Chemical Reaction Optimization Algorithm to Find Maximum Independent Set in a Graph

Mohammad A. Asmaran[1], Ahmad A. Sharieh[2], Basel A. Mahafzah[3]
Department of Computer Science, The University of Jordan, Amman, Jordan

***Abstract*—Finding maximum independent set (MIS) in a graph is considered one of the fundamental problems in the computer science field, where it can be used to provide solutions for various real life applications. For example, it can be used to provide solutions in scheduling and prioritization problems. Unfortunately, this problem is one of the NP-problems of computer science, which limit its usage in providing solution for such problems with large sizes. This leads the scientists to find a way to provide solutions of such problems using fast algorithms to provide some near optimal solutions. One of the techniques used to provide solutions is to use metaheuristic algorithms. In this paper, a metaheuristic algorithm based on Chemical Reaction Optimization (CRO) is applied with various techniques to find MIS for application represented by a graph. The suggested CRO algorithm achieves accuracy percentages that reach 100% in some cases. This variation depends on the overall structure of the graph along with the picked parameters and colliding molecule selection criteria during the reaction operations of the CRO algorithm.**

Keywords*—*Chemical reaction optimization; graph; maximum independent set; metaheuristic algorithm; modified Wilf algorithm; optimization problems

I. Introduction

In this paper, a metaheuristic Chemical Reaction Optimization (CRO) algorithm has been utilized to find out maximum independent set (MIS) in a graph. In this approach, computational steps are formulated as a set of molecules reactions that leads toward approximated solution. CRO approach considers two types of collisions that could happen: On-Wall collision and Inter-molecular collision as illustrated in [1]. These collisions could be effective or ineffective depending on the nature and the type of the problem to be implemented or solved. The effective On-Wall collision is called decomposition, where the colliding molecule is supposed to be decomposed into several parts (mainly two parts). Effective inter-molecular collision is called synthesis, which involves merging the colliding molecules together.

In [2], Independent Sets (ISs) are described to be some of the useful information that can be concluded from graphs and used in real life applications; such as project scheduling and social network analysis, while they are important concept in building bipartite graphs [3,4] which are fundamental in many computing areas; such as coding theory and projective geometry. Independent set can be defined as a set of nodes in a graph that are not connected. Note that a graph may contain several independent sets, and finding the maximum one is the best goal to achieve. The IS with maximum size is referred to as MIS.

Finding an MIS in a graph is considered very useful approach for many real life applications and problems, such as optimization problems, job scheduling, and social network analysis. MIS can be determined using brute force approach in $O(N^2 \times 2^N)$ run time units, where N is the number of vertices in a graph. This approach takes a lot of time to find an MIS for large N as described in [5]. Many approaches and algorithms are proposed to find out an MIS of a graph, but with very long run time. So, many algorithms have been proposed to find out an approximation to actual exact MIS solution with less time complexity as in [6,7,8,2,9].

Here are some definitions related to MIS and CRO:

- An undirected graph is G(V, E), where V is a set of vertices and E is a set of edges in G. The set of vertices is a collection or group that contains the vertices (nodes) in the graph and these vertices (nodes) are connected to each other by links that are called Edges. The collection or group that contains all graph edges is called Edge Set noted by E.
- An Independent Set (IS) in a graph G(V,E) is defined in [10,11] to be a set V', where $V' \subseteq V$ and there is not exist an edge that connects v_s and v_e, where $e \in E$, $v_s \in V'$ and $v_e \in V'$ (i.e. either v_s or $v_e \in V'$);Where a Maximum Independent Set (MIS) is defined to be the IS of the largest size among all available ISs in G.
- Chemical Reaction Optimization (CRO) is defined in [1,15] as a metaheuristic approach that mimics the process of chemical reactions in the field of Computer Science. It relays on minimizing the potential energy to the minimal value without sticking in local minima. This algorithm defines an objective function that is used to calculate potential energy of the current state of reaction (execution) process. Just like genetic algorithms, this is done by iterating for a predefined number of iterations or meeting optimal objective value.
- On-wall ineffective collision is a CRO operation that involves colliding the molecule on the wall without any effective restructure of the colliding molecule.
- Decomposition (On-wall effective collision), is a CRO operation that involves colliding the molecule on the wall effectively so that colliding molecule is decomposed (divided) into multiple molecules.
- Inter-molecular ineffective collision is a CRO operation that involves colliding two molecules

together ineffectively so that no major structural change would occur.

- Synthesis (Inter-molecular effective collision) is a CRO operation that involves colliding two molecules together effectively so that a new molecule of the merged collided molecules will be generated.

The solution for a problem based on CRO is represented as a molecular structure noted as (ω), which has a minimum potential energy that is determined by a problem specific objective function noted as PE_ω, which is determined by an objective function $f(\omega)$. Each molecule has a kinetic energy that illustrates the tolerance of having worse solutions and noted as *KE*.

In this paper, different techniques are implemented over CRO algorithm to provide an approximate solution for the MIS problem. In these techniques, an implementation of CRO is provided to solve MIS problem to provide near optimal solution.

In the remaining sections, a review of related work is presented in Section 2. A description of the proposed algorithms will be explained in Section 3. This is followed by experimental results in Section 4 and discussion in Section 5. Section 6 presents the conclusion and intended future research.

In this paper, a new approach is applied to find Maximum Independent Set to explore its ability in order to find better approximation results than previous approaches that cannot be applied on huge graphs which may contain millions of nodes. Finding a maximum independent set with near optimal results would be used to provide a solution of many real-life applications; such as prioritization and scheduling applications.

II. Related Work

In their research for finding solution of the MIS problem, researchers have handled the issue using different approaches based on type of final result or the nature of graph, such as degree of nodes, as illustrated in [10,11,12] where such approaches have been used. In general, finding MIS can be done using one of the following three approaches: using brute force algorithm, approximation algorithms, and exact algorithms for special type of graphs.

The first approach is using an exact (brute force) algorithm. The direct way to solve such problem is to check all possible solutions by representing the presence of node in the solution by 1 and the absence by 0 as mentioned in [11]. So, we can represent the solution by a binary number with length *N*, where *N* is the number of nodes in the given graph. This involves checking 2^N numbers that represent all possible subsets of the original set of nodes. For each solution (binary number), all nodes must be checked to ensure disconnection of nodes (N^2). So, final run time complexity would be $O(N^2 \times 2^N)$. Nevertheless, some researchers have produced exact algorithms with better runtime. In [14], the authors proposed an algorithm, which achieved an exact solution in $O(1.2132^n)$ time for a graph of size *n* vertices, while in [15], the authors provided an algorithm with running time complexity of $O(1.2114^n)$ to find an exact solution. Such a reduction in the run time could have high influence in case of critical run time applications like process scheduling on a CPU.

The second approach is using approximation algorithms based on heuristics to provide approximate solution in polynomial-time. According to [12], "Most polynomial-space algorithms for MIS use the following simple idea to search a solution: branch on a vertex of maximum degree by either excluding it from the solution set, or including it to the solution set. In the first branch, we will delete the vertex from the graph and in the second branch we will delete the vertex together with all its neighbors from the graph". Algorithms that use such heuristics can be found in [7,16]. More evolutionary heuristic approaches can be found. For example, in [6,17,18,19], genetic algorithm was used to find an approximate solution for the MIS problem. In [20], a swarm intelligence approach based on ant-colony optimization was used to find a solution. Note that, approximation algorithms are used in real applications just like in [21], where genetic algorithm is used to generate data for testing PLSQL (Procedural Language extension to Structured Query Language) program units. This generated data is a sub-set of the actual data range that can't be covered in some extreme cases, where data to be tested is huge and can't be tested using normal brute-force concept. A more generic test data generation for software testing is proposed in [22] to generate test data using genetic algorithm for software testing purposes rather than using normal brute-force test data generation.

The third approach is using exact algorithms to find exact solution in polynomial-time, but for graphs of special classes, such as designing a polynomial run time algorithm that finds an exact solution in graphs with vertex of degree 2 at maximum. Such algorithms are case sensitive ones and can't be generalized to find exact solutions to graph of random shape and arbitrary degree. Examples of this form of algorithms can be found in [23], where an exact algorithm is provided for graphs with vertices of maximum degree of 3, or in [12,24], where, in addition to an exact solution provided for any random graphs, the authors provided a $O(1.1571^N)$, $1.1737^N \times N^{O(1)}$, $1.1893^N \times N^{O(1)}$, and $1.1970^N \times N^{O(1)}$, for graphs of maximum degree of 4, 5, 6, and 7, respectively.

As mentioned before, all exact solutions attempts consume a very large amount of time to execute. Such algorithms would decrease the feasibility of e solutions. So, a new paradigm of computing near optimal solution has been proposed, such as in [7,8,9,16]. As illustrated in [8], this is done using heuristic or metaheuristic techniques. Combining of exact and meta-heuristic algorithms can provide near optimal solution in a shorter time like in [25] where better execution time has been achieved. Moreover, there are some known strategies to do parallel implementation of metaheuristic approaches. By parallelizing these algorithms such as in [26], an enhanced version with better performance could be achieved.

In [27], CRO has been used to find optimal solution for task scheduling and resource allocation in grid computing. They propose several versions of CRO to solve task scheduling problem. These versions have been experimented

and tested against four metaheuristic approaches. The results show that CRO outperforms the other approaches in terms of accuracy and performance, especially in case of large test instances.

In [1], CRO has been used to provide a solution for quadratic assignment problem described in [28]. CRO implementation has been tested against various evolutionary approaches. Test results show that CRO implementation outperforms other implementations in many cases. Parallel implementation of CRO has been used to solve the same problem in [29], where test results show that parallel CRO implementation provides better performance along with solution quality in comparison with sequential one.

In [1], CRO algorithm has been used to solve resource-constrained project scheduling problem described in [30] as planning the project milestones according to predefined priorities. In real life, project is divided into fixed time slots. Project activities are assigned to time slots according to available resources that are limited, while activities could be dependent on each other. CRO is used to find best scheduling of tasks that minimizes project lifetime. Test results show that CRO implementation can achieve better results for known benchmarks.

