO. An experimental study of BFO parameters is performed. The results show that the algorithm is quite robust and can obtain good results for diverse combinations of parameter values.

***Keywords*—**Bacterial foraging optimization, hydrogels, neural networks, modeling.

I. Introduction

SEMI- or inter-penetrating multicomponent network-type hydrogels are materials with three-dimensional structure, characterized by a high swelling capacity. Due to the high water retention ability, i.e. more than 90%, such networks possess a flexibility similar to that of the natural tissue.

Special properties such as mechanical, diffusion and absorption, of the three-dimensional network-type hydrogels make possible their use in various domains: food, cosmetics, pharmaceutical industry, medicine, tissue engineering, agriculture, electrotechnics, electronics, etc. [1]-[6].

For many applications, e.g. controlled-release systems, agrochemical products, these materials are required to present a high capacity of biodegradation under the action of the biologic fluids or of the microorganisms present in the soil.

The multicomponent networks based on polyacrylamide (PAAm) are particularly appropriate for biomedical, pharmaceutical or agricultural applications. Thus, due to the analgesic effects and to their ability to speed up the healing process, polyacrylamide hydrogels (PAAm - 5% and oxygenated water - 95%) are used in aesthetic surgery [5], [7].

Polyacrylamide hydrogels present selective biodegradability under the action of the gastro-intestinal juices, so that they can be used as covering agent for tablets to protect the active principle, to conceal the non-agreeable taste and smell, as well as to control the release of the active principle. Three-dimensional networks based on polyacrylamide are used in ophthalmology as mechanical protectors for iris, retina and corneal endothelia [5].

Polyacrylamide gels are very useful for soil improvement, i.e. for the stabilization of sandy soils, in promoting the selective sorption of nutrients by plants, in increasing the permeability and agricultural efficiency of non-structured soils or stopping of erosion [6], [8], [9]. Multicomponent networks based on polyacrylamide are used as controlled release systems for fertilizers and present a high efficiency in the cultivation of saplings and plants on reduced surfaces, e.g. green houses or gardens.

The present case study deals with the synthesis and swelling behavior of semi- and interpenetrated multicomponent networks based on polyacrylamide. The networks were prepared by a "single step" process of polymerization/ crosslinking and consist of a polyacrylamide matrix and a biodegradable interpenetrated polymer [10], [11].

The database was prepared in a previous study [12]. In this paper, we use a different optimization method, namely the BFO algorithm in order to find the appropriate values of the reaction conditions that maximize the yield in crosslinked polymer.

We organize our paper as follows. In Section II, we include a general description of the BFO algorithm, Section III presents an original software application that implements the algorithm and in Section IV we show the case study that involves the hydrogels dataset and the experimental results. Finally, Section V contains the conclusions of our work.

II. BFO

The BFO algorithm is inspired by the behavior of bacteria such as *Escherichia coli* or *Myxococcus xanthus* when they search for food, namely chemotaxis. Bacteria perceive chemical gradients in their surrounding environment, such as nutrients, and move toward or in the opposite direction to those signals. In addition, bacteria produce attractive or repulsive substances in the environment, which are perceived by other bacteria. Using locomotion mechanisms such as flagella, bacteria can move, sometimes randomly by tumbling or spinning, and other times in an orderly manner similar to swimming. Bacterial cells are equivalent to some agents in an environment that use their perception of food and other bacteria to move. Depending on their interactions with one-another, bacteria can self-organize in swarms around a feeding source, ignore or reject each other.

