Most Valuable Player Algorithm for Solving Minimum Vertex Cover Problem

Hebatullah Khattab¹, Ahmad Sharieh², Basel A. Mahafzah³

Department of Computer Science
King Abdullah II School of Information and Technology
The University of Jordan, Jordan

Abstract—Minimum Vertex Cover Problem (MVCP) is a combinatorial optimization problem that is utilized to formulate multiple real-life applications. Owing to this fact, abundant research has been undertaken to discover valuable MVCP solutions. Most Valuable Player Algorithm (MVPA) is a recently developed metaheuristic algorithm that inspires its idea from team-based sports. In this paper, the MVPA_MVCP algorithm is introduced as an adaptation of the MVPA for the MVCP. The MVPA_MVCP algorithm is implemented using Java programming language and tested on a Microsoft Azure virtual machine. The performance of the MVPA_MVCP algorithm is evaluated analytically in terms of run time complexity. Its average-case run time complexity is ceased to Θ(I( |V| + |E|)), where I is the size of the initial population, |V| is the number of vertices and |E| is the number of edges of the tested graph. The MVPA_MVCP algorithm is evaluated experimentally in terms of the quality of gained solutions and the run time. The experimental results over 15 instances of DIMACS benchmark revealed that the MVPA_MVCP algorithm could, in the best case, get the best known optimal solution for seven data instances. Also, the experimental findings exposed that there is a direct relation between the number of edges of the graph under test and the run time.

Keywords—Most valuable player algorithm; minimum vertex cover problem; metaheuristic algorithms; optimization problem

I. INTRODUCTION

In general, many heuristic and metaheuristic algorithms were used to solve many optimization problems; such as Minimum Vertex Cover Problem (MVCP), Traveling Salesman Problem (TSP), 15 puzzle problem, task scheduling, software testing, and non-optimization problems [1-4]. Examples of heuristic algorithms are A* (IDA*) heuristic search algorithm and iterative deepening algorithms [5]. Examples of metaheuristic algorithms are sea lion optimization, humpback whale optimization, Genetic Algorithm (GA), Ant Colony Optimization (ACO), Artificial Bee Colony (ABC), Particle Swarm Optimization (PSO), and Chemical Reaction Optimization (CRO) [6-14].

The MVCP is a combinatorial optimization problem in computer science. It is a classic example of an NP-hard optimization problem. The MVCP is the problem of finding the smallest set of vertices such that at least one endpoint of each edge of the tested graph belongs to that set [15].

The Vertex Cover Problem (VCP) can be defined as follows: let \( G = (V, E) \) be an undirected graph with set \( V \) of vertices and set \( E \) of edges. If \( S \) is a subset of \( V \) (\( S \subseteq V \)) and \((x, y)\) is an edge in \( G \), then \( S \) is the cover of \( G \) if either \( x \in S \) or \( y \in S \) or both [15]. The MVCP is the problem of finding a subset \( S \) such that \(|S|\) is the minimum. The MVCP can be formulated as in (1).

\[
\min \sum_{x \in V} w_x \quad (1)
\]

Subject to: \(|E| = \sum_{(x,y) \in E} c_{(x,y)}\)

where \( w_x = \begin{cases} 1 & \text{if } x \text{ exists in the solution} \\ 0 & \text{if } x \text{ does not exist in the solution} \end{cases} \)

and

\( c_{(x,y)} = \begin{cases} 1 & \text{if at least } x \text{ or } y \text{ exists in the solution} \\ 0 & \text{if neither } x \text{ nor } y \text{ exists in the solution} \end{cases} \)

For the sake of clarity, Fig. 1 depicts an illustrative example for the MVCP. In this figure, the graph \( G \) consists of 6 vertices and 6 edges. For instance, the subset \{1, 2, 5, 6\} represents a cover for \( G \), but it is not the minimum. In fact, the subsets \{1, 5\} and \{1, 6\} are the minimum ones.

Many real-life problems and applications can be developed as MVCP. Therefore, many scientists have been encouraged to create higher attempts to discover efficient solutions. Networks and communications [16], engineering [17], and bioinformatics [18] are some of the real-world applications that they were represented and solved as MVCP.

To solve the MVCP, distinct kinds of algorithms were developed to introduce worthy solutions. Several exact [19, 20], heuristic [21-23], and metaheuristic [10-14, 24] algorithms were presented to achieve the purpose of solving the MVCP. The exact algorithms always find the optimal solution to the optimization problems if the problem size is comparatively small. This is because, in a feasible time, the optimal solution can be achieved. But once the problem size starts to increase, heuristic and metaheuristic algorithms are needed. Both of these types of algorithms can find a solution as close as possible to the optimal solution. Maximum Degree Greedy (MDG), Vertex Support Algorithm (VSA), and New Modified Vertex Support Algorithm (NMVSA) [21, 22] are some of the heuristic algorithms that were introduced specifically for the MVCP.

Many researchers adapted metaheuristic algorithms to handle MVCP. The GA was used to solve the MVCP in [10, 11]. The ACO algorithm also tackled the MVCP in [12, 13]. Furthermore, it was addressed by the CRO algorithm in [14].
Fig. 1. A Graph $G$ with 6 Vertices and 6 Edges, where $MVC = \{1, 5\}$.