In [1], CRO has been used to provide a solution for channel assignment problem in wireless mesh networks described in [31] to assign available channels to multiple wireless networks. It is used for wireless communication channel selection to be used in the communication between neighboring mesh routers without suffering any interference or communication problems. The results show that CRO has improved current solutions of the problem.

In [32], CRO has been used to solve population transition problem in peer-to-peer live streaming. In this problem, network live streaming has been improved by grouping peers into multiple colonies according to delay. Peers with less delay can act as service providers for longer delay ones. So, the system is said to be in universal streaming when all peers are served with sufficient streaming data. Test results show that evolutionary approach of CRO outperforms existing non-evolutionary approaches.

In [33], CRO has been used to find a solution for network coding optimization problem described in [34] to provide coding mechanism for network with minimum number of digits. In this problem, network coding has been used to enhance transmission rate between routers on certain interfaces. This strategy of coding specific interfaces could increase transmission rate without avoidance of extra computational overhead by coding all available interfaces. Test results show that CRO outperformed existing algorithms.

In [35], CRO has been used in Artificial Neural Network (ANN) training. ANN is composed of layers that contain multiple computational units called neurons. Neurons must be assigned weights to provide best results. Tuning is done by training the network with set of training data. Test results show that CRO trained ANN has better testing error.

In [36], CRO has been used to solve Set Covering Problem (SCP) while in [37] a strengthened version of clique covering has been investigated. SCP can be formulated as the following:

- Given a set M, $M_j \subseteq M$, j = 1,...,n are n subsets of M, and weights of the subsets, c_i, j = 1,...,n; and set cover is a collection $T \subseteq \{1,...,n\}$ such that $\bigcup_{j \in T} M_j$ = M. SCP tries to minimize the cost of covering the entire set using a subset of the original set. There are two types of set covering problem, unicost, and non-unicost. CRO outperformed the accuracy of other algorithms in case of non-unicost SCP, where optimal solution has been determined in 65 experiments. In case of unicost SCP, CRO shows outstanding performance in comparison to other approaches.

In [38], a version of CRO called Greedy CRO (CROG) has been proposed and implemented to solve 0-1 Knapsack Problem. Experimental results show that CROG outperforms other metaheuristic approaches, such as genetic algorithms, ant-colony, and quantum-inspired evolutionary algorithms.

In [39], enhanced version of CRO has been used to find optimal road network design that takes into consideration the cost along with noise and vehicles emissions. Proposed CRO was tested against Genetic Algorithm (GA) for comparison. Test results show that CRO outperformed GA in most cases.

In [40], Objective Power Flow (OPF) problem has been solved using CRO algorithm. OPF aims to minimize power generation cost by considering many constraints, such as the balance of the power, bus voltage magnitude limits, transmission line flow limits, and transformer tap settings. The results show that CRO can provide the best results among other algorithms on the IEEE-30 test case. Note that best result is the one with lowest power flow cost.

In [27], CRO implementation has been extended using parallel approach to solve the Quadratic Assignment Problem (QAP). QAP seeks to optimally assign facilities to locations in a way to minimize transportation cost of facilities, as they are required in multiple locations. Parallel CRO has been compared with sequential one in solving QAP, experimental results show that parallel CRO reduces computational time with more accurate results.

In [41,42], CRO implementation has been done to solve Max Flow problem (MFP) in a way that is close to Ford-Fulkerson algorithm. In [42], the results have been compared with GA in term of accuracy and performance. The results show that the problem is solvable by CRO and GA; however, the GA one outperforms the CRO one.

In this research, we provide adapted versions of CRO to find a solution of the MIS problem. Several scenarios are investigated when a molecule (subset of the graph) is selected randomly among available molecules, and a molecule is selected according to certain criteria. The selected criteria are the minimum connectivity. The adapted CRO algorithm with its implementation and performance are presented.

III. CRO Algorithm for MIS

In CRO, a molecule is represented by a node in a graph. Thus, an MIS has set of not connected nodes, or set of none neighboring nodes. In such representation, the CRO considers each molecule a candidate solution (i.e. Independent Set). Molecule potential energy is defined as the number of remaining graph nodes that are not contained in the molecule. So, if the number of graph nodes is 50 and the molecule contains 5 nodes, the potential energy is 50-5=45. Fig. 1 shows the flowchart of the CRO algorithm.

Initially, there are *N* molecules manipulated by the algorithm, since every node is considered as one molecule, which is the minimum solution (each node is an independent set). A molecule is selected for the purpose of collision in each iteration. Collision type is selected according to the initial inter-molecular to on-wall collisions ratio.

In case of inter-molecular collision, effectiveness of the collision depends on whether selected molecules can be merged together or not. This is done by checking the confliction between the two molecules, so that each node in the second molecule is checked with the conflicting (i.e. neighbors) nodes of the first molecule. If the node is found among the conflicting nodes of the first molecule, the collision is defined to be none effective collision and nothing would happen because the two molecules are not eligible to be merged. This is because each molecule is assumed to be an independent set, and it is not allowed to contain conflicting nodes. On the other hand, if the entire nodes of the second molecule are not exist among the conflicting nodes of the first molecule, the collision is defined to be effective, so that selected molecules are merged together and new molecule is formulated. This new molecule contains the whole nodes of the collided molecules.

Table I shows the mapping of chemical notations to their corresponding mathematical representation defined in [1,13]. The solution is represented by a molecular structure noted as (ω).

Fig. 1 was adapted from [1,13], shows a flowchart of the CRO algorithm, which indicates that the first step of the algorithm is the initialization, as described in [1]. Initialization includes pre-processing (e.g. preparing the data in appropriate data structure, and removing unnecessary data), and initial values calculations (e.g. algorithm variables and constants). This step is followed by the iteration checking condition, which examines stopping criteria condition to avoid infinite calculations or iterations. If stopping criteria condition is met, the algorithm execution is finished, and no more iteration is done. On the other hand, if the condition is not satisfied, no more iteration is done. In each iteration, a collision must be performed, which could be either on-wall or inter-molecular collision. This involves determination of which action to be taken in the next iteration. If the collision type is selected, the next step is to decide whether the selected collision type is effective or ineffective according to the selected collision molecules. In case of intermolecular collision, effective collision is called synthesis, which indicates that collided molecules should be merged. In case of on-wall collision, effective collision is called decomposition, which indicates that collided molecule should be decomposed into two molecules. Regardless of collision type or its effectiveness, all affected molecules potential energy should be calculated and checked with previously registered minimum value of the molecules.

In case of on-wall collision, effectiveness of the collision depends on how many times a molecule collision did happen without any improvement in the solution. So, if a predefined number of iterations are reached without any improvement in its minimum value, the collision is defined to be an effective on-wall collision. In this case, the molecule is divided into two molecules, where each molecule contains the same number of original molecule's nodes. For example, if the collided molecule contains nodes {1, 10, 19, 50}, this molecule will be divided into two molecules one molecule contains {1, 10}, while the other one contains {19, 50}.

Another main factor of the proposed algorithm is molecule selection, which indicates to how a molecule is selected for further processing, such as on-wall collision or inter-molecular collision. In this proposed algorithm, multiple scenarios are tested, as the following:

1) A molecule is selected randomly among available molecules.

2) A molecule is selected according to certain criteria. The selected criteria are the minimum connectivity.

TABLE. I. Mapping Chemical Reaction to Mathematical Meaning

Chemical Meaning	**Mathematical Meaning**	**Mathematical Representation**
Molecular structure	Solution	Ω (e.g. MIS)
Potential energy	Objective function value	$PE_\omega = f(\omega)$ (e.g. number of remaining nodes in a graph that are not selected as in the solution)
Kinetic energy	Measure of tolerance of having worse solutions	KE_ω(e.g. the same value determined by the original algorithm)
Number of hits	Current total number of moves	(e.g. number of iterations)
Minimum structure	Current optimal solution	(e.g. the best solution found during the execution of the algorithm)
Minimum value	Current optimal function value	(e.g. the potential energy of the minimum structure)
Minimum hit number	Number of moves when the current optimal solution is found	(e.g. number of iterations "hits" till finding the minimum structure)

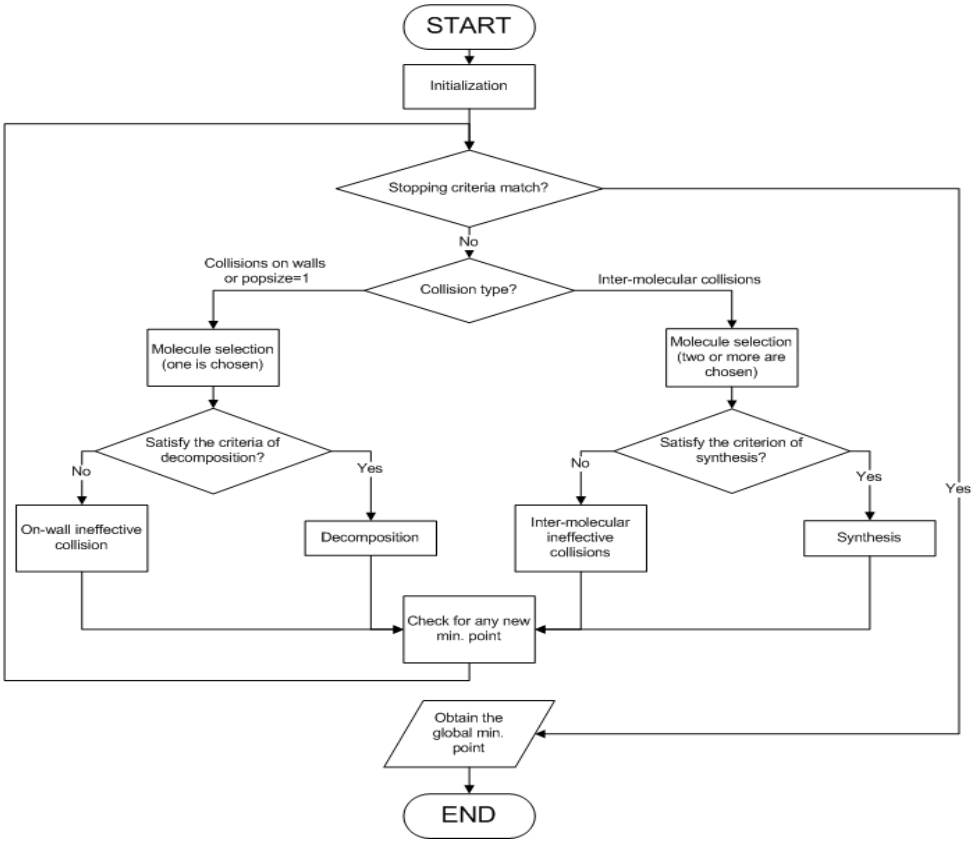

Fig. 1. A General Flowchart of the CRO Algorithm.