F. Leon is with the Department of Computer Science and Engineering, "Gheorghe Asachi" Technical University of Iași, Bd. Mangeron 27, 700050 Iași, Romania (e-mail: florin.leon@tuiasi.ro)

S. Curteanu is with the Department of Chemical Engineering, "Gheorghe Asachi" Technical University of Iași, Bd. Mangeron 73, 700050 Iași, Romania (e-mail: scurtean@ch.tuiasi.ro)

The strategy of the algorithm is to allow cells to evolve in a stochastic and collective way to the optimum. This is done through three processes:

- *Chemotaxis*, according to which the cost of a bacteria is reduced by the proximity of other cells, and bacteria move along the cost surface one by one;
- *Reproduction*, in which only those bacteria that have had good results during their lifetimes can contribute to the next generation;
- *Elimination-dispersion* in which, with a low probability, some bacteria are eliminated from the population and new randomly initialized bacteria are introduced into the population.

The cost of a cell (or bacteria) is reduced by the interaction with other cells. This interaction function g has the following expression:

$$g(c_k) = \sum_{i=1}^{S}\left[-d_a \cdot \exp\left(-w_a \cdot \sum_{m=1}^{P}\left(c_m^k - o_m^i\right)^2\right)\right] + \sum_{i=1}^{S}\left[h_r \cdot \exp\left(-w_r \cdot \sum_{m=1}^{P}\left(c_m^k - d_m^i\right)^2\right)\right] \quad (1)$$

where c_m^k is the cell for which the function is computed, o_m^i is a neighbor cell, d_a and w_a are attraction coefficients, h_r and w_r are repulsion coefficients, S is the number of cells in the population, and P is the number of dimensions of the position vectors, i.e. the size of the problem.

Other parameters involved in the algorithm are: the number of elimination-dispersion steps, the number of reproduction steps, the number of chemotaxis steps, the number of swim steps and the probability of a cell being eliminated from the population.

A description of the BFO algorithm and its applications can also be found in [13]-[18].

The algorithm was designed for continuous domains. Given the loops in the algorithm, it can be configured in various ways to determine different search behaviors. Generally, it is recommended to have a large number of chemotaxis iterations and a small number of other iterations.

The default coefficients recommended by the authors for the search behavior are the following: $d_a = 0.1$, $w_a = 0.2$, $h_r = d_a$ and $w_r = 10$. The step size is usually a small fraction of the search space, for example 0.1. During reproduction, usually half of the population, i.e. the cells with low health, is eliminated, and two copies of each bacteria from the first half of the population are kept. The probability of elimination and dispersal is usually high, e.g. 0.25.

III. Software Application

Based on the algorithm, a software application was implemented in the C # programming language, whose source code is presented below. Fig. 1 shows the graphical user interface of the program. As a first example, a very simple function is considered:

$$\min f(\mathbf{x}) = (x_1 - 1)^2 + (x_2 + 2)^2 \quad (2)$$

with solution $\mathbf{x}^*$ = (1, –2). Then, we will describe the optimization of a more complex function, learned from the chemical process related to hydrogels.

In the source code of the main window of the application, one only needs to instantiate a *BacterialOptimization* object, and call its *Search* method:

```
private void runToolStripMenuItem_Click(object sender, EventArgs
e)
{
   BacterialOptimization bo = new BacterialOptimization();
   Cell sol = bo.Search();

   richTextBox.AppendText(string.Format("Location: {0:F6}
{1:F6}\r\n",
      sol.Vector[0], sol.Vector[1]));
   richTextBox.AppendText(string.Format("Cost: {0:F6}\r\nFitness:
      {1:F6}\r\nInter: {2:F6}\r\nSum nutrients: {3:F6}\r\n\r\n",
      sol.Cost, sol.Fitness, sol.Inter, sol.SumNutrients));
   }
}
```

The class that encodes a bacterium or a cell is called *Cell*. Its state is given by the following C# properties:

```
public double[] Vector { get; set; }
public double Cost { get; set; }
public double Inter { get; set; }
public double Fitness { get; set; }
public double SumNutrients { get; set; }
```