The Most Valuable Player Algorithm (MVPA) is a recent metaheuristic algorithm that was introduced in 2017 for solving optimization problems [25]. The algorithm is inspired by sport, where the players make up teams, then they compete (in teams) together to win the championship. They also compete for the best player award on an individual basis.

In this paper, the MVPA is adapted and implemented for treating the MVCP. This algorithm is implemented using Java programming language and executed on a Microsoft Azure virtual machine. The performance of the implemented algorithm is evaluated analytically in terms of run time complexity and cost function metrics. Furthermore, it is evaluated experimentally in terms of solution quality and run time. The experiments have been conducted using different DIMACS benchmark instances for MVCP.

The structure of this paper is summarized as follows: Section II reviews some of the related work. In Section III, the MVPA is briefly lunched and explained. Section IV shows how the MVPA is tailored to the MVCP. Section V analytically evaluates the implemented MVPA. Section VI shows and discusses the experimental outcomes. Finally, Section VII concludes the conducted work and suggests some future work.

II. RELATED WORK

As several essential applications are depicted as MVCP, it has been the heart of extensive research. In [26], a new local search algorithm that is named NuMVC (New Minimum Vertex Cover) has been introduced to tackle the MVCP. The primary concept for NuMVC was to confront some weaknesses, which normally occur in the local search algorithms, linked to the exchange of vertices and weight of edges. NuMVC has come up with new approaches to address these weaknesses.

VEWLS (Vertex Edge Weighting Local Search) algorithm is presented in [27]. This algorithm integrates the vertex weighting scheme with the edge weighting scheme. In comparison with the NuMVC algorithm, VEWLS performance was evaluated. The findings showed that the VEWLS algorithm was superior to the NuMVC algorithm.

Moreover, GA has been used to solve the MVCP in [10]. The primary objective of this research was to demonstrate the effects of growing the population size. The results exposed that the number of generations required getting the optimal solution decreases as the population size increases.

The ACO algorithm has been exploited in [13] to solve the minimum weight VCP. In this research, a heuristic strategy has been put forward to rule out suspicious elements to correct the pheromone. This strategy is based on extracting information related to the best solution. This information enhanced the canonical ACO algorithm by avoiding the premature trapping of the local optima. The findings indicated a noticeably shorter time, while the results achieved were somewhat better.

A version of CRO algorithm called Hybrid Chemical Reaction Optimization Algorithm (HCROA) was designed to solve the minimum vertex cover problem for an undirected graph in [14]. In this algorithm, a greedy approach is adopted for implementing the main reactions operators. The HCROA was compared with genetic algorithm and branch and bound approach. The comparison was in terms of the number of iterations that were executed to reach the optimal solution. The results exposed that HCROA outperformed the genetic algorithm and the branch and bound approach.

There are not many research work so far as the MVPA is concerned. This is due to the fact that MVPA is invented so recently [25]. However, the same author who introduced the MVPA has investigated his algorithm to tackle the optimal design of circular antenna arrays for maximum sidelobe levels reduction [28]. The results of testing the proposed algorithm showed that it is superior to many other counterpart algorithms.

In [29], a comparison has been conducted between the MVPA and other sport-inspired metaheuristic algorithms. All compared algorithms have been tested using unimodal and multimodal problems. The MVPA has been proved to be the best algorithm regarding unimodal problems. For the multimodal problems, it has been the best together with two other algorithms.

III. MOST VALUABLE PLAYER ALGORITHM

As previously stated, Bouchekara has recently introduced the MVPA in 2017 [25]. He inspired the idea of the MVPA from the team-based sports; where all participated players are grouped in teams. Algorithm 1 illustrates the MVPA phases.

The inputs of the MVPA are the problem size (the dimension of the tackled problem), players size (the number of players), teams size (the number of teams), and MaxNFix (the maximum number of fixtures (iterations)). The Most Valuable Player (MVP) that represents the best-obtained players is the output of the MVPA.

The MVPA begins with the initial phase. In this phase, the initial population of players is randomly created. In the main phase, all other phases are executed and repeated until the stop condition is satisfied. The first step that is executed in the main phase is the distribution of the players of the population into teams. The competition phase iterates for all teams. For each selected team, two types of competitions are carried out, individual and team competitions. In the individual competition, each player of the selected team tries to improve his sporting skills to be the best player in his team and the league. Concerning the team competition, it is performed among the competed teams. Each selected team plays against another randomly picked team. As a result of this play, the players of the selected team follow a certain mechanism to update their skills.
Algorithm 1: MVPA

Inputs: problem size, players size, teams size, and MaxNFix.
Output: MVP

Initial Phase:
Creation of the initial population of players

Main Phase:
Distribution of population players in teams
Do
Competition Phase:
for all teams
Individuals Competition Phase
Team Competition Phase
Bound Checking Phase
end for

Greediness Phase
Elitism Phase
Duplicates Removing Phase
Until the stop criterion is satisfied

It must be mentioned that the player skills have lower and upper bounds, and in each competition, all players are constantly seeking to improve their skills. So, if the improvement tries updating the players' skills out of these bounds, these skills must be brought back to their bounds. This checking of the player's skills bounds is regularly performed in the bound checking phase.