According to the above criteria of molecule selection, multiple combinations are tested to find out whether things go better or not, as follows:

1) Multiple random molecules: In this scenario, molecules are selected randomly for collision. In each iteration, a random molecule is selected for collision with another random selected molecule, or to collide with wall.

2) Single random molecule with random molecules: In this scenario, a random molecule is selected as a main molecule. In each iteration, this molecule is selected as the main molecule. In case of inter-molecular collision is performed, the second molecule is selected randomly. So, in this scenario, all iterations are done on the same molecule, but the variation appears in the second molecule only.

3) Single random molecule with minimum degree molecules: This scenario appears to be the same as the previous one, where a single random starting molecule is selected and used for every iteration in the reaction life cycle. But, the variation is that second molecule in case of inter-molecular collision is selected according to the criteria that is not random. Instead of that, the second molecule is selected according to its connectivity degree, where minimum connectivity degree molecule is selected to collide with fixed starting random molecule.

4) Single minimum molecule with random degree molecules: In this scenario, minimum connectivity degree molecule is selected at the beginning and used for every iteration. In case of inter-molecular collision iteration, the second molecule is selected randomly.

5) Single minimum molecule with minimum degree molecules: In this scenario, the same behavior of the previous scenario (4) is done with a difference that the second molecule in case of inter-molecular collision is selected according to its connectivity degree, so that minimum connectivity degree is selected to collide with initial minimum connectivity degree molecule.

If the collision is defined to be an effective inter-molecular collision, the components of the molecule are merged together and the conflicting nodes are computed with redundant nodes removal (no redundancy in conflicting nodes). The old molecules are removed from the pool of available molecules, while the resultant molecule is added to the pool.

In an iteration, potential energy is updated according to equation (1).

$$f(\omega) = N - Size(\omega) \quad (1)$$

Where ω denotes a molecule, Size(ω) denotes number of nodes in a molecule, and *N* denotes the number of nodes in the graph.

Kinetic energy doesn't affect the process of CRO in this proposed algorithm, since each molecule is assumed to be effective and capable of reacting with other molecules at any moment, regardless of its situation or kinetic energy.

A. CRO Algorithm

CRO algorithm relies on two major operations. These operations determine the way of finding the final solution. These operations are on-wall collision and inter-molecular collision. On-wall collision divides the molecule into two equally size molecules in case if it is effective. Otherwise (i.e. ineffective), the original molecule remains without any change. This operation is presented in Fig. 2. The other operation is to collide with another molecule (i.e. inter-molecular collision). If the collision is effective, the molecules are merged together (synthesized); otherwise, nothing happens as illustrated in Fig. 3. A function called next is called to determine whether to continue in executing next iteration or not, as it is illustrated in Fig. 4. Choosing collision molecules is done by one of two functions "chooseMinimumConflecting Molecule" or "chooseRandom Molecule", which is called once in case of on-wall collision and twice in case of inter-molecular collision to select necessary molecule(s) for collision. There are two implementations for this function, because there are two cases for the second selected molecule that are random or minimum connected molecules; as it is in Fig. 5. There are two cases for passing parameters to both functions: null value or non-null value. In case of null value, it means that this is the first selected molecule in the iteration regardless of collision type.

First selected molecule is fixed by selecting the same node all the time. So, the algorithm seeks the molecules looking for the molecule containing the initially selected node at the beginning of the algorithm execution. In case of non-null value, the algorithm has two cases: the second molecule is selected randomly or the molecule with minimum neighbors is selected.

In the code in Fig. 5, random molecule is selected. In the code in Fig. 6, the minimum neighbor molecule is selected.

At the beginning of execution, initror initmc function is called, where it is responsible of initializing molecules pool by adding created initial molecules to it and selecting the base molecule for reactions, as in Fig. 7 and Fig. 8.

```
Name: collideOnWall
Input: Molecule object that should collide on wall, and Boolean value to
specify if the collision is effective.
Output: array of molecules either with size 1 (in case of ineffective
collision) or 2 (in case of effective collision).

Function collideOnWall(Molecule molecule, boolean effective) {
   MISMolecule results[] = null;
   if(effective == true){
     Molecule results[] = new Molecule[2];
     int mid = (Number Of Nodes in molecule)/2;
     results[0] = new molecule of colliding molecule nodes indexed
     between 0 and mid-1.
     results[1] = Create Molecule of molecule Nodes indexed from mid to
     the end of the list.
   }
   else{
     results = new MISMolecule[1];
     results[0] = molecule;
   }
   return results;
}
```

Fig. 2. Collideonwall Function that Performs the Collision on Wall of the Selected (Parameterized) Molecule According to the Selected Effectiveness.

```
Name: collideWithMolecule
Input: Two molecule objects that should collide together, and Boolean
value to specify if the collision is effective.
Output: Molecule object that represents the synthesized molecule
(effective collision) or null (ineffective collision).

Function collideWithMolecule(Molecule molecule1, Molecule
molecule2, boolean effective) {
   if(effective==true){
      MISMolecule result = create molecule of nodes contained in
      molecule1 and molecule2
      return result
   }
   return null
}
```

Fig. 3. Collidewithmolecule Function that Performs the Collision between two Selected (Parameterized) Molecules According to the Selected Effectiveness.

```
Name: next
Input: number of hits (iterations), and maximum number of hits
(iterations).
Output: Boolean value that indicates whether CRO algorithm should
continue or stop its operations.

Function next() {
   if(TotalNumberOfHits<NoOfIterations){
      return true
   }
   return false
}
```

Fig. 4. Next Function Decides whether CRO Algorithm should Perform Further Steps or Stop its Work.

```
Name: chooseRandomMolecule
Input: Molecule object, and available molecules list.
Output: Chosen Molecule object (Random Selection).

Function chooseRandomMolecule(Molecule molecule) {
   Molecule pickedMolecule= null
   if(molecule == null){
      for each molecule in the available molecules{
         if(the molecule contains the default selected node){
            pickedMolecule = current molecule
            Break
         }
      }
   }else{
      int index = random number between 0 and number of available
      molecules
      pickedMolecule = select molecule located at the random index in the
      available molecules list
   }
   return pickedMolecule
}
```

Fig. 5. ChooseRandomMolecule Function that Chooses a Molecule from Available Molecules.

Note that in the code of Fig. 8, the initial selected node that would be selected during CRO life cycle is determined according to the number of neighbors, where it is the node with minimum number of nodes. While in the other implementation, the node is selected randomly among graph nodes regardless of its connectivity, as in Fig. 7.

```
Name: chooseMinimumConflectingMolecule
Input: Molecule object, and available molecules list.
Output: Chosen Molecule object (Minimum Connectivity Degree Molecule).
Function chooseMinimumConflectingMolecule(Molecule molecule){
   Molecule pickedMolecule=null
   if(molecule == null){
      for each molecule in the available molecules {
         if(the molecule contains the default selected node){
            pickedMolecule = current molecule
            Break
         }
      }
   }
   else {
      Molecule minimum = null
      int min = Number of nodes in graph
      for each molecule in the available molecules {
         if (currentMolecule != molecule){
            int temp = number of conflecting nodes in currentMolecule
            if ((minimum == null) or (temp < min)){
               minimum = choosenMolecule
               min = temp
            }
         }
      }
      pickedMolecule = minimum
   }
   return pickedMolecule
}
```

Fig. 6. Chooseminimumconflectingmolecule Function that Chooses Minimum Connectivity Degree Molecule from Available Molecules.

```
Name: initr
Input:Graph Nodes
Output: initializing molecules (conversion of graph nodes into CRO
molecules).
Function initr(){
   noOfIterations = number of graph nodes
   minimumNoOfIterations = 0
   minimumSize = number of graph nodes
   intselectedIndex= pick random number between 0 and number of nodes-1
   foreach node in the graph nodes{
      MISMolecule molecule = create molecule containing current graph node
      only
      molecule.PotentialEnergy = number of graph nodes-1
      molecule.NumberOfHits = 0
      molecule.MinimumHitNumber = 0
      molecule.MinimumStructure  = molecule;
      molecule.MinimumValue = molecule.PotentialEnergy
      add molecule to the available molecules
      if (node index=selectedIndex){
         selectedNode = node
      }
   }
   remove molecules that contain selected node neighbors nodes from the
   available molecules
   noOfIterations = noOfIterations  - number of removed molecules
}
```

Fig. 7. Initr Function that Initializes the Execution of CRO Algorithm and Chooses Starting Molecule Randomly.

B. Example

In this section, an example of the algorithm execution is provided by considering (**M**inimum initial node &**M**inimum iteration node) algorithm. Consider the graph in Fig. 9. The algorithm will initialize CRO molecules by representing each graph node by a single molecule. The potential energy equals to the number of remaining nodes not included in the molecule. So, initially, there are 5 molecules where these molecules contain nodes 1, 2, 3, 4, and 5; while potential energy for each of them is 4. This is because there is a graph node in the molecule, and the remaining graph nodes are not included in the molecules.

