Int. J. Appl. Math. Comput. Sci., 2020, Vol. 30, No. 1, 173–184
DOI: 10.34768/amcs-2020-0014

A GENETIC ALGORITHM FOR THE MAXIMUM 2–PACKING SET PROBLEM

JOEL ANTONIO TREJO-SÁNCHEZ [a], DANIEL FAJARDO-DELGADO [b],
J. OCTAVIO GUTIERREZ-GARCIA [c,*]

[a]Center for Research in Mathematics
National Council of Science and Technology
Carretera Sierra Papacal, Chuburna Puerto km 5, Mérida, 97302, Mexico
e-mail: joel.trejo@cimat.mx

[b]Department of Systems and Computation
National Technological Institute of Mexico—Ciudad Guzman
Av. Tecnológico 100, Ciudad Guzmán, Jalisco, 49000, Mexico
e-mail: dfajardo@itcg.edu.mx

[c]Department of Computer Science
Mexico Autonomous Institute of Technology (ITAM)
Río Hondo 1, Ciudad de México, 01080, Mexico
e-mail: octavio.gutierrez@itam.mx

Given an undirected connected graph $G = (V, E)$, a subset of vertices S is a maximum 2-packing set if the number of edges in the shortest path between any pair of vertices in S is at least 3 and S has the maximum cardinality. In this paper, we present a genetic algorithm for the maximum 2-packing set problem on arbitrary graphs, which is an NP-hard problem. To the best of our knowledge, this work is a pioneering effort to tackle this problem for arbitrary graphs. For comparison, we extended and outperformed a well-known genetic algorithm originally designed for the maximum independent set problem. We also compared our genetic algorithm with a polynomial-time one for the maximum 2-packing set problem on cactus graphs. Empirical results show that our genetic algorithm is capable of finding 2-packing sets with a cardinality relatively close (or equal) to that of the maximum 2-packing sets. Moreover, the cardinality of the 2-packing sets found by our genetic algorithm increases linearly with the number of vertices and with a larger population and a larger number of generations. Furthermore, we provide a theoretical proof demonstrating that our genetic algorithm increases the fitness for each candidate solution when certain conditions are met.

Keywords: maximum 2-packing set, genetic algorithms, graph algorithms.

1. Introduction

A k-packing set of graph vertices (Garey and Johnson, 2002) is a collection of vertices, such that the shortest path between any pair of vertices has at least $k + 1$ edges. Notice that the 1-packing set is commonly referred to as the independent set. In this paper, we focus on the problem of finding a 2-packing set, i.e., finding a subset S of vertices such that two given vertices in S have no common neighbors.

Graphs have been used widely for modeling multiple real-world problems. For instance, in the work of Baran (2018), the problem of finding closest paths in graphs is used for pattern recognition in the context of X-ray imaging. In this regard, in this paper, the problem of finding 2-packing sets in graphs has as a variety of applications when we need mutual exclusion between nodes with overlapping neighbourhoods (Gairing *et al.*, 2004a); for instance, in network modeling (Agrawal *et al.*, 1995), facility allocation (Feitelson, 1996), or frequency assignment (Hale, 1980), among others. In addition, recently, this problem has attracted the attention of the distributed computing community, mainly for the design of self-stabilizing algorithms (see, e.g., Gairing *et al.*,

[*]Corresponding author

2004a; 2004b; Manne and Mjelde, 2006; Turau, 2012; Shi, 2012; Trejo-Sánchez and Fenández-Zepeda, 2012, Trejo-Sánchez *et al.*, 2017).

The problem of computing a maximum 2-packing set for arbitrary graphs is NP-hard (Hochbaum and Shmoys, 1985). Although it is possible to obtain optimal solutions for trivial instances of this problem, as the size of the problem grows, the execution time increases exponentially. Genetic algorithms (GAs) (Holland, 1975) have been successfully applied to this kind of problems, where efficient exact algorithms are unknown, as, for example, the problem of setting optimal times to traffic lights in junctions (Adacher and Gemma, 2017). GAs are search methods based on the principles of biological evolution to efficiently solve multi-modal and multi-dimensional optimization problems (Eiben and Smith, 2015). In this context, a set of individuals, representing candidate solutions to a given problem, evolves towards a population of better individuals through operations based on natural selection (survival and reproduction of the fittest) and genetics (crossover and mutation). When the GA finishes its execution, the resulting solution is represented by the individual with the highest fitness value. Solutions provided by a simple GA can be improved through the addition of other methods or data structures incorporated within them.