At the end of the competition phase, the new population of players is shaped. This new population faces the greediness phase where the player who only got better skills is accepted, otherwise, his skills remain without accepting the conducted changes. In the elitism phase, a specific number of worst players in the new population are replaced with the same number of elite players who have the best skills. As a final phase, any duplicated player is replaced by a player from the winning teams.

As mentioned beforehand, the main phase iterates until the stop condition is satisfied. In [25], this condition is determined by the number of iterations which equals a specific number that is assigned to the MaxNFix. All information about the MVPA designing details can be found in [25].

IV. MOST VALUABLE PLAYER ALGORITHM FOR THE MINIMUM VERTEX COVER PROBLEM

The first step of adapting the MVPA for the MVCP is to align its main concepts of the MVPA with the MVCP. Table I shows the main concepts of the MVPA and their related meanings of the MVCP. The player concept in the MVPA represents a solution of the graph considered. In order to implement a solution, let a graph \( G = (V, E) \), then the vector \( a = (a_1, a_2, ..., a_{|V|}) \) where \( a_i \in \{0, 1\} \), is a binary vector that represents the solution. If the \( i^{th} \) vertex contributes to covering the graph \( G \), then \( a_i = 1 \), otherwise, \( a_i = 0 \). In consequence, the number of ones in the binary vector represents the solution size which is noted as player skills in the MVPA. To clarify the idea, consider the following example. For instance, for a graph \( G \) with 6 vertices, if \( a = (1, 1, 0, 1, 0, 1) \), then the solution size is 4 and vertices 1, 2, 4 and 6 cover all edges of the graph \( G \).

The problem size in the MVPA corresponds to the number of vertices in the graph considered for the MVCP; which represents the length of the created binary vector.

In the MVCP, the number of initial solutions that are created to form the initial population corresponds to the players size concept in the MVPA. When the initial population's solutions are distributed over teams, the solution which has the minimum size in a team represents the Franchise player. However, the solution which has the minimum size in all teams is considered the most valuable player. The best-gained solution is the most valuable player after all the MaxNFix iterations.

The adaptation of MVPA for the MVCP is described in Algorithm 2. As inputs, the algorithm requires a graph to be tested, it is denoted as \( G \). The initial population size which represents the number of initial solutions is symbolized by \( I \). Variable \( T \) stands for the number of groups that the population solutions will be distributed into. The maximum number of times that the algorithm should iterate is denoted by \( M \). The best solution obtained after the \( M \) iterations is represented by the Minimum Vertex Cover (MVC) which is the output of Algorithm 2.

In regards to MVPA phases, they are adapted for the MVCP firstly by assembling the competition, bound checking, greediness, and elitism phases in the main phase. Duplicates removing phase is ignored because of what will be explained after a while. Besides, the original order of the MVPA phases is modified to meet the needs of adaptation.

In the initial phase, as in line 1 of Algorithm 2 shows, \( I \) of initial solutions are randomly created to form the initial population. The initial solutions are created using a Random Bit-Vector (RBV) approach [30]. In RBV, the solution binary vector is made up by assigning each vertex value of 0 or 1 based on a generated random number. If this number is greater than a predefined constant, then the value of the vertex will be 1, or 0 otherwise. Calculating the sizes of these initial solutions is the next step. After then, the best solution (i.e. the solution with the minimum size) of the initial population is determined as illustrated in line 2.

| TABLE I. THE MVPA CONCEPTS FOR THE MVCP |
|----------------------------------------|
| **MVPA Concept** | **MVCP Meaning** |
| Player | Solution. |
| Player skills | Solution size. |
| Problem size | Tested graph size (the number of its vertices). |
| Players size | The number of initial solutions (initial population size). |
| Franchise player | The solution with the minimum size in a team. |
| Most valuable player | The solution with the minimum size in all teams. |
Algorithm 2: MVPA_MVCP

Inputs
Graph $G(V,E)$,
$I$ (initial population size),
$T$ (number of groups) and
$M$ (maximum number of iterations)

Output
MVC (minimum vertex cover; i.e. the best solution obtained)

// Initial Phase
1. Create the initial population ($P$) by generating $I$ random initial solutions and calculate the sizes of these solutions.
2. Find the best solution in $P$ (i.e. the solution with minimum size) and denoted as $L$.

// Main Phase
3. for $f = 1$ to $M$
4. Distribute $P$ solutions over $T$ groups

// Competition Phase
5. for $i = 1$ to $T$
6. Retrieve group $i$ ($g_i$)
7. Pick randomly group $j$ ($g_j$), given that $i \neq j$
8. Find the best solutions in $g_i$ and $g_j$. Denote them as $B_i$ and $B_j$, respectively.