The algorithm will pick molecule with minimum conflicting nodes first, and do all reactions on that molecule. In this example, the algorithm can pick one of the molecules containing nodes 1, 4, and 5, as each has minimum number of conflicting nodes, which equals to 2.

```
Name: initmc
Input: Graph Nodes.
Output: initializing molecules (conversion of graph nodes into CRO
molecules) and picks minimum connected node molecule as initial starting
solution.
function initmc(){
   noOfIterations = number of graph nodes
   //the algorithm will iterate exactly the number of nodes
   minimumNoOfIterations = 0
   //initial minimum number of iterations to find solution is 0
   minimumSize = number of graph nodes
   /*minimum solution initially is same number of graph nodes (maximum
   excluded nodes in worst case)*/
   int minimumLinks = number of graph nodes + 1
   /*initial minimum number of node linkes is the number of graph nodes +1
   note that this variable is used to keep track of discovered minimum no of
   node neighbors*/
   foreach node in the graph nodes{
      MISMolecule molecule = create molecule containing current graph node
      only
      //each node in the graph would be represented as a unique molecule.
      molecule.PotentialEnergy = number of graph nodes-1
      /*initial molecule potential energy is the no of remaining graph nodes
      not included in the molecule which is the number of graph nodes-1*/
      molecule.NumberOfHits = 0
      //initial no of hits is 0 where no collisions have occurred.
      molecule.MinimumHitNumber = 0
      //minimum no of hits to find best solution is initially 0
      molecule.MinimumStructure  = molecule;
      /*minimum structure (best solution) is the initial one which is the current
      molecule structure (one node)*/
      molecule.MinimumValue = molecule.PotentialEnergy
      /* minimum value of potential energy (best solution value) is the initial
      one which is the initial potential energy of molecule*/
      add molecule to the available molecules
      //adding molecule to the molecules pool.
      if (minimumLinks> number of node Neighbors)
      {
         selectedNode = node
         minimumLinks = number of node Neighbors
      }
      /*check the number of current node neighbors so that if it is less than
      minimum observed links, then its corresponding molecule will be
      selected to be initial colliding molecule and its number of neighbors is
      saved in minimumLinks to keep track of it and compared to remaining
      nodes*/
   }
   remove molecules that contain selected node neighbors nodes from the
   available molecules
   /*selected node neighbors should be excluded from the molecules pool
   since they won't be part of the solution (IS) since their neighbor node is
   selected to be initial part of the solution*/
   noOfIterations = noOfIterations  - number of removed molecules
   //number of  iterations decreased by the number of removed molecules
}
```

Fig. 8. Initmc Function that Initializes the Execution of CRO Algorithm and Chooses Starting Molecule with Minimum Connectivity Degree.

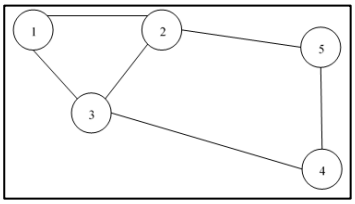

Fig. 9. Example of a Graph of 5 Nodes; Initially, each Node is Considered a Molecule and a Potential Energy for Each is 4.

Pick one of these three nodes randomly; and assume that molecule of node 5 is selected. The algorithm will iterate 5 times (number of nodes in graph). In the first iteration, the algorithm will choose another molecule (molecule that contains node 1) and do the collision with previously selected one (contains node 5). The algorithm will check the effectiveness of collision by checking whether the nodes in the two molecules are conflicting (neighbors) or not; which in this case, they are not. So, the collision is effective, and the molecules should merge (synthesized). This will produce new molecule that contains nodes 1 and 5, and the number of conflicting nodes is 3, and the potential energy is modified to be 3 instead of 4. Assume a molecule that contains node 4 is selected in the next iteration, the collision with the molecule that contains nodes 1 and 5 won't be effective, and nothing would happen because the node 4 conflicts with node 5 included in the molecule.

In case on-wall collision is decided to be performed, the molecule that contains nodes 1 and 5 would be divided into two molecules: a molecule would contain node 1, and another molecule would contain node 5. At the end, the result will be the molecule that achieves lower potential energy, which is in our simple iteration is the molecule that contains nodes 1 and 5. So, the Maximum Independent Set is {1, 5}. In case the molecule that contains node 4 has been chosen to collide with the original molecule (i.e. molecule of node 1), the collision will be effective, and the two molecules will synthesize and form new molecule that contains both nodes. While in case the same molecule that contains node 4 has been chosen to collide with the molecule that contains nodes {1, and 5}, the collision will be marked as ineffective and nothing would happen. This is because node 4 conflicts with node 5 contained in the main molecule.

C. Analytical Evaluation

Given a graph with N nodes, the algorithm will iterate exactly N iterations (Stopping criteria is the iteration of N iterations, where N is the number of nodes in graph). Within each iteration, first of all, the collision type is chosen and defined to be on-wall or inter-molecular collision. If the collision is defined to be on-wall collision, one of the followings will be done according to the effectiveness of collision:

1) Effective on-wall collision: The original molecule is divided into two molecules containing the halves of the original molecule. The original molecule is removed from the molecules pool, and the resultant molecules are added to that pool. In this case, the run time complexity of dividing the molecule is $O(N/2) \approx O(N)$.

2) Ineffective on-wall collision: The original molecule remains with same structure and nothing happens at all, since the original molecule is not affected by the collision. See *collideOnWall* in Fig. 2. In this case, a constant number of steps $O(K)$ is performed, where K is constant number that represents the number of steps needed to check effectiveness flag and going forward to the next step.

On the other hand, if the collision is defined to be inter-molecular collision, one of the followings will be done according to the effectiveness of collision:

1) Effective inter-molecular collision: This is referred to as "collideWithMolecule" function in Fig. 3. Note that effective is a Boolean parameter that indicates whether the collision is effective or not. If the collision is effective, the algorithm will iterate through first molecule, and the second molecule will create new molecule that contains all the nodes contained by the two molecules. So, in worst case, the first molecule contains half of the graph nodes, and the second one contains the other half of the graph nodes. The merge process will iterate with run time cost of $O(N/2)$ to add first molecule nodes; while in the addition of the second molecule it will check every node to prevent adding the same node twice. Every node in the second molecule will be checked across first molecule nodes with run time cost of $O(N/2)$, and this will be done for each node in the second molecule. So, the overall run time complexity is $O([N/2]+[N/2] \times [N/2]) \approx O(N^2)$.

After merging the molecules, the conflicts will be computed by adding first molecule conflicting nodes list to the second molecule conflicting nodes list with redundancy removals. In the worst case, first molecule conflicting nodes are N-2 (e.g. all graph nodes except itself and the merging node), and the second node conflicting nodes are N-2 (e.g. all graph nodes except itself and the merging node). The algorithm will iterate (N-2) to add first conflicting nodes and will iterates (N-2) to add second molecule conflicting nodes. But, while adding second molecule conflicting nodes, it will check the list of the first molecule conflicting node to prevent duplication of the nodes. In this case, all the conflicting nodes in the second molecule will be found in the first molecule. So, the second molecule conflicting nodes will be found in the list of size (N-2) added by the first molecule. This involves finding all the conflicting nodes of the second molecule in the first molecule conflicting list by iterating 1, 2, 3,…, (N/2) iterations. So, the algorithm will iterate [1+2+3+…+(N/2)] iterations to add second molecule conflicting nodes. Thus, the overall complexity is $O(N^2)$.

2) Ineffective inter-molecular collision: In this case, the resultant run time complexity of collision execution will be $O(2N^2)$. In this case, the algorithm won't do anything, refer to "collideWithMolecule" in Fig. 3, and nothing happens while the molecules are returned back to the molecules pool without any processing. So, constant number of steps is performed.

Complexity of collision effectiveness computation:

1) On-wall collision: As described in [1,13,32], the effectiveness of on-wall collision is determined by checking the number of ineffective iterations of the molecule. Ineffective iterations are the iterations that have been done on the molecule after minimum value is found without any improvement. If the number of iterations exceeds a predefined constant value, the collision will be defined to be an effective one; otherwise it is not. So, the run time complexity of determining effectiveness of the collision is constant.

2) Inter-molecular collision: The effectiveness of the collision is determined by checking the readiness of molecules to be merged together. This is done by checking the existence of any of the second molecule nodes within the first molecule conflicting nodes. If the check determines that any of the second molecule nodes exists in the conflicting nodes of the first molecule, the collision is defined to be effective; otherwise, it is not. In the worst case, first molecule contains one node and (*N*-1) conflicting nodes, while the second molecule contains all the remaining graph nodes so that its size is (*N*-1). To check the existence of second molecule nodes in the conflicting nodes of the first molecule, the whole list of the first molecule nodes should be iterated for every node in the second molecule, until finding the checked node or reaching the end of the list and the node is assumed to be not conflicting. So, the iterations are, [1,2,3,…,*N*-1] and the run time complexity is [1+2+3+…+(*N*-1)].In this case, the run time complexity of the effectiveness calculation is O((*N*-2)(*N*-3)/2)=O(N^2).

Complexity of molecule selection types:

1) Random selection: In the random selection, the algorithm will pick a random molecule from the list of available molecules to perform intended operation. So, in this case, no processing is done, and a constant number of steps (*K*) is performed.

2) Minimum connectivity degree node selection: In this case, the algorithm will iterate through the available molecules to select the molecule with minimum number of conflicting nodes. In the worst case, the number of molecules is equal to the number of graph nodes (*N*). So, the algorithm will iterate through *N* molecules to find out the one with minimum number of conflicting nodes. The complexity of finding minimum connectivity degree among *N* nodes is O(*N*). The run time complexity of finding the same initial molecule is O(*N*).