It has both a default constructor and a copy constructor:

```
public Cell()
{ }

public Cell(Cell c)
{
   this.Vector = new double[c.Vector.Length];
      for (int i = 0; i < c.Vector.Length; i++)
         this.Vector[i] = c.Vector[i];

   this.Cost = c.Cost;
   this.Inter = c.Inter;
   this.Fitness = c.Fitness;
   this.SumNutrients = c.SumNutrients;
   }
}
```

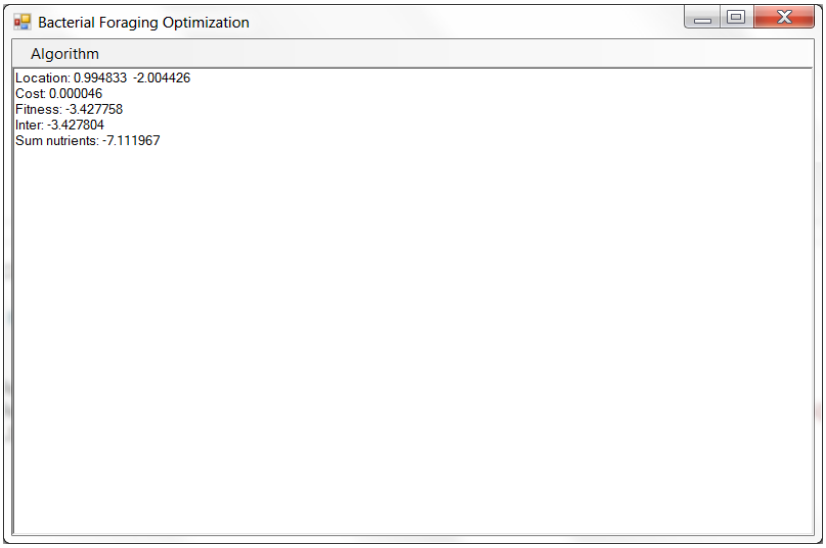

Fig. 1 The graphical user interface of the optimization application

TABLE I
EXPERIMENTAL RESULTS OBTAINED WITH THE BFO ALGORITHM FOR DIFFERENT VALUE COMBINATIONS OF THE PARAMETERS

S	s_s	s_{ed}	s_r	s_c	s_l	p_{ed}	d_a	w_a	h_r	w_r	μ	σ
10	0.1	1	4	100	4	0.25	0.1	0.2	0.1	10	98.9453	0.9727
10	0.1	1	4	20	4	0.25	0.1	0.2	0.1	10	98.7178	2.7188
30	0.1	1	4	20	4	0.25	0.1	0.2	0.1	10	99.7188	0.3767
10	0.1	1	2	20	4	0.25	0.1	0.2	0.1	10	98.8828	0.9755
10	0.1	1	2	20	4	0.05	0.1	0.2	0.1	10	98.8428	2.9096
10	0.1	1	2	20	4	0.5	0.1	0.2	0.1	10	98.8272	1.9623
10	0.1	1	2	20	4	0.25	0.2	0.4	0.2	20	98.2247	3.2784
10	0.1	1	2	20	4	0.25	0.2	0.4	0.05	5	98.5909	3.4324
10	0.1	1	2	20	4	0.25	0.05	0.1	0.2	20	98.3708	3.9522
2	0.1	1	2	20	4	0.25	0.1	0.2	0.1	10	80.4822	18.2642
10	1	1	2	20	4	0.25	0.1	0.2	0.1	10	97.0839	5.3906
10	0.01	1	2	20	4	0.25	0.1	0.2	0.1	10	96.2224	6.0530
200	0.1	1	2	20	4	0.25	0.1	0.2	0.1	10	99.9964	0.0260