// Individual Competition Phase
9. for each solution ($X$) in $g_i$
10. for each vertex $d$ of $X$
11. $X_d = X_d + rand \times (B_{i,d} - X_d) + 2 \times rand \times (L_d - X_d)$

// Bound Checking Phase - Stage 1
12. if $X_d \leq 0$ then $X_d = 0$, else $X_d = 1$

end for

// Greediness Phase - Stage 1
13. if the new $X$ cover graph $G$ and its size less than the original $X$ size then
14. Accept the new $X$
15. else
16. keep the original $X$
17. end if

end for

// Team Competition Phase
18. Calculate the probability of winning $g_i$ against $g_j$ and $g_j$ against $g_i$.
19. for each solution ($X$) in $g_i$, Do
20. for each vertex $d$ of $X$
21. if $g_i$ wins against $g_j$
22. $X_d = X_d + rand \times (X_d - B_{i,d})$
23. else
24. $X_d = X_d + rand \times (B_{i,d} - X_d)$
25. end if

// Bound Checking Phase - Stage 2
26. if $X_d \leq 0$ then $X_d = 0$, else $X_d = 1$

end for

// Greediness Phase - Stage 2
27. if the new $X$ cover graph $G$ and its size less than the original $X$ size then
28. Accept the new $X$
29. else
30. keep the original $X$
31. end if

end for

// Elitism Phase
32. Recollect the solutions from all groups in the population ($P$).
33. Sort $P$ solutions based on their sizes.
34. Replace one-third of the worst solutions with one-third of the best solutions
35. end for
36. end for

The main phase, which is represented in lines 3-40, iterates $M$ times. The first step in each time is to subsequently spread the population solutions across the $T$ groups as depicted in line 4. Shortly afterwards, the competition phase (lines 5-36) starts with retrieving the group that is due to be processed ($g_i$) as shown in line 6. In line 7, another group ($g_j$) is randomly retrieved to confront $g_i$ in the team competition phase, where $g_i$ and $g_j$ should be different. Before delving in executing any of the competition phases (individual or team), it is needed first to find the best solution in $g_i$ and $g_j$ as exposed in line 8. Concerning the individual competition phase, it extends between lines 9 and 19. In this phase, each solution of $g_i$ undergoes an improvement attempt to minimize its size using the equation in line 11 [25]. In this equation, the vertices values of each solution are updated based on the values of the vertices of the best solution in the $g_i$ ($B_i$) and the best solution in population $P$ ($L$). Regarding the team competition phase, its steps are allocated in lines 20-35. Its first step is to determine the winner group by calculating the probability of winning $g_i$ against $g_j$ and $g_j$ against $g_i$. Equation (2) is used to calculate these probabilities [25].

$$P_{g_i \text{ wins against } g_j} = 1 - \frac{(\text{size}(N(g_a)))^k}{(\text{size}(N(g_a)))^k + (\text{size}(N(g_b)))^k} \quad (2)$$

where $g_a$ and $g_b$ are any competed groups, $k$ is a constant and size$N(g_a)$ is the normalized size of $g_a$’s solutions sizes that is calculated as in (3) [25].

$$\text{size}N(g_a) = \min(\text{size}(B_1), \text{size}(B_2), \ldots, \text{size}(B_T)) \quad (3)$$

On the consequence of the winner group determination, the vertices values of $g_i$ solution are updated using either the equation in line 24 or in line 26 [25].

As stated formerly, the solution vertices values are restricted to be 0 or 1. However, when the equations in lines 11, 24, and 26 are used to update the values of the vertices, some of the obtained values differ from 0 or 1. So, in the bound checking phase, these values must be checked and brought back to 0 or 1. Specifically, when the value acquired of applying any of these equations is less than 1, then the vertex value is determined to be 0. Otherwise, it is considered to be 1. Since the values of the vertices are updated in both individual and team competition phases, the bound checking phase is executed twice, once after each updating process. This is illustrated in lines 12 and 28.

Intuitively, after each updating, the sizes of the updated solutions are re-calculated. Accepting these solutions essentially is based on the fact that their sizes must be smaller than the original one, with emphasizing that the accepted
solutions should cover the graph under test. The decision of accepting the updated solutions or rejecting them mainly is made in the greediness phase. Taking into consideration that the solutions are updating twice, as mentioned beforehand, the greediness phase is accomplished also twice, once is after the updating process in the individual competition phase as shown in lines 14-18. Once again is at the end of each iteration of the team competition phase as presented in lines 30-34.

The last phase that is included in the main phase is the elitism phase. In this phase, as recommended in [25], one-third of all solutions which have worst sizes (i.e. largest sizes) are replaced with these one-third solutions which have the best sizes (i.e. smallest sizes). With an aim to perform this replacement, first of all, the solutions from all groups must be collected back to the population \(P\). In the aftermath, these solutions are sorted in a non-descending manner based on their sizes. As clearly observed in Algorithm 2, the elitism phase is implemented in lines 37-39. Ultimately, after executing all \(M\) iterations, the best-obtained solution is announced as MVC like is revealed in line 41.

It is worth noting that the duplicates removing phase is ignored during the adaptation of MVPA to the MVC. Since it is inapplicable in case of the MVC, this inapplicability attributed to that in some cases, the number of solutions of the graph under consideration is less than the initial population size. Thereupon, the duplication is unavoidable. Consequently, this phase cannot be applied.