One of the main constants to be defined prior to algorithm execution is Inter-Molecular to On-Wall collisions Ratio (*R*). According to the value of *R*, the number of inter-molecular collisions equals to *R*× (Number of CRO iterations) and on-wall collisions will be (1-*R*) × (Number of CRO iterations). So, equations (2) and (3) will hold.

$$\text{Number of Inter} - \text{Molecular Collisions} = \text{O}(R \times N) \quad (2)$$

$$\text{Number of On} - \text{Wall Collisions} = \text{O}((1 - R) \times N) \quad (3)$$

The overall run time complexity of the collision is the complexity of collision effectiveness calculation, the molecule selection complexity, and the collision execution complexity according to its effectiveness; and is expressed as in equation (4).

$$\text{O(Collision)} = \text{O(Eff. Calculation)} + \text{O(Mol. Selection)} + \text{O(Col. Execution)} \quad (4)$$

In case of On-Wall collision, there are two cases:

1) Ineffective collision: By applying equation (4), the resultant is equation (5) for On-Wall collision complexity

$$\text{O(Collision)} = 1\ [\text{O}(K)] + \text{O(Mol. Selection)} + 1\ [\text{O}(K)] \quad (5)$$

Where O(Mol. Selection) depends on the molecule selection criteria. So, in case of random molecule selection, the resultant equation is represented in equation (6). While in case of minimum connectivity degree molecule selection is used, the collision run time complexity is as in equation (7).

$$\text{O(Rand. Mol. Sel. Col.)} = \text{O(constant)} \quad (6)$$

$$\text{O(Min. Con. Deg. Mol. Sel. Col.)} = \text{O}(N) \quad (7)$$

2) Effective collision: By applying equation (4), the resultant equation (8) of collision complexity is as in equation (7).

$$\text{O(Col.)} = 1\ [\text{O}(K)] + \text{O(Mol. Sel.)} + N\ [\text{O}(N)] \quad (8)$$

Where O(Mol. Selection) depends on the molecule selection criteria. So, in case of random molecule selection, the resultant equation is as in equation (9). While in case of minimum connectivity degree molecule selection is used, the collision complexity is as in equation (10).

$$\text{O(Rand. Mol. Sel. Col.)} = 1 + 1\ [\text{O}(K)] + N = \text{O}(N) \quad (9)$$

$$\text{O(Min. Con. Deg. Mol. Sel. Col.)} = 1 + N\ [\text{O}(KN)] + N = \text{O}(N) \quad (10)$$

In case of Inter-Molecular collision, there are two cases:

1) Ineffective collision: By applying equation (4), the resultant collision complexity is as in equation (11). The ineffective collision does not perform any operation on the colliding molecule(s). So, the complexity of its execution is constant (*K*). But the calculation of collision effectiveness in worst-case would check the half of graph nodes against the second half of graph nodes that could be fully connected. So, the final equation would look like the following:

$$\text{Infc} = N^2 \times [\text{O}(\textstyle\sum_{i=0}^{N-1} i) = \text{O}(\frac{(N-2)(N-3)}{2})] + \text{O(Mol. Selection)} + 1\ [\text{O}(K)] \quad (11)$$

Where O(Mol. Selection) depends on the molecule selection criteria. So, in case of random molecule selection, the resultant is as in equation (12). While in case of minimum connectivity degree molecule selection is used, the collision complexity is as in equation (13).

$$\text{O(Col.)} = N^2 + 1[\text{O}(K)] + 1 = \text{O}(N^2 + 1) = \text{O}(N^2) \quad (12)$$

$$O(\text{Col.}) = N^2 + N[O(KN)] + 1 = O(N^2 + N + 1) = O(N^2) \quad (13)$$

2) Effective collision: By applying equation (4), the resultant of collision complexity is as in equation (14).

$$O(\text{Col.}) = N^2 \times \left[O(\sum_{i=0}^{N-1} i) = O\left(\frac{(N-2)(N-3)}{2}\right)\right] + O(\text{Mol. Sel.}) + N^2[O(2N^2)] \quad (14)$$

In case of random molecule selection, the resultant is as in equation (15)

$$O(\text{Col.}) = N^2 + 1[O(K)] + N^2 = O(N^2) \quad (15)$$

While in case of minimum connectivity degree molecule selection is used, the collision complexity is as in equation (16).

$$O(\text{Col.}) = N^2 + N[O(KN)] + N^2 = O(N^2) \quad (16)$$

Overall complexity of on-wall collision is as in equation (17).

$$O(\text{on} - \text{wall collision}) = O((1 - R) \times N^2) = O(N^2) \quad (17)$$

Overall complexity of inter-molecular collision is as in equation (18).

$$O(\text{inter} - \text{molecular collision}) = R \times N \times [O(N^2 + N^2)] = O(R \times N^3) \quad (18)$$

The overall run time complexity of CRO algorithm is as in equation (19).

$$O(\text{CRO}) = O(\text{on} - \text{wall collision}) + O(\text{inter} - \text{molecular collision})$$

$$O(\text{CRO}) = O(N^2 + N^3) = O(N^3) \quad (19)$$

IV. Experimental Results

The CRO variations are implemented using Java programming language and tested for comparison purposes using random generated graphs. The generated graphs are saved on permanent storage to insure the execution of various CRO versions on the same graphs for more accurate comparison. Moreover, multiple graph sizes have been generated with different connectivity degree percentages. As in [7], most of these graphs are manipulated using Modified Wilf algorithm to find out their exact MIS solution. These exact solutions are used to find the accuracy percentage of CRO results. CRO has been executed with different inter-molecular to on-wall collision ratio values (4 values) seeking for the best ratio value in term of time and accuracy. Moreover, these executions are repeated 5 times to calculate average execution time looking for more accurate measures. The tested ratio values are (0.25, 0.50, 0.75, and 0.95). The accuracy percentage is defined, as in equation (20).

$$\text{Accuracy Percentage} = \frac{100 \times \text{Size(MIS by CRO)}}{\text{Size (MIS by Modified-Wilf)}} \quad (20)$$

The algorithms are tested on a laptop with the following specifications: CPU: Intel (R) Core(TM) i7-4510U CPU @ 2.00GHz 2.60GHz; Memory: 8.00 GB; and Operating System: 64-bit Operating System (Windows 10 Home).

After executing the CRO algorithm over a set of randomized graphs, the algorithm variations have been tested over a set of benchmark datasets to measure accuracy, where optimal solution has achieved in some cases, as shown later.

The resulting execution times are listed below for graphs of sizes range from 100 nodes to 1000 nodes with connectivity degrees (20%, 60%, 80%, and 90%) for the various versions of the proposed CRO algorithm.

Tables II to V show that when we increase the ratio of inter-molecule collision to on-wall collision ratio, the smoothness of chart increases, which indicates that the algorithm runtime much closer to theoretical analysis. On the other hand, the algorithms show unpredictable time results due to on-wall collisions that divide the molecules to two different parts. Note that those new molecules will start over collecting other molecules to formulate new solutions.

The results show that high collision ratio provides better performance for low connectivity degree. Note that this is not the case for graphs with higher connectivity degree, where the maximum collision ratio consumes the highest time. When the collision ratio is decreased, the execution time of the algorithm on highly connected graphs achieves the minimum run time, and for those with lowest connectivity degree it achieves the worst run time. This scenario is a result of checking the efficiency of collision between two molecules. As described in the algorithm code, in order to check whether the molecules can collide effectively the connections (neighbors) of the colliding molecules are checked to be sure of conflicting neighbors. So, in case of high connectivity graphs, this will be done by a higher number of iterations. As long as the ratio of collisions is low, the number of inter-molecule collisions is low, which decreases the number of checks between molecules that minimizes the time of execution.

After calculating average time for the different algorithm executions and for the different selected graphs with different sizes and different connectivity degrees, Fig. 10 shows that the best execution time is achieved when a random starting node and picking minimum connected node in each iteration, and when the inter-molecule to on-wall collision ratio is 75%. Moreover, the results show that when picking minimum connected molecules, in each iteration, it provides better in execution time performance than picking random molecule at each stage. This happens because the number of checks of conflicting nodes between colliding molecules is minimum when the two molecules are picked according to minimum connectivity. While in case of random molecules, there is no guarantee of the number of connections in the picked molecules which could be the highest, so that the number of checks of conflicting nodes is high.

TABLE. II. EXPERIMENTAL EXECUTION TIME IN MSEC. FOR 20%, 60 %, 80%, AND 90% CONNECTIVITY DEGREES' GRAPHS WITH INITIAL RANDOM GRAPH NODE SELECTION AND RANDOM MOLECULE SELECTION (RR) IN EACH ITERATION WITH (COLLISION RATIOS 0.25, 0.5, 0.75, AND 0.95)

Connectivity Degree	**20%**				**60%**				**80%**				**90%**			
Size/Collision Ratio	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**
100	593	362	256	222	50	59	46	50	19	16	19	12	6	6	3	9
200	4619	3209	1912	1925	272	256	246	271	28	53	53	49	15	15	15	16
300	13675	9078	5958	6147	900	698	831	872	137	128	121	156	37	31	34	28
400	35356	22709	15665	14917	1791	1493	1725	2094	212	234	262	281	44	47	47	46
500	62235	47346	29810	30129	4531	2631	3256	4047	218	412	409	528	84	75	84	87
600	117545	72077	48263	51952	8056	4695	5490	6882	968	625	825	912	75	112	118	134
700	187109	126150	79830	82306	10953	8171	8324	11051	1250	953	1122	1424	140	140	165	193
800	269999	192701	107425	121437	12533	11324	12495	16707	831	1297	1787	2106	237	187	240	281
900	407757	277431	166954	177255	18281	14379	18175	22828	2515	1796	2368	2940	124	234	356	406
1000	551154	373748	232953	242983	26922	20995	24464	32371	2762	1965	3231	4069	281	381	431	544

TABLE. III. EXPERIMENTAL EXECUTION TIME IN MSEC. FOR 20%, 60 %, 80%, AND 90% CONNECTIVITY DEGREES' GRAPHS WITH INITIAL MINIMUM GRAPH NODE SELECTION AND RANDOM MOLECULE SELECTION (MR) IN EACH ITERATION WITH (COLLISION RATIOS 0.25, 0.5, 0.75, AND 0.95)

Connectivity Degree	**20%**				**60%**				**80%**				**90%**			
Size/Collision Ratio	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**
100	606	409	287	251	109	81	65	69	28	25	21	18	19	12	6	10
200	4230	3035	2130	2057	553	379	296	367	90	78	95	91	43	28	34	26
300	14841	11393	7000	6832	1234	906	991	1198	212	250	253	280	59	53	59	57
400	32222	24763	14987	15814	3309	2185	2100	2555	496	303	381	451	77	87	93	106
500	67398	45029	31119	30814	4398	3722	4239	5010	543	510	793	925	168	122	178	183
600	121295	75942	49422	54483	6157	5507	7376	9413	700	862	925	1171	122	206	224	246
700	189972	137178	87550	89818	12060	10264	11134	13771	1034	1090	1503	1940	412	275	331	425
800	266150	188499	123894	130012	24717	15082	16855	21770	1847	2119	2647	3414	390	365	478	597
900	409454	267807	161301	187702	24989	19514	23766	31255	4853	2406	3150	3885	432	453	621	757
1000	545105	389631	229173	256270	35639	26005	27629	37977	2232	3428	4365	5817	409	568	797	1044