In this paper, we present a genetic algorithm called *gen2pack* for the maximum 2-packing set problem. We enhance the algorithm with a local improvement procedure specifically designed for the maximum 2-packing set problem. By doing so, *gen2pack* is capable of finding (sub)optimal 2-packing sets on arbitrary graphs (as supported by our empirical evidence). We also provide a theoretical proof demonstrating that *gen2pack* increases the fitness for each candidate solution when certain conditions are met. We evaluate the effectiveness of *gene2pack* by performing comparisons with (i) the optimal solution, (ii) an extended version of the genetic algorithm proposed by Back and Khuri (1994), originally designed for the maximum independent set problem, and (iii) a polynomial-time algorithm to compute maximum 2-packing sets on cactus graphs proposed by Flores-Lamas *et al.* (2018).

This paper is organized as follows. Section 2 provides some basic definitions and concepts. Section 3 presents the previous work on the maximum 2-packing set problem and similar issues. Section 4 presents our algorithm and some theoretical results. Section 5 discusses our empirical results. Finally, in Section 6, we present some concluding remarks and future work.

2. Preliminaries

Let $G = (V, E)$ be an undirected graph. The *distance* between two vertices in G is the number of edges in a shortest path connecting them. The *neighborhood* of a vertex x, denoted as $N(x)$, is the set of vertices at distance one from x, i.e., $N(x) = \{y | \exists \{x, y\} \in E\}$. The *two-distance neighborhood* of a vertex x, denoted as $N^2(x)$, consists of the set of vertices that are at distance two from x, i.e., $N^2(x) = \{z | \exists y \in V$ such that $\{x, y\} \wedge \{y, z\} \in E\}$.

A subset of vertices $I \subseteq V$ is an *independent set* if there is no edge $\{x, y\} \in E$ such that both x and y belong to I. The problem of finding an independent set of maximum cardinality is an NP-hard one (Karp, 1972). A subset $S \subseteq V$ is a *2-packing set* of G if, given two vertices $x, y \in S$, there exist at least three edges between them, i.e., the distance $dist(x, y) \geq 3$. A 2-packing set S is *maximal* if there is no 2-packing set S' such that $S \subset S'$. The *maximum* 2-packing set S' is the maximal 2-packing set of the largest cardinality.

3. Related work

Most of the results for the 2-packing set are in the context of distributed algorithms (Gairing *et al.*, 2004a; 2004b; Manne and Mjelde, 2006; Shi, 2012; Trejo-Sánchez and Fernández-Zepeda, 2012; 2014; Trejo-Sánchez *et al.*, 2017). However, these algorithms compute a maximal 2-packing set on the input graph, and not necessarily a maximum 2-packing set. In contrast, our present work is focused on finding maximum 2-packing sets.

In addition, some of the research efforts tackling the 2-packing set problem are only focused on particular types of graphs (for which polynomial-time algorithms can be implemented), such as those of Trejo-Sánchez and Fernández-Zepeda (2012) or Trejo-Sánchez *et al.* (2017) for cactus graphs, Trejo-Sánchez and Fernández-Zepeda (2014) for outer planar graphs, and Soto *et al.* (2018) for 2-token graphs. Following this vein, Mjelde (2004) designed an exact algorithm to compute a maximum 2-packing set on rooted trees by using dynamic programming. More recently, Flores-Lamas *et al.* (2018) devised an exact algorithm to compute a maximum 2-packing set on a cactus graph, firstly computing a maximum 2-packing set in unicyclic graphs, and then extending such result for general cacti. Soto *et al.* (2018) compute the size of the maximum 2-packing set in 2-token graphs, and prove that this value is useful for error correcting codes.