The class that performs the actual optimization is *BacterialOptimization*. Its fields are presented as follows. As it can be seen, it maintains a population of bacteria, implemented as a list of *Cell* objects:

```
private int _problemSize = 2;
private double[,] _searchSpace;
private int _populationSize = 50; // should be even
private double _stepSize = 0.1;
private int _elimDispSteps = 1;
private int _reproSteps = 4;
private int _chemSteps = 100;
private int _swimLength = 4;
private double _pEliminate = 0.25;
private double _dAttr = 0.1;
private double _wAttr = 0.2;
private double _hRep = 0.1; // = _dAttr;
private double _wRep = 10;
private Random _rand = new Random();
private List<Cell> _cells;
```

In the constructor, the user must define the search space. As a first example, in the case below we search between -5 and 5, for the two dimensions of the problem in (2).

```
public BacterialOptimization()
{
   _searchSpace = new double[,] { { -5, 5 }, { -5, 5 } };
}
```

In the *ObjectiveFunction* method, the user must include the actual function that needs to be optimized, i.e. the function in (2).

```
private double ObjectiveFunction(double[] x)
{
   return (x[0] - 1) * (x[0] - 1) + (x[1] + 2) * (x[1] + 2);
}
```

The rest of the class implements the algorithm-specific operations.

```
private double[] RandomVector()
```

```
{
   double[] v = new double[_problemSize];
   for (int i = 0; i < _problemSize; i++)
      v[i] = _searchSpace[i, 0] + (_searchSpace[i, 1] - _searchSpace[i,
0]) *
         _rand.NextDouble();
   return v;
}

private double[] GenerateRandomDirection()
{
   return RandomVector();
}
```

```
private double ComputeCellInteraction(Cell cell, double d, double w)
{
   double sum = 0;

   foreach (Cell other in _cells)
   {
      double diff = 0;

      for (int i = 0; i < _problemSize; i++)
      {
         diff += (cell.Vector[i] - other.Vector[i]) * (cell.Vector[i] –
            other.Vector[i]);
      }

      sum += d * Math.Exp(w * diff);
   }

   return sum;
   }
```

```
private double AttractRepel(Cell cell)
{
   double attract = ComputeCellInteraction(cell, -_dAttr, -_wAttr);
   double repel = ComputeCellInteraction(cell, _hRep, -_wRep);
   return attract + repel;
}
```

```
private void Evaluate(Cell cell)
{
   cell.Cost = ObjectiveFunction(cell.Vector);
   cell.Inter = AttractRepel(cell);
   cell.Fitness = cell.Cost + cell.Inter;
}
```

```
private Cell Tumble(Cell cell)
{
   double[] step = GenerateRandomDirection();
   double[] vector = new double[_problemSize];

   for (int i = 0; i < _problemSize; i++)
   {
      vector[i] = cell.Vector[i] + _stepSize * step[i];
      if (vector[i] < _searchSpace[i, 0])
         vector[i] = _searchSpace[i, 0];
      if (vector[i] > _searchSpace[i, 1])
         vector[i] = _searchSpace[i, 1];
   }

   Cell newCell = new Cell();
   newCell.Vector = vector;
   return newCell;
}
```

```
private Cell Chemotaxis()
{
   Cell best = null;

   for (int j = 0; j < _chemSteps; j++)
   {
      List<Cell> movedCells = new List<Cell>();

      for (int i = 0; i < _cells.Count; i++)
      {
         double sumNutrients = 0;
         Evaluate(_cells[i]);

         if (best == null || _cells[i].Cost < best.Cost)
            best = _cells[i];

         sumNutrients += _cells[i].Fitness;

         for (int m = 0; m < _swimLength; m++)
         {
            Cell newCell = Tumble(_cells[i]);
            Evaluate(newCell);
            if (_cells[i].Cost < best.Cost)
               best = _cells[i];

            if (newCell.Fitness > _cells[i].Fitness)
               break;

            _cells[i] = newCell;

            sumNutrients += _cells[i].Fitness;
         }

         _cells[i].SumNutrients = sumNutrients;
         movedCells.Add(_cells[i]);
      }

      _cells = movedCells;
   }

   return best;
}
```

```
public Cell Search()
{
   _cells = new List<Cell>();
   for (int i = 0; i < _populationSize; i++)
   {
      Cell c = new Cell();
      c.Vector = RandomVector();
      _cells.Add(c);
   }

   Cell best = null;
   for (int l = 0; l < _elimDispSteps; l++)
   {
```

```
    for (int k = 0; k < _reproSteps; k++)
    {
      Cell cBest = Chemotaxis();
      if (best == null || cBest.Cost < best.Cost)
        best = cBest;

      Cell[] sortedCells = ShellSort(_cells);

      for (int ii = 0; ii < _populationSize / 2; ii++)
      {
        _cells[ii * 2] = sortedCells[ii];
        _cells[ii * 2 + 1] = sortedCells[ii];
      }
    }

    foreach (Cell cell in _cells)
    {
      if (_rand.NextDouble() <= _pEliminate)
        cell.Vector = RandomVector();
    }
  }

  return best;
}
```

```
private Cell[] ShellSort(List<Cell> cells)
{
  int dist = 0, i = 0, j = 0;
  Cell aux;
  int count = cells.Count;

  Cell[] newCells = new Cell[count];
  for (i = 0; i < count; i++)
    newCells[i] = new Cell(cells[i]);

  for (dist = count / 2; dist > 0; dist /= 2)
    for (i = dist; i < count; i++)

      for (j = i - dist; j >= 0 && newCells[j].SumNutrients <
        newCells[j + dist].SumNutrients; j -= dist)
      {
        aux = newCells[j];
        newCells[j] = newCells[j + dist];
        newCells[j + dist] = aux;
      }

  return newCells;
  }
}
```