TABLE. IV. EXPERIMENTAL EXECUTION TIME IN MSEC. FOR 20%, 60 %, 80%, AND 90% CONNECTIVITY DEGREES' GRAPHS WITH INITIAL RANDOM GRAPH NODE SELECTION AND MINIMUM MOLECULE SELECTION (RM) IN EACH ITERATION WITH (COLLISION RATIOS 0.25, 0.5, 0.75, AND 0.95)

Connectivity Degree	**20%**				**60%**				**80%**				**90%**			
Size/Collision Ratio	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**
100	476	328	212	193	80	47	43	46	21	15	12	9	7	6	3	0
200	3917	2423	1525	1439	299	218	241	218	56	34	34	31	21	21	15	12
300	5952	5385	4818	4869	753	616	679	650	134	87	100	93	34	31	31	28
400	23999	16660	11689	11410	975	1113	1263	1486	362	228	196	203	56	44	34	37
500	52396	34185	24442	22005	3730	2803	2370	2844	478	359	325	387	62	53	56	68
600	34862	36307	38833	38067	4160	3831	4451	4900	375	353	515	669	68	81	106	97
700	94925	101125	54167	61317	5401	6825	7222	7741	356	750	903	968	81	97	128	149
800	233388	173169	94075	91432	2106	8976	10587	11448	1053	697	1262	1478	200	206	200	231
900	334125	149173	143130	129653	7993	16879	14036	16306	1284	1028	1750	2112	259	253	250	284
1000	309296	230572	168135	176303	8178	10025	19302	22409	1659	2281	2365	2872	331	337	322	365

TABLE. V. BEST ACCURACY RESULTS OF INITIAL RANDOM GRAPH NODE SELECTION AND RANDOM MOLECULE SELECTION (RR) IN EACH ITERATION ON 20%, 60%, 80%, AND 90% CONNECTIVITY DEGREE GRAPHS WITH (COLLISION RATIOS 0.25, 0.5, 0.75, AND 0.95)

Connectivity Degree	**20%**				**60%**				**80%**				**90%**			
Size/Collision Ratio	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**
100	32	58	63	74	57	71	86	71	67	67	67	67	50	75	75	75
200					44	56	56	67	80	80	100	80	75	75	75	75
300					42	58	50	67	67	67	67	83	60	60	80	80
400					45	55	64	55	57	57	57	71	60	60	80	80
500					50	70	60	70	43	57	57	57	60	60	60	80
600					36	43	57	50	57	57	57	71	80	80	80	80
700					45	64	64	64	57	57	71	71	60	80	60	80
800					36	50	57	57	38	63	50	63	80	80	80	80
900					46	54	62	62	50	63	75	63	50	50	67	83
1000					31	44	44	56	56	44	56	56	50	67	67	67

Fig. 10. Average Execution Time for All Tested CRO Algorithm Versions with (Collision Ratios 0.25, 0.5, 0.75, and 0.95).

In Table VI, we demonstrate the accuracy of CRO algorithm running using full random selection of initial and iteration molecules, where the accuracy is calculated using equation (20) according to Modified Wilf algorithm results. In 20% connectivity degree, Modified Wilf algorithm can obtain an MIS from a graph of up to 150 nodes in an acceptable time. While in the higher connectivity degrees (60%, 80%, and 90%) the solutions are obtained in a graph of up to 1000 nodes. The results show that the accuracy is dropping when the number of nodes is going up. When the connectivity degree increases, the accuracy becomes more stable and near to constant regardless of graph size. Moreover, when CRO algorithm is run using 95% inter-molecules to on-wall collision ratio, it provides better results. This is a result of performing more inter-molecule collisions, which provides more combinations of nodes (solutions), so that better solutions could be discovered.

Table VII demonstrates the accuracy of CRO algorithm using random selection of iteration molecules, while starting with minimum molecule (minimum connected node), where the accuracy is calculated using equation (20) according to Modified Wilf algorithm results. The results show that the accuracy is dropping when the number of nodes is growing up. When the connectivity degree increases, the accuracy becomes more stable and near to constant regardless of graph size. Moreover, when CRO algorithm is run using 95% inter-molecules to on-wall collision ratio, it provides better results.

Table VIII shows accuracy results of CRO algorithm using random selection of initial iteration molecules and picking minimum connectivity node in each iteration, where the accuracy is calculated using equation (20) according to Modified Wilf algorithm results. In the higher connectivity degrees (60%, 80%, and 90%), the solutions are obtained from graph of up to 1000 nodes. The results show that the accuracy is dropping when the number of nodes is growing up. This is a normal result of increasing the number of nodes, where the size of MIS becomes greater, so that the percentage won't be affected by low number of nodes, not like small solutions, where a single node could increase the percentage of accuracy by a significant value.

Table IX shows the accuracy results of CRO algorithm using minimum connectivity molecule and selecting minimum connectivity molecule in each iteration, where the accuracy is calculated using equation (20) according to Modified Wilf

algorithm results. In the higher connectivity degrees (60%, 80%, and 90%), the solutions are obtained up to 1000 nodes graph size. The results show that the accuracy is dropping when the number of nodes is going up. Moreover, the algorithm shows almost identical accuracy regardless of inter-molecule to on-wall collisions ratio. This indicates that the best results are obtained early at the beginning of execution, so that it doesn't differ if the collisions between the molecules are increased or not. This indication can be used to decrease the number of iterations in case of higher ratio; but the problem is how to obtain stopping condition?

Fig. 11 shows the average accuracy of each type of algorithms along with inter-molecule to on-wall collisions ratio. The figure shows that random selection of molecules in CRO iterations provides better accuracy results, especially, when the ratio of inter-molecule to on-wall collisions increases. The algorithm performance on graph with95% ratio provides better results in case of random selection. These results represent the worst results in term of accuracy among all tests. This is a result of using minimum number of neighbors as selection criteria for initial base molecule and other molecules in each iteration, so that static selection of colliding nodes is performed, and less nodes combinations are discovered.

Extra experiments have been done to test proposed implementation on benchmark datasets, such as Graph50_10, Graph100_10, Hamming6_2, Hamming6_4, Hamming8_4, and Hamming10_4 obtained from [43,44,45]. The results listed in Table X show that the CRO algorithm provides optimal solution in some cases, specially, when the selection of molecules is done in random and the inter-molecular to on-wall collisions ratio is high, such as (75% or 95%). On the other hand, the results show that minimum degree molecule selection criteria provide lower accuracy, which tends to be the result of selecting special molecules each time of collision, which could deviate from the correct path of optimal solution that may contain higher degree nodes. The results show that optimal solution (Exact solution) of MIS could be achieved by CRO. But, the main problem is that this result is not guaranteed. CRO should be executed many times (in our case 10 times) to have more solutions that may contain the optimal one. So, if the execution of CRO is finished within 1 second, and the re-execution is done 10 times, this means that the total execution time is 10 seconds, which is the actual time to be compared with. This makes Modified-Wilf better choice and more worthy to use in case of small problems (lower graph size and higher connectivity), since the difference of achieved performance is low with guaranteed results.

TABLE. VI. BEST ACCURACY RESULTS OF INITIAL MINIMUM GRAPH NODE SELECTION AND RANDOM MOLECULE SELECTION (MR) IN EACH ITERATION ON 20%, 60%, 80%, AND 90% CONNECTIVITY DEGREE GRAPHS WITH (COLLISION RATIOS 0.25, 0.5, 0.75, AND 0.95)

Connectivity Degree	**20%**				**60%**				**80%**				**90%**			
Size/Collision Ratio	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**
100	42	58	74	68	43	71	86	100	50	67	67	67	50	75	75	100
200					44	56	67	67	60	80	80	80	75	75	75	75
300					42	50	58	58	67	83	83	67	80	60	80	80
400					45	55	64	64	57	71	71	71	60	60	60	80
500					50	60	70	70	43	57	71	71	60	80	80	80
600					36	50	50	57	57	57	71	71	60	60	80	80
700					55	55	73	73	57	57	71	71	80	60	60	80
800					43	43	50	57	50	63	63	63	80	80	60	80
900					38	54	62	62	50	75	63	75	50	67	50	67
1000					38	44	50	56	44	56	56	78	50	50	67	67

TABLE. VII. EXPERIMENTAL EXECUTION TIME IN MSEC. FOR 20%, 60 %, 80%, AND 90% CONNECTIVITY DEGREES' GRAPHS WITH INITIAL MINIMUM GRAPH NODE SELECTION AND MINIMUM MOLECULE SELECTION (MM) IN EACH ITERATION WITH (COLLISION RATIOS 0.25, 0.5, 0.75, AND 0.95)

Connectivity Degree	**20%**				**60%**				**80%**				**90%**			
Size/Collision Ratio	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**
100	602	385	266	209	127	84	75	50	27	18	12	12	16	15	15	6
200	1002	1493	1562	1616	622	427	293	262	54	46	62	68	36	31	21	22
300	13926	8450	6857	5362	299	403	719	912	384	275	206	200	53	40	43	43
400	38450	23482	13688	11688	2965	2118	1897	1776	600	440	331	319	87	62	75	81
500	49903	36941	26663	23738	972	1603	2972	3635	1228	837	671	653	215	200	128	131
600	113015	70506	47772	40013	15934	10204	7210	6739	1650	1290	937	802	87	121	159	190
700	90279	80873	63352	66949	2159	4441	7888	9837	362	794	1084	1406	153	187	284	293
800	138417	97888	83193	98044	29062	19616	15420	15520	3347	2821	2600	2434	221	268	322	447
900	384084	229520	160319	139371	3503	9079	16972	22223	772	1409	2290	2816	1187	737	615	525
1000	27344	79701	140458	187492	41688	37861	27133	26490	959	2088	3381	4256	225	400	603	718