V. ANALYTICAL EVALUATION

This section offers a detailed discussion of an analytical evaluation for MVPA_MVC algorithm in terms of the run time complexity. Given that, \(M\) is the maximum number of iterations, \(I\) is the initial population size, \(T\) is the number of groups, and \(|V|\) and \(|E|\) are, respectively, the number of vertices and edges in the graph under test.

The run time of an algorithm, as indicated in [15], is described as the number of steps executed over a particular size of input. The run time complexity calculated in this section is the average-case of the run time.

The run time complexity of MVPA_MVC algorithm is the result of summing the run time complexity of its phases. Nonetheless, the run time complexity of creating the initial population solutions and calculating their sizes are not taken into consideration. This is due to the fact that it is assumed to be a preprocessing step. Theorem 1 remarks MVPA_MVC algorithm average-case run time complexity. Yet, the details of calculating this complexity are illustrated in the following proof based on tracing the steps and phases listed in Algorithm 2.

Theorem 1 The average-case run time complexity of MVPA_MVC algorithm is \(\Theta(I \times(|V| + |E|))\).

Proof: In the initial phase (lines 1-2), the process of generating the initial population is ignored. This is because it is assumed to be a preprocessing step. Finding the best solution in \(P\) requires \(I\) steps. Thus, the total run time complexity of the initial phase is \(I\).

The main phase (lines 3-40) iterates \(M\) times. In each time, the following steps and phases are executed:

- Distributing the population solutions into \(T\) groups needs \(I\) steps.
- The competition phase (lines 5-36) iterates \(T\) times by executing the following steps and phases:
  - Finding the best solution in any group requires \(\frac{I}{T}\) steps. For the group that is currently processed and the randomly retrieved group, they require \(2 \times \frac{I}{T}\) steps.
  - The individual competition phase (lines 9-19), in this phase, processing all solutions of the group under consideration requires \(T\) iterations. In each iteration, \(|V|\) steps are needed to update the vertices of the processed solution. Another \(|V|\) steps are needed to check the updated values of the vertices in the bound checking phase. To accomplish the greediness phase, and in order to decide on accepting the updated solutions or not, it is needed to check the updated solution capability of covering the graph under test. This checking entails dropping all edges of the solution vertices. The number of these edges may range from 1 to \(|E|\) as an upper limit. Therefore, the average number of steps that are maybe needed is \(\frac{\sum_{i=1}^{[E]} i}{|E|} = \frac{|E| \times (|E| + 1)}{2 \times |E|} = \frac{|E| + 1}{2}\) steps. Consequently, the run time complexity of the individual competition phase, including the first executions of the bound checking phase and the greediness phase, is \(\frac{I}{T} \times \left(2 \times |V| + \frac{(|E| + 1)}{2}\right)\).
- The team competition phase (lines 20-35) follows the same main steps and phases included in the individual competition phase. It processes \(\frac{I}{T}\) solutions. For each solution, all its vertices values are updated and checked with \(2 \times |V|\) steps. Additionally, its capability of covering the graph under test is checked with \(\frac{|E| + 1}{2}\) steps, on average. As a result, the run time complexity of the team competition phase is also \(\frac{I}{T} \times \left(2 \times |V| + \frac{(|E| + 1)}{2}\right)\). Based on the above analysis, the total run time complexity of the competition phase is

\[
T \times \left(2 \times \frac{I}{T} + 2 \times \left(\frac{I}{T} \times \left(2 \times |V| + \frac{(|E| + 1)}{2}\right)\right)\right)
\]

- The elitism phase (lines 37-39), the first step of applying this phase is to recollect the solutions from all the groups back to the population \(P\). Actually, this step costs \(T \times \frac{I}{T} = I\) steps. Worst one-third solutions that are replaced by the best one-third solutions costs \(\frac{I}{3}\) steps. But to be able to do this replacement, sorting the
solutions regarding their sizes is needed. Radix sort is chosen to perform this task. The time complexity of radix sort is \( \Theta(I \times D) \), where \( D \) is the number of digits of the largest number to be sorted [14]. Accordingly, the elitism phase run time complexity is \( I + (I \times D) + \frac{l}{3} \).

That is to say, the total run time complexity of the main phase is

\[
M \times \left[ I + \left( T \times 2 \times \left( \frac{l}{7} + 2 \times \left( \frac{l}{7} \times \left( 2 \times |V| + \left( \frac{|E| + 1}{2} \right) \right) \right) \right) \right] + \left( I + (I \times D) + \frac{l}{3} \right)
\]

In conclusion, the overall run time complexity of the MVPA_MVCP algorithm is

\[
I + \left( M \times \left[ I + \left( T \times 2 \times \left( \frac{l}{7} + 2 \times \left( \frac{l}{7} \times \left( 2 \times |V| + \left( \frac{|E| + 1}{2} \right) \right) \right) \right) \right] \right) + \left( I + (I \times D) + \frac{l}{3} \right)
\]

However, \( M, T \) and \( D \) can be dropped since they are constants which are much less than \( I, |V| \), and \( |E| \). As a result and by dropping all other constants, the average-case run time complexity of MVPA_MVCP algorithm is ended to be \( \Theta\left( I \times \left( |V| + |E| \right) \right) \). This completed the proof of Theorem 1.