IV. Case Study: The Hydrogels Dataset

The hydrogels dataset contains 177 experimental data, available from the preparation of the multicomponent hydrogels based on polyacrylamide. The experiments were designed to cover uniformly the variation domain of the parameters; this is an important requirement for obtaining a reliable mathematical model. The chosen input variables of the network are the reaction conditions that influence the polymer yield.

These seven input variables are the following: C_M (monomer concentration), C_I (initiator concentration), C_A (crosslinking agent concentration), PI (the amount of inclusion polymer), T (temperature), t (reaction time) and the type of the included polymer codified as 1: no polymer added, 2: starch, 3: PVA and 4: gelatin. The output of the neural model is η, the yield in the crosslinked polymer. Therefore, a neural network is used to model the influence of reaction conditions on reaction yield. Then, the BFO algorithm is used to optimize the inputs such that the output (reaction yield) is maximized.

Since the training set consists of a discrete series of training instances, first a neural network model was created to approximate the output values of these data (the dependent variable). A single-layer multilayer perceptron network with 20 neurons was used. One of the seven entries is symbolic, therefore a one hot encoding was used in this case: the single discrete input was transformed into four distinct inputs, of which only one has the value 1 for a particular instance, and the other ones are 0.

The network was trained separately, obtaining the correlation coefficient $r = 0.97$. For optimization with the BFO algorithm, the learned parameters of the network are then directly used.