TABLE. VIII. BEST ACCURACY RESULTS OF INITIAL RANDOM GRAPH NODE SELECTION AND MINIMUM MOLECULE SELECTION (RM) IN EACH ITERATION ON 20%, 60%, 80%, AND 90% CONNECTIVITY DEGREE GRAPHS WITH (COLLISION RATIOS 0.25, 0.5, 0.75, AND 0.95)

Connectivity Degree	**20%**				**60%**				**80%**				**90%**			
Size/Collision Ratio	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**
100	37	37	42	26	57	43	57	43	50	50	67	50	75	100	100	75
200					56	44	44	44	60	80	60	40	75	100	100	100
300					33	33	33	33	67	50	33	50	60	60	60	80
400					27	36	36	36	43	43	43	43	60	60	60	60
500					30	40	30	40	43	43	29	43	40	60	60	80
600					36	21	29	21	43	29	29	43	60	60	60	60
700					27	27	36	27	43	57	57	43	40	40	80	60
800					21	21	29	21	38	25	50	38	60	60	60	40
900					23	23	31	23	38	25	50	50	50	50	33	33
1000					19	13	19	25	33	44	33	33	50	50	33	50

TABLE. IX. BEST ACCURACY RESULTS OF INITIAL MINIMUM GRAPH NODE SELECTION AND MINIMUM MOLECULE SELECTION (MM) IN EACH ITERATION ON 20%, 60%, 80%, AND 90% CONNECTIVITY DEGREE GRAPHS WITH (COLLISION RATIOS 0.25, 0.5, 0.75, AND 0.95)

Connectivity Degree	**20%**				**60%**				**80%**				**90%**			
Size/Collision Ratio	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**	**0.25**	**0.5**	**0.75**	**0.95**
100	32	32	32	21	43	43	43	43	33	33	33	33	75	75	75	75
200					33	33	33	33	40	40	40	40	75	75	100	75
300					17	17	17	17	50	50	50	50	40	40	40	40
400					27	27	27	27	43	43	43	43	40	40	40	40
500					20	20	20	20	43	43	43	43	60	60	60	60
600					29	29	29	29	57	57	57	57	40	40	40	40
700					18	18	18	18	29	29	29	29	40	40	40	40
800					21	21	21	21	50	50	50	50	40	40	40	40
900					15	15	15	15	25	25	25	25	83	83	83	83
1000					25	25	25	25	22	22	22	22	33	33	33	33

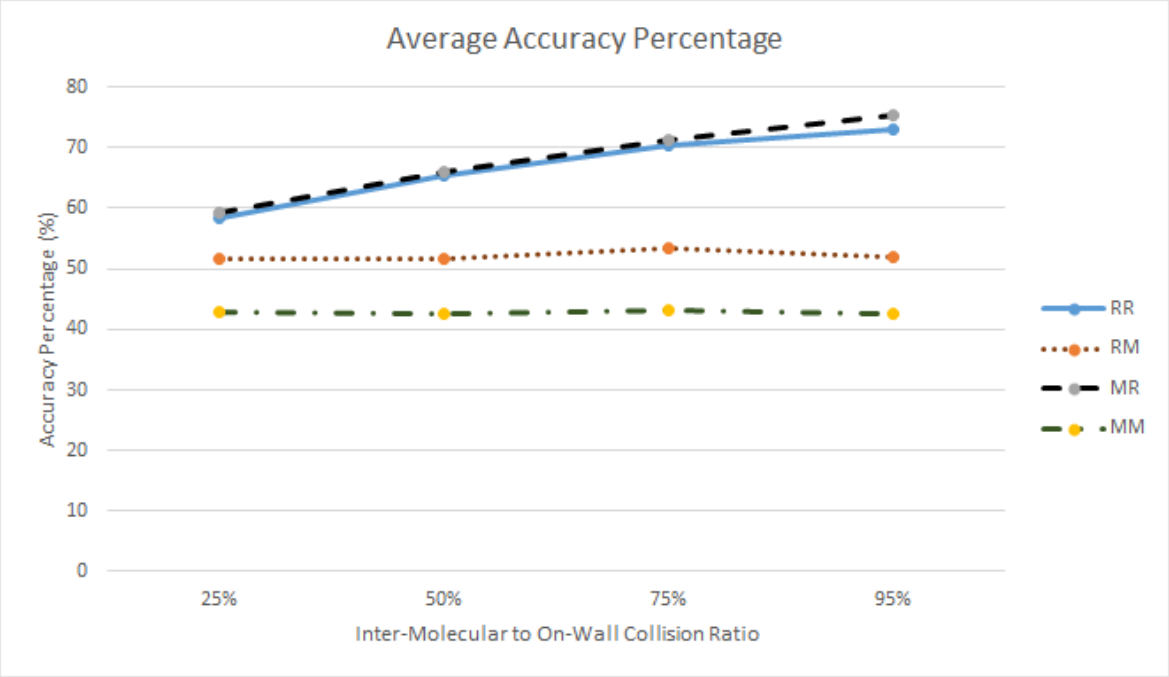

Fig. 11. Average Accuracy for the Tested CRO Algorithm Versions with (Collision Ratios 0.25, 0.5, 0.75, and 0.95).

TABLE. X. SIZES OF MIS RESULTED FROM EXECUTING CRO ALGORITHM ON A SELECTED SET OF BENCHMARK DATASETS

CRO Algorithm	Optimal MIS	CRO-RR				CRO-RM				CRO-MR				CRO-MM			
Benchmark/Collision Ratio		0.25	0.5	0.75	0.95	0.25	0.5	0.75	0.95	0.25	0.5	0.75	0.95	0.25	0.5	0.75	0.95
Graph50_10	15	7	11	14	15	14	15	15	15	8	12	14	15	14	14	14	14
Graph100_10	30	25	30	30	30	30	30	30	30	22	30	30	30	30	30	30	30
Hamming6_2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Hamming6_4	12	10	10	12	12	9	10	10	9	8	10	12	12	8	10	9	9
Hamming8_4	16	8	13	16	16	10	12	12	11	8	14	16	16	10	10	10	12
Hamming10_4	20	7	11	20	20	11	12	12	11	8	11	17	20	11	11	11	11

V. CONCLUSIONS AND FUTURE WORK

In this paper, scenarios of CRO algorithm have been implemented and applied to solve the MIS problem. The CRO algorithm is applied and evaluated on a set of randomized graphs and the accuracy of the results has been compared against exact solutions obtained by Modified Wilf algorithm.

The algorithm mainly converts graph nodes into a set of molecules. After that, it picks one of the molecules to be its main molecule for reaction. Then, it iterates while picking another interacting molecule to collide with initial one. In this paper, the selection of molecule was implemented using random one and minimum connected one.

The algorithm is tested with variety of parameters, and provides good results in some cases; such as the results of executing RR version with higher collision ratio of the algorithm, as shown in Tables VI and X. This shows that CRO technique can be used to find the solution of MIS problem, and it can be modified to provide better results. The random technique of selecting initial molecule and selecting molecules that are involved in the reaction iterations achieves better accuracy, while guided technique that depends on the degree of connectivity achieves better execution time. This can lead to look for a combination of both techniques to achieve better results in term of execution time and accuracy. Note that the average accuracy of random approach is about 75%, and achieves exact solution in some cases, while minimum approach outperforms random approach by decreasing the execution time by at least 25%.

As future work, this algorithm will be adapted to run on a parallel architecture, just like in [46,47], where a parallel heuristic local search is used to solve travelling salesman problem using four parallel architectures (e.g. “OTIS-Hypercube”, “OTIS-Mesh”, OTIS hyper hexa-cell, and OTIS mesh of trees optoelectronic architectures) and testing against other architectures, such as “The optical chained-cubic tree interconnection network” which is illustrated in [48].

REFERENCES

[1] A. Lam, V. Li, “Chemical-Reaction-Inspired Metaheuristic for Optimization”. IEEE Transactions on Evolutionary Computation, 2010, 14(3): pp: 381 – 399, https://doi.org/10.1109/TEVC.2009.2033580.

[2] S. Butenko, “Maximum Independent Set And Related Problems, With Applications”. PhD dissertation, University Of Florida, 2003.

[3] Y. Shang, "Groupies in random bipartite graphs". Applicable Analysis and Discrete Mathematics, 2010, 4(2), pp: 278–283, DOI: 10.2298/AADM100605021S.

[4] Y. Shang, "On the Hamiltonicity of random bipartite graphs". Indian Journal of Pure and Applied Mathematics, 2015, 46(2), pp: 163–173, DOI: 10.1007/s13226-015-0119-6.

[5] Y. Liu, J. Lu, H. Yang,X. Xiao, and Z. Wei, “Towards maximum independent sets on massive graphs”. in Proceedings of the VLDB Endowment, 2015, 8(13), pp:2122-2133, https://doi.org/10.14778/2831360.2831366.

[6] S. Abu Nayeem and M. Pal Madhumangal, “Genetic algorithmic approach to find the maximum weight independent set of a graph”, Journal of Applied Mathematics and Computing, 2007, 25(1), pp: 217-229. https://doi.org/10.1007/BF02832348.

[7] A. Al-Jaber and A. Sharieh, “Algorithms Based on Weight Factors for Maximum Independent Set”. DIRASAT , 1999,27(1), pp: 74-90.

[8] D. Andrade, M. Resende, and R. Werneck, “Fast local search for the maximum independent set problem”. Journal of Heuristics, (2012), 18(4), pp: 525-547. https://doi.org/10.1007/s10732-012-9196-4.

[9] T. Chan and S. Har-Peled , “Approximation Algorithms for Maximum Independent Set of Pseudo-Disks”. Discrete & Computational Geometry,2012, 48(2), pp: 373-392, https://doi.org/10.1007/s00454-012-9417-5.

[10] J. Robson, “Algorithms for maximum independent sets”. Journal of Algorithms, 1986, 7(3), pp:425-440, https://doi.org/10.1016/0196-6774(86)90032-5.

[11] Wikipedia. “Independent set (graph theory)”.2016, [online] Available at: https://en.wikipedia.org/wiki/Independent_set_(graph_theory) [Accessed 01 Nov. 2018].