VI. EXPERIMENTAL RESULTS AND DISCUSSION

To evaluate the performance of the MVPA_MVCP algorithm experimentally, it is first implemented using Java programming language under 64-bit windows 10 operating system. Then, the MVPA_MVCP algorithm was tested on a Microsoft Azure virtual machine which has a 2.00 GHz Intel Xeon processor with 64 GB memory. Different instances of the DIMACS benchmark for the MVCP have been used to test the MVPA_MVCP algorithm. These instances are listed in Table II [31, 32], where the benchmark instances name, number of vertices, number of edges and their best known optimal solutions sizes [32] are presented.

As an overview of Microsoft Azure, it is one of the most recent services of Microsoft [33]. It is a cloud computing service that provides facilities for building, testing, deploying, and managing applications and services over Microsoft global datacenters network. In addition to supporting many frameworks and tools, it supports various programming languages. One of the most important facilities that Microsoft Azure implemented as a computer service is the creation of virtual machines [33].

| Benchmark Instances | Number of Vertices | Number of Edges | Best Known Optimal Solution Size |
|----------------------|--------------------|-----------------|----------------------------------|
| johnson8-2-4         | 28                 | 168             | 24                               |
| graph50_6            | 50                 | 857             | 38                               |
| graph50_10           | 50                 | 612             | 35                               |
| Hamming6-4           | 64                 | 1312            | 60                               |
| graph100_10          | 100                | 4207            | 70                               |
| johnson16-2-4        | 120                | 1680            | 112                              |
| keller4              | 171                | 5100            | 160                              |
| cfat200_1            | 200                | 18366           | 188                              |
| brock200_2           | 200                | 10024           | 188                              |
| Hamming8_4           | 256                | 11776           | 240                              |
| phat300_1            | 300                | 33917           | 292                              |
| phat300_2            | 300                | 22922           | 275                              |
| sanr400_0_5          | 400                | 39816           | 387                              |
| johnson32-2-4        | 496                | 14880           | 480                              |
| phat700-1            | 700                | 99800           | 689                              |

In point of fact, the evaluation of the MVPA_MVCP algorithm is based on two performance metrics, the gained solution quality and the run time. For each benchmark instance, all tests are performed 10 times. The best-case, average-case and worst-case of these 10 tries are recorded for the solution quality metric. The run times of these three cases are also recorded. Besides, it is important to emphasize that the average-case of the solution quality metric is calculated as \( \frac{\sum_{i=1}^{10} r_i}{10} \), where \( r_i \) is the minimum solution size gained in try \( i \).

It is essential to draw the attention to the values of the main variables of the MVPA_MVCP algorithm before beginning to show and discuss the experimental outcomes. Regarding the initial population size, the number of groups, and \( k \) constant that is used in (2), they are specified as recommended in [25]. Their values are 100, 5, and 1, respectively. As regards the maximum number of iterations (\( M \)), several experiments have been performed to explore its best possible value that achieves a compromise between the gained solution quality and the run time. Three instances were used as samples for the conducted exploratory experiments. These instances varied in size; small (graph50_6, 50 vertices), medium (Hamming8_4, 256 vertices) and large (phat700-1, 700 vertices). The implemented MVPA_MVCP algorithm was executed using these three instances with \( M \) values 2, 4, 6, 8, and 10. The results showed that the most appropriate value was 6. This is because of the fact that the value of best-gained solutions of all three instances has not changed after the sixth iteration. On this foundation, \( M \) was assigned to 6 in all conducted experiments.

A. Solution Quality

First and foremost, solution quality is considered as one of the most expressive performance metrics to evaluate the metaheuristic algorithms. It specifies how much a gained solution has diverted from the optimal one. In the conducted experiments, the quality of a gained solution can be assessed since the best known optimal sizes of the solutions of the selected DIMACS data instances are recorded [32]. As an
Indeed, this paper introduces the MVPA_MVCP algorithm, which is compared with other existing algorithms. The last row of Table III shows the best known optimal solution sizes. If the value of $\rho_\alpha$ equals to 1, then the size of the gained solution is the same as the size of the best known optimal solution. But with a value above 1, the size of the gained solution is worse than the size of the best known optimal solution.

In light of that, Table III demonstrates the best, average, and worst sizes of the obtained solutions and their respective approximation ratios. In this table, the chosen benchmark instances are sorted in non-descending order according to their number of vertices. Additionally, the noted bolded values appear in Table III indicate to those solutions that their sizes equal to the best known optimal solutions sizes. For the best case, average case and worst case, the MVPA_MVCP algorithm could gain respectively, seven, three and two solutions, with sizes equal to the best known optimal solutions sizes.

With respect to the approximation ratio, the gained solutions that have sizes equal to the sizes of the best known optimal solutions, their approximation ratios are equal to 1. Intuitively, the solutions that have sizes larger than the sizes of the best known optimal solutions, their values of the approximation ratios are greater than 1. All approximation ratios that have values equal to 1 are also bolded in Table III.