The neural network is implemented in the *NeuralNetwork* class. As mentioned above, the model has been previously trained, and the code below is only responsible for the feed-forward prediction part.

```
public class NeuralNetwork
{
  public static void Hydrogel(double parin1, double parin2, double
parin3,
    double parin4, double parin5, double parin6, double parin7,
    double parin8, double parin9, double parin10, out double
parout1)
  {
    int inputs = 10; int hidden1 = 20; int outputs = 1;

    // biases
    double[] bias_hid1 = { -0.746670307285115, ... };
    double[] bias_out = { 0.0258458101528987 };

    // weights
    double[] weights_in_hid1 = { 0.277194345245721, ... };
    double[] weights_hid1_out = { -0.511781826768753, ... };

    // scaling factors
    double[] sc_in = { 1.8, ... };
    double[] sc_out = { 0.0188127090301003, ... };

    // input scaling
    double[] yi = new double[10];
    yi[0] = parin1 * sc_in[0] + sc_in[1];
    ...
    yi[9] = parin10 * sc_in[18] + sc_in[19];

    // forward propagation

    double[] yhid1 = new double[hidden1];
    for (int i = 0; i < hidden1; i++)
    {
```

```
        double netinput = 0;
        for (int j = 0; j < inputs; j++)
           netinput += yi[j] * weights_in_hid1[i * inputs + j];
        double xx = netinput + bias_hid1[i];
        yhid1[i] = (1 - Math.Exp(-2 * xx)) / (1 + Math.Exp(-2 * xx));
     }

     double[] yout = new double[outputs];
     for (int i = 0; i < outputs; i++)
     {
        double netinput = 0;
        for (int j = 0; j < hidden1; j++)
           netinput += yhid1[j] * weights_hid1_out[i * hidden1 + j];
        double xx = netinput + bias_out[i];
        yout[i] = (1 - Math.Exp(-2 * xx)) / (1 + Math.Exp(-2 * xx));
     }

     // output scaling
     parout1 = (yout[0] - sc_out[1]) / sc_out[0];

     // for hydrogels, the output should be between 0 and 100

     if (parout1 < 0)
        parout1 = 0;
     if (parout1 > 100)
        parout1 = 100;
  }
}
```

Compared to the program that implements the BFO algorithm presented above, the following changes were made to accommodate the analyzed case study: the minimum and maximum limits for each input were computed, the neural network that calculates the objective function was introduced and the first four entries were processed in order to meet the constraints given by the one hot coding described above.

For our particular problem, the search space and objective function were modified correspondingly.

```
public BacterialOptimization()
{
  _searchSpace = new double[,] { { 0, 1 }, { 0, 1 }, { 0, 1 }, { 0, 1 }, {
0.23, 3.15 },
    { 0.06, 9.39 }, { 0.5, 12.33 }, { 29, 64 }, { 1, 7 }, { 0, 1.5 } };
}
private double ObjectiveFunction(double[] x)
{

  // the first 4 inputs are "one hot" (they correspond to 1 symbolic
input)

  double max = x[0]; int n = 0;

  for (int i = 1; i < 4; i++)
  {
     if (x[i] > max)
     {
        max = x[i];
        n = i;
     }
  }

  for (int i = 0; i < 4; i++)
     x[i] = 0;
  x[n] = 1;

  // using the neural network to compute the objective function
  double output;
  NeuralNetwork.Hydrogel(x[0], x[1], x[2], x[3], x[4], x[5], x[6], x[7],
x[8], x[9],
     out output);

  return -output; // maximization
}
```

To evaluate the performance of the BFO algorithm for different parameter values, a statistics module was added to calculate the mean and standard deviation for 100 different algorithm runs.

```
private void statisticsToolStripMenuItem_Click(object sender,
EventArgs e)
{
  int noRuns = 100;

  List<double> results = new List<double>(noRuns);

  for (int i = 0; i < noRuns; i++)
  {
     richTextBox.AppendText(string.Format("Run {0} / {1}\r\n", i + 1,
        noRuns));
     Application.DoEvents();

     BacterialOptimization bo = new BacterialOptimization();
     Cell sol = bo.Search();
     results.Add(-sol.Cost);
  }

  double mean, stdev;
  CalculateSimpleStatistics(results, out mean, out stdev);

  richTextBox.AppendText(string.Format("\r\nMean: {0:F4}    StDev:
     {1:F4}\r\n", mean, stdev));
}
```

```
private void CalculateSimpleStatistics(IEnumerable<double> values,
  out double mean, out double stdev)
{
  mean = 0; stdev = 0;
  int n = values.Count();

  if (n > 0)
  {
     double avg = values.Average();
     mean = avg;

     if (n > 1)
     {
        double sum = values.Sum(d => Math.Pow(d - avg, 2));
        stdev = Math.Sqrt(sum / (double)(n - 1));
     }
  }
```