[12] A. Lam, Li Victor, “Chemical Reaction Optimization: a tutorial. Memetic Computing”, 2012, 4(1), pp: 3-17, https://doi.org/10.1007/s12293-012-0075-1.

[13] M. Xiao and H. Nagamochi. “Exact Algorithms for Maximum Independent Set”. Algorithms and Computation, 2013, pp: 328-338. Berlin, Heidelberg: Springer, https://dx.doi.org/10.1016/j.ic.2017.06.001.

[14] J. Kneis, A. Langer, and P. Rossmanith, “A Fine-Grained Analysis of a Simple Independent Set Algorithm, ” in Proceedings of FSTTCS 2009, 2009, pp: 287–298, https://doi.org/10.4230/LIPIcs.FSTTCS.2009.2326.

[15] N. Bourgeois, B. Escoffier, V. Paschos, and J. Van Rooij, (2012) “Fast algorithms for Max Independent Set,” Algorithmica, 62(1).382–41, https://doi.org/10.1007/s00453-010-9460-7.

[16] A. Sharieh, W. Al-Rawagepfeh, M. Mahafzah, and A. Al-Dahamsheh, “An Algorithm for finding Maximum Independent Set in a Graph”. European Journal of Scientific Research, 2008, 23(4), pp: 586-596.

[17] T. Back and S. Khuri, “An evolutionary heuristic for the MIS problem, Evolutionary Computation” IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on Computational Intelligence, 1994, pp: 531-535, https://doi.org/10.1109/ICEC.1994.350004.

[18] X. Liu, A. Sakamoto, T. Shimamoto, “A genetic algorithm for maximum independent set problems”. in Proceedings of 1996 IEEE International Conference on Systems, Man and Cybernetics, Beijing, China, 14-17 October 1996, pp: 1916 - 1921 vol.3, https://doi.org/10.1109/ICSMC.1996.565404.

[19] S. Mehrabi, A. Mehrabi, and A. Mehrabi, "A New Hybrid Genetic Algorithm for Maximum Independent Set Problem". In Proceedings of the 4th International Conference on Software and Data Technologies, (ICSOFT 2009), Sofia, Bulgaria, July 26-29, 2009, pp: 314 – 317, https://doi.org/10.5220/0002253403140317.

[20] L. Youmei and X. Zongben, "An Ant Colony Optimization Heuristic for Solving MIS Problems, "Computational Intelligence and Multimedia Applications, ICCIMA 2003. Proceedings. Fifth International Conference, pp: 206-211, 2003, http://dx.doi.org/10.1109/ICCIMA.2003.1238126.

[21] M. Alshraideh, B. Mahafzah, H. Salman, and I. Salah, "Using genetic algorithm as test data generator for stored PL/SQL program units", Journal of Software Engineering and Applications,2013, 6(2), pp: 65-73, http://dx.doi.org/10.4236/jsea.2013.62011.

[22] M. Alshraideh, B. Mahafzah, and S. Al-Sharaeh, "A multiple-population genetic algorithm for branch coverage test data generation", Software Quality Journal, 2011, 19(3), pp: 489-513, https://doi.org/10.1007/s11219-010-9117-4.

[23] I. Razgan, "Faster Computation of MIS and Parameterized Vertex Cover for Graphs with Maximum Degree 3," Journal of Discrete Algorithms, 2009, 7(2), pp: 191-212, https://doi.org/10.1016/j.jda.2008.09.004.

[24] M. Xiao and H. Nagamochi, "An exact algorithm for maximum independent set in degree-5 graphs". Discrete Applied Mathematics 199, 2016, pp: 137–155, https://doi.org/10.1016/j.dam.2014.07.009.

[25] J. Puchinger and G. Raidl, "Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization: A Survey and Classification". Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach (pp" 41-53). Berlin, Heidelberg: Springer,2005, https://doi.org/10.1007/11499305_5.

[26] V. Cung, S. Martins, C. Ribeiro, and C. Roucairol. "Strategies for the Parallel Implementation of Metaheuristics". Essays and Surveys in Metaheuristics, US: Springer,2002, (pp. 263-308). https://doi.org/10.1007/978-1-4615-1507-4_13.

[27] H. Kim, H. Lam, and S. Kang, "Chemical Reaction Optimization for Task Scheduling in Grid Computing". IEEE Transactions on Parallel and Distributed,2011, 22(10), pp: 1624 – 1631, https://doi.org/10.1109/TPDS.2011.35.

[28] E. Loiola, N. de Abreu, P. Boaventura-Netto, P. Hahn, and T. Querido, "A survey for the quadratic assignment problem". Eur J Oper Res, 2007, 176(2), pp:657–690, https://doi.org/10.1016/j.ejor.2005.09.032.

[29] J. Xu, A. Lam, and V. Li, "Parallel Chemical Reaction Optimization for the Quadratic Assignment Problem". Proceedings of the 2010 International Conference on Genetic and Evolutionary Methods, GEM 2010, July 12-15, 2010, Las Vegas Nevada, USA.

[30] E. Demeulemeester and W. Herroelen,"Project scheduling: a research handbook". Academic Publishers, Boston, MA, USA, 2002, https://doi.org/10.1007/b101924.

[31] A. Subramanian, H. Gupta, S. Das, and J. Cao, "Minimum interference channel assignment in multiradio wireless mesh networks". IEEE Trans Mobile Comput, 2008, 7(12), pp:1459–1473, https://doi.org/10.1109/TMC.2008.70.

[32] A. Lam, J. Xu, and V. Li, "Chemical reaction optimization for population transition in peer-to-peer live streaming". Proceedings of the IEEE congress on evolutionary computation. Barcelona, Spain, 2010, https://doi.org/10.1109/CEC.2010.5585933.

[33] B. Pan, A. Lam, and V. Li, "Network coding optimization based on chemical reaction optimization". Proceedings of the IEEE global communications conference. Houston, TX, USA, 2011, https://doi.org/10.1109/GLOCOM.2011.6133697.

[34] M. Kim, M. Medard, V. Aggarwal, U. OReilly, W. Kim, and C. Ahn, "Evolutionary approaches to minimizing network coding resources". Proceedings of the 26th annual IEEE conference on computer Communications, Anchorage, AK, USA,2007, https://doi.org/10.1109/INFCOM.2007.231.

[35] P. Palmes, T. Hayasaka, and S. Usui, "Mutation-based genetic neural network". IEEE Trans Neural Network, 2005, 16(3), pp:587–600, https://doi.org/10.1109/TNN.2005.844858.

[36] J. Yu, A. Lam, and V. Li, "Chemical reaction optimization for the set covering problem". in Proceedings of 2014 IEEE Congress on Evolutionary Computation (CEC 2014), Beijing, China, 6-11 July 2014, In IEEE CEC Proceedings, 2014, pp: 512-519, https://doi.org/10.1109/CEC.2014.6900233.

[37] Y. Shang, "Poisson approximation of induced subgraph counts in an inhomogeneous random intersection graph model". Bulletin of the Korean Mathematical Society, in press.

[38] T. Truong, K. Li, and Y. Xu, "Chemical reaction optimization with greedy strategy for the 0–1 knapsack problem". Applied Soft Computing, 2013, 13(4), pp: 1774–1780, https://doi.org/10.1016/j.asoc.2012.11.048.

[39] W. Szeto, Y. Wang, and S. Wong, "The chemical reaction optimization approach to solving the environmentally sustainable network design problem". Computer-Aided Civil and Infrastructure Engineering, 2014, 29(2), pp: 140-158, https://doi.org/10.1111/mice.12033.

[40] Y . Sun, A. Lam, V. Li, J. Xu, and J. Yu, "Chemical reaction optimization for the optimal power flow problem". The 2012 IEEE Congress on Evolutionary Computation (CEC 2012), Brisbane, Australia, 10-15 June 2012. In IEEE CEC Proceedings, 2012, pp: 1-8, https://doi.org/10.1109/CEC.2012.6253003.

[41] Y. Khanafseh, M. Surakhi, A. Sharieh, and A. Sleit, "A Comparison between Chemical Reaction Optimization and Genetic Algorithms for Max Flow Problem", International Journal of Advanced Computer Science and Applications (IJACSA), 2017, 8(8), pp: 8-15, http://dx.doi.org/10.14569/IJACSA.2017.080802.

[42] R. Barham, A. Sharieh, and A. Sliet, "Chemical Reaction Optimization for Max Flow Problem", (IJACSA) International Journal of Advanced Computer Science and Applications, 2016, 7(8), pp: 189-196.

[43] K. Xu,"Vertex Cover Benchmark Instances (DIMACS & BHOSLIB)". IJEA (international journal of Experimental algorithms),2012, 3(1), pp: 1-18.

[44] Penn State Harrisburg University. Vertex Cover Benchmark Instances, 2019. [online] Available at: https://turing.cs.hbg.psu.edu/benchmarks/vertex_cover.html [Accessed 27 March 2019].

[45] DIMACS. the Center for Discrete Mathematics and Theoretical Computer Science, 2019. [online] Available at: http://dimacs.rutgers.edu [Accessed 8 March 2019].

[46] A. Al-Adwan, B. Mahafzah, and A. Sharieh, "Solving traveling salesman problem using parallel repetitive nearest neighbor algorithm on OTIS-Hypercube and OTIS-Mesh optoelectronic architectures", Journal of Supercomputing, 2018, 74(1), pp: 1-36, https://doi.org/10.1007/s11227-017-2102-y.

[47] A. Al-Adwan, A. Sharieh, and B. Mahafzah, "Parallel heuristic local search algorithm on OTIS hyper hexa-cell and OTIS mesh of trees optoelectronic architectures" Applied Intelligence, 2018, 49(10), pp: 1-28, https://doi.org/10.1007/s10489-018-1283-2.

[48] B. Mahafzah, M. Alshraideh, T. Abu-Kabeer, E. Ahmad, and N. Hamad, "The optical chained-cubic tree interconnection network: Topological structure and properties" Computers & Electrical Engineering,2012, 38(2), pp: 330-345, https://doi.org/10.1016/j.compeleceng.2011.11.023.