As an accumulated view, the last row of Table III shows the average of the approximation ratio values of all benchmark instances for all cases (i.e. best, average, and worst). These average values indicate that, on average, the MVPA_MVCP algorithm slightly diverged by only 0.01, 0.021, and 0.033 from the best known optimal solutions in the best case, average case and worst case, respectively.

### B. Run Time

In order to discuss the run time (RT), the number of edges of a graph must be taken into consideration. Indeed, this number significantly impacts the entire RT, based on the fact that it impacts one process that is frequently repeated. Usually, this process checks the solution’s capability to cover the involved graph. Actually, this check is performed by removing the edges of those vertices which are composing the solution. Thusly, when the number of edges becomes larger, more RT is needed to end the checking test. Consequently, the total RT will increase. Taking this fact into account, the DIMAC benchmark instances in Table IV are re-sorted in an ascending manner depending on their number of edges to clarify the effect of this number on the RT.

In Table IV, the RT (in seconds) of executing the MVPA_MVCP algorithm over the selected benchmark instances are recorded. The relationship between the number of edges and the RT, which outlined beforehand, can be clearly observed in all cases (best, average and worst) of gained solutions sizes. In fact, the general observation in Table IV is that, as the number of edges increases, the RT increases too.

Furthermore, Fig. 2 is created to graphically clarify the behaviour of the RT when the number of edges increases. Particularly, Fig. 2 demonstrates how long the average-case solutions can be accomplished. Moreover, since there is a large difference between the smallest and largest values of the RT in Table IV, the vertical axis of Fig. 2 is labelled by the logarithmic values of base 10 of the RT. This is to present these times more clearly. As laid out in Fig. 2, it also depicts the direct relationship between the RT and the number of edges.

| Benchmark Instances | Number of Vertices | Best Known Optimal Solution Size $\beta$ | Best $\alpha$ | Best $\beta$ | Average $\alpha$ | Average $\beta$ | Worst $\alpha$ | Worst $\beta$ |
|---------------------|-------------------|----------------------------------------|--------------|-------------|----------------|--------------|-------------|-------------|
| johnson8-2-4        | 28                | 24                                     | 24           | 1           | 24             | 1            | 24          | 1           |
| graph50_6           | 50                | 38                                     | 38           | 1           | 39             | 1.026        | 40          | 1.053       |
| graph50_10          | 50                | 35                                     | 35           | 1           | 38             | 1.086        | 40          | 1.143       |
| Hamming6-4          | 64                | 60                                     | 60           | 1           | 61             | 1.017        | 62          | 1.033       |
| graph100_10         | 100               | 70                                     | 72           | 1.029       | 73             | 1.043        | 76          | 1.086       |
| johnson16-2-4       | 120               | 112                                    | 112          | 1           | 112            | 1            | 113         | 1.009       |
| keller4             | 171               | 160                                    | 162          | 1.013       | 164            | 1.025        | 165         | 1.031       |
| cfat200_1           | 200               | 188                                    | 188          | 1           | 188            | 1            | 188         | 1           |
| brock200_2          | 200               | 188                                    | 191          | 1.016       | 192            | 1.021        | 193         | 1.027       |
| Hamming8_4          | 256               | 240                                    | 248          | 1.033       | 248            | 1.033        | 249         | 1.038       |
| phat300_1           | 300               | 292                                    | 292          | 1           | 293            | 1.003        | 294         | 1.007       |
| phat300_2           | 300               | 275                                    | 285          | 1.036       | 285            | 1.036        | 287         | 1.044       |
| sanr400_0.5         | 400               | 387                                    | 393          | 1.016       | 393            | 1.016        | 395         | 1.021       |
| johnson32-2-4       | 496               | 480                                    | 482          | 1.004       | 482            | 1.004        | 483         | 1.006       |
| phat700-1           | 700               | 689                                    | 694          | 1.007       | 694            | 1.007        | 696         | 1.01        |
| **Average of $\rho_{0.5}$** |               |                                         | **1.01**     |             | **1.021**      |              | **1.033**   |             |

TABLE III. THE BEST, AVERAGE AND WORST GAINED SOLUTION SIZES ($\alpha$) AND THEIR APPROXIMATION RATIOS ($\rho_\alpha$)
TABLE IV. THE RUNTIME (RT) (IN SECONDS) OF THE TESTED BENCHMARK INSTANCES

| Benchmark Instances | Number of Edges | RT_{Best\_n} | RT_{Average\_n} | RT_{Worst\_n} |
|---------------------|-----------------|---------------|-----------------|----------------|
| johnson8-2-4        | 168             | 0.021         | 0.021           | 0.025          |
| graph50_10          | 612             | 0.109         | 0.109           | 0.094          |
| graph50_6           | 857             | 0.2           | 0.28            | 0.213          |
| Hamming6-4          | 1213            | 0.295         | 0.262           | 0.239          |
| johnson16-2-4       | 1680            | 0.587         | 0.532           | 0.445          |
| graph100_10         | 4207            | 1.469         | 1.689           | 1.407          |
| keller4             | 5100            | 1.9           | 2.218           | 2.201          |
| brock200_2          | 10024           | 4.995         | 5.271           | 6.457          |
| Hamming8_4          | 11776           | 8.772         | 9.176           | 8.448          |
| johnson32-2-4       | 14880           | 10.75         | 19.622          | 12.687         |
| cfar200_1           | 18366           | 15.806        | 16.106          | 16.13          |
| phat300_2           | 22922           | 20.02         | 24.803          | 22.758         |
| phat300_1           | 33917           | 28.682        | 27.965          | 27.077         |
| sanr400_0.5         | 39816           | 38.961        | 60.254          | 38.665         |
| phat700_1           | 183651          | 133.538       | 115.903         | 61.814         |

Fig. 2. The Run Time of Gaining Average-Case Solutions.