}

Table I shows the experimental results obtained, with 100 runs for each configuration. The columns of the table are:

- S: the size of the population of cells (bacteria);
- s_s: the step size;
- s_{ed}: the number of elimination and dispersal steps;
- s_r: the number of reproduction steps;
- s_c: the number of chemotaxis steps;
- s_l: the length of the swimming step;
- p_{ed}: the probability of elimination and dispersion;
- d_a: the attraction coefficient;
- w_a: the weight of the attraction coefficient;
- h_r: the repulsion coefficient;
- w_r: the weight of the repulsion coefficient;
- μ: the average of the obtained results;
- σ: the standard deviation of the obtained results.

From this table, it can be seen that the BFO algorithm is quite robust, as it obtains good results for various combination of parameters. Its performance is clearly poor when the population size is very small ($S = 2$, $\mu = 80.4822$). When the size of the population is much higher, the results are better ($S = 200$, $\mu = 99.9964$ and σ is very low), but comparable to those obtained for lower values of the parameters. The population size directly affects the execution time.

When the other parameters change, the results are approximately equal, so their influence is not critical and the default values recommended by the authors can be used. Only difference in standard deviation can be highlighted here, which indicates a greater variability of solutions in cases where the number of chemotaxis steps, the attraction and repulsion coefficients and their weights change from the recommended values.

Table II presents the values of the decision variables (reaction conditions) that lead to the optimal solution, i.e. maximum yield.

TABLE II
THE VALUES OF THE REACTION CONDITIONS THAT LEAD TO MAXIMUM YIELD

Parameter	*Value*
Type of the included polymer	Starch
Monomer concentration	3.1500
Initiator concentration	9.2361
Crosslinking agent concentration	6.7856
Temperature	62.5106
Reaction time	5.7519
Amount of inclusion polymer	0.1493
Yield	99.9998

V. CONCLUSIONS

A simultaneous polymerization/crosslinking procedure was applied for the preparation of polyacrylamide hydrogels to obtain materials with different swelling capacities and morphologies. The synthesis was carried out in the presence of some natural (starch and gelatin) or synthetic (PVA) polymers with high biodegradability and yielded semi-interpenetrated networks, without excluding the formation of more or less important interpenetrated domains.

The dependence between reaction conditions (amount of monomer, initiator, crosslinking agent, inclusion polymer, temperature and reaction time) and yield in crosslinked polymer was modeled with artificial neural networks and then optimized with the BFO algorithm.

The paper emphasizes a general methodology of modeling and optimization. As future directions of investigation, on the one hand, other machine learning techniques can be used for process modeling and other optimization algorithms can be applied for this problem. On the other hand, the BFO algorithm can be used for other optimization problems.

ACKNOWLEDGMENT

This work was supported by the "Program 4, Fundamental and Border Research, Exploratory Research Projects" financed by UEFISCDI, project no. 51/2017.

REFERENCES

[1] J. M. Gonzalez-Saiz and C. Pizarro, "Polyacrylamide gels as support for enzyme immobilization by entrapment. Effect of polyelectrolyte carrier, pH and temperature on enzyme action and kinetics parameters", *European Polymer Journal*, vol. 37, no. 3, 2001, pp. 435–444.

[2] H. Ghandehari, P. Kopeckova and J. Kopecek, "In vitro degradation of pH-sensitive hydrogels containing aromatic azo bonds", *Biomaterials*, vol. 18, 1997, pp. 861–872.

[3] V. Compan, J. Guzmaun and E. Riade, "A potentiostatic study of oxygen transmissibility and permeability through hydrogel membranes", *Biomaterials*, vol. 23, 1998, pp. 2139–2145.