It should be noted that, to the best of our knowledge, there is no research effort dealing with the maximum 2-packing set problem for arbitrary graphs. Nevertheless, several metaheuristic approaches have been designed for the maximum independent set problem for arbitrary graphs. In this regard, Back and Khuri (1994) defined an evolutionary algorithm for the maximum independent set problem. They represent each individual of the population as a binary string of a length equal to the number of vertices in the graph. If a vertex belongs to an independent set, it is represented as a 1-bit, otherwise it is the 0-bit. Back and Khuri's algorithm performs crossover and mutation operations according to the traditional canonical genetic algorithms. Lamm *et al.* (2017) designed evolutionary heuristics that compute independent sets on very large sparse graphs. The algorithm of Lamm *et al.* (2017) combines complex reduction rules to reduce the size of the input graph to a smaller one, in such a way that the solution in the reduced graph is valid for the original. However, such reduction rules are not necessarily valid for the maximum 2-packing set problem because the vertices of a 2-packing set of a reduced graph may be in conflict with other vertices of the original graph.

Also in the context of the maximum independent set problem, Andrade *et al.* (2012) designed a local search method to compute independent sets. A local search is a technique that, given a solution S_i of the problem, it always tries to improve the current solution in the next iteration S_{i+1} of the algorithm. Andrade *et al.*'s (2012) local search swaps j vertices from a current independent set with k vertices where $k > j$. Their algorithm guarantees an improvement of an independent set, if possible. Following this line of work, Nogueira *et al.* (2018) designed a local search algorithm for the weighted maximum independent set problem (which consists of finding an independent set I with weighted vertices, such that the sum of the weights in I is maximum). Nogueira *et al.*'s (2018) algorithm, similar to that of Andrade *et al.* (2012), swaps some vertices from a given solution to obtain a better weighted independent set.

In this work, we propose a pioneering research effort to address the maximum 2-packing set problem for arbitrary graphs, which is described next.

4. *Gene2pack*: A genetic algorithm for the maximum 2-packing set problem

Gene2pack produces approximated solutions for the maximum 2-packing set problem for arbitrary graphs. The input of the *gene2pack* algorithm is a (0,1)-adjacency matrix that represents an unweighted undirected graph $G = (V, E)$, where $|V| = n$, and each vertex has a unique integer identifier between 1 and n. The output is a set $S \subseteq V$ that represents a 2-packing set of G.

The representation of each individual in the *gene2pack* algorithm is a binary vector $\vec{x} = (x_1, x_2, \dots x_n) \in \{0,1\}^n$, such that $x_i = 1$ indicates that vertex v_i belongs to S and $x_i = 0$ otherwise, $\forall i \in [1 \dots n]$. Pseudo-code 1 shows an outline of the *gene2pack* algorithm.

Step 1 of Pseudo-code 1 consists in adding a self-edge (loop) to each vertex of the input graph G. This step takes $O(n)$ time to complete. We refer to this graph as G' and to its corresponding adjacency matrix as A'. Step 2 computes $A' \times A'$ and stores the results in matrix C in $O(n^3)$ time. The value of each element c_{ij} of C represents the number of walks of a length of 2 from vertex i to vertex j. Therefore, when $c_{ij} = 0$, the distance between vertices i and j is at least three edges.

Algorithm 1. *Gene2pack* algorithm.

```
Input: An adjacency matrix $A$ corresponding to an undirected connected graph $G = (V, E)$.
Output: A set $S$ that represents the vertices of the 2-packing solution of $G$.
 1: Compute matrix $A'$ obtained by setting every element $\{a_{ii}\}$ of $A$ to 1.
 2: Compute $C = A' \times A'$ and set every element $\{c_{ii}\} = 0$.
 3: (Initialization) Produce a set of $\mu$ individuals at random as the initial population.
 4: (Evaluation) Evaluate the fitness for each individual of the initial population.
 5: repeat
 6:     (Parent selection) Build a pool of $\mu$ individuals using a roulette wheel method.
 7:     (Crossover) Recombine pairs of the selected parents to generate $\lambda = P_c\mu$ offspring, where $P_c$ is the crossover rate.
 8:     (Mutation) Mutate each offspring with probability $P_m$ selected randomly and uniformly.
 9:     (Local improvement) Execute Local_Improvement