VII. CONCLUSIONS AND FUTURE WORK

On one side, the MVCP is one of the NP-hard problems that many scientists have been dealing with. This is because it has demonstrated its flexibility in solving problems in several applications in real-life. On the other side, the MVP_A has recently developed as one of the metaheuristic algorithms that has been influenced by its concept in team sports. In this paper, the MVP_A implementation is presented as an adaptation of the MVP for the MVCP. The MVP_A algorithm is analytically evaluated, and several tests are conducted with a target of experimental evaluation. Regarding the analytical evaluation, the MVP_A is evaluated in terms of the run time complexity. It has been shown that its average-case run time complexity ended to be $O(I|V| + |E|)$, where $I$ is the size of the initial population, $|V|$ is the number of vertices and $|E|$ is the number of edges of the graph under test.

For the conducted experiments, the MVP_A is developed using Java programming language and it is executed on a Microsoft Azure virtual machine that has a 2.0 GHz Intel Xeon processor with 64 GB memory. As test data set, 15 DIMACS benchmark instances for minimum vertex cover problem are used.

The experimental results are evaluated in terms of the run time; in addition to the quality of the gained solutions. These results clarified that there is a direct relation between the number of edges of the processed graph and the run time. Where when the number of edges increases, the run time increases too. Besides, they showed that, in the best case, the MVPA_MVCP algorithm could gain seven solutions that have sizes exactly as the best known optimal solutions sizes.

As future work, the MVP_A algorithm can be compared with other metaheuristic algorithms such as GA and ACO. Microsoft Azure service of creating multi-core virtual machines can be also invested to parallelize the MVP_A algorithm. Additionally, it can be parallelized over some types of interconnection networks like Chained-Cubic Tree interconnection network (CCT) [34], Optical Chained-Cubic Tree interconnection network (OCCT) [35], and Optical Transpose Interconnection System (OTIS) networks; such as OTIS-hypercube, OTIS-mesh, OTIS hyper hexa-cell, and OTIS mesh of trees [36, 37]. These interconnection networks expose their usefulness for solving various problems in a parallel mode [36-38]. Solving the MVP_A algorithm on parallel computing environment could greatly reduce the run time, and it should not affect the quality of the obtained solutions.

REFERENCES

[1] Y. Raju, and N. Devarakonda, “Cluster based Hybrid Approach to Task Scheduling in Cloud Environment,” International Journal of Advanced Computer Science and Applications(IJACSA), vol. 10, no. 4, pp. 425–429, 2019. http://dx.doi.org/10.14569/IJACSA.2019.0100452.
[2] A. Alazzawi, H. Rais, and Sh. Basri, “ABCVS: An Artificial Bee Colony for Generating Variable T-Way Test Sets,” International Journal of Advanced Computer Science and Applications(IJACSA), vol. 10, no. 4, pp. 259-274, 2019. http://dx.doi.org/10.14569/IJACSA.2019.0100431.
[3] N. Salami, B. Omar, and D. Esztergari-Kiss, “Simulation Results for a Daily Activity Chain Optimization Method based on Ant Colony Algorithm with Time Windows,” International Journal of Advanced Computer Science and Applications(IJACSA), vol. 10, no. 1, pp. 425-430, 2019. http://dx.doi.org/10.14569/IJACSA.2019.0100156.
[4] A. Wicaksono, and A. Supianto, “Hyper Parameter Optimization using Genetic Algorithm on Machine Learning Methods for Online News Popularity Prediction,” International Journal of Advanced Computer Science and Applications(IJACSA), vol. 9, no. 12, pp. 263-267, 2018. http://dx.doi.org/10.14569/IJACSA.2018.091238.
[5] D. Poole, and A. Mackworth, Artificial Intelligence: Foundations of Computational Agents, 2nd ed., Cambridge University Press, 2017.
[6] R. Masadeh, B. Mahafzah, and A. Sharieh, “Sea lion optimization algorithm,” International Journal of Advanced Computer Science and Applications, vol. 10, no. 5, pp. 388–395, 2019.
[7] R. Masadeh, A. Sharieh, and B. Mahafzah, “Humpback whale optimization algorithm based on vocal behavior for task scheduling in cloud computing,” International Journal of Advanced Science and Technology, vol. 13, no. 3, pp. 121–140, 2019.
[8] M. Alshraideh, E. Jawabreh, B. Mahafzah, and H. AL Harahsheh, “Applying genetic algorithms to test JUH DBs exceptions,” Journal of Software Engineering and Applications, vol. 6, no. 2, pp. 65–73, 2013.