[4] H. Park, *Hydrogels in Bioapplications: Hydrogels and Biodegradable Polymers for Bioapplications*, eds. R. M. Ottenbrite, S. J. Huang, K. Park, American Chemical Society, Washington D. C., 1996, pp. 2–10.

[5] L. H. Christensen, V. B. Breiting, A. Aasted, A. Jorgensen and I. Kebuladze, "Long-term effects of polyacrylamide hydrogel on human breast tissue", *Plastic and Reconstructive Surgery*, vol. 111, no. 6, 2003, pp. 1883–1890.

[6] A. El-Hag Ali, H. A. Shawky, H. A. Abd El Rehim and E. A. Hegazy, "Synthesis and characterization of PVP/AAc copolymerhydrogel and its applications in the removalof heavy metals from aqueous solution", *European Polymer Journal*, vol. 39, no. 12, 2003, pp. 23–37.

[7] J. Tirthankar, C. R. Bidhan and M. Sukumar, "Biodegradable film", *Polymer Degradation and Stability*, vol. 37, 2001, pp. 861–864.

[8] S. Zlatkovic and L. Raskovic, "The effect of the polyacrylamide, polyvinylalcohol, and carboxymethylcellulose on the aggregation of the soil and on the growth of the plants", *Facta Universitatis*, vol. 1, no. 3, 1998, pp. 17–23.

[9] J. P. Baker, L. H. Hong, H. W. Blanch and J. M. Prausnitz, "Effect of Initial Total Monomer Concentration on the Swelling Behavior of Cationic Acrylamide-Based Hydrogels", *Macromolecules*, vol. 27, 1994, pp. 14–46.

[10] C. Mihăilescu, A. Dumitrescu, B. C. Simionescu and V. Bulacovschi, "Synthesis of polyacrylamide-based hydrogels by simultaneous polymerization/crosslinking", *Revue Roumaine de Chimie*, vol 52, no. 11, 2007, pp. 1071–1076.

[11] S. Curteanu, A. Dumitrescu, C. Mihăilescu and B. C. Simionescu, "Neural network modeling applied to polyacrylamide based hydrogels synthetized by single step process", P*olymer-Plastics Technology and Engineering*, vol. 47, 2008, pp. 1061–1071.

[12] S. Curteanu, A. Dumitrescu, C. Mihăilescu and B. C. Simionescu, "The synthesis of polyacrylamide-based multicomponent hydrogels. A neural network modeling", *Journal of Macromolecular Science, Part A Pure and Applied Chemistry*, vol. A46, no. 4, 2009, pp. 368–380.

[13] J. Brownlee, *Clever Algorithms: Nature-Inspired Programming Recipes*, chapter: "Bacterial Foraging Optimization Algorithm", http://www.cleveralgorithms.com/nature-inspired/swarm/bfoa.html, 2012 (last accessed: 17 April 2019).

[14] S. Das, A. Biswas, S. Dasgupta and A. Abraham, "Bacterial Foraging Optimization Algorithm: Theoretical Foundations, Analysis, and

Applications", *Foundations of Computational Intelligence*, vol. 3: Global Optimization, 2009, pp. 23–55.
[15] Y. Liu and K. M. Passino, "Biomimicry of Social Foraging Bacteria for Distributed Optimization: Models, Principles, and Emergent Behaviors", *Journal of Optimization Theory and Applications*, vol. 115, no. 3, 2002, pp. 603–628.
[16] S. D. Müller, J. Marchetto, S. Airaghi and P. Koumoutsakos, "Optimization Based on Bacterial Chemotaxis", *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, 2002, pp. 16–29.
[17] K. M. Passino, "Biomimicry of bacterial foraging for distributed optimization and control", *IEEE Control Systems Magazine*, vol. 22, no. 3, 2002, pp. 52–67.
[18] K. M. Passino, "Bacterial Foraging Optimization", *International Journal of Swarm Intelligence Research*, vol. 1, no. 1, 2010, pp. 1–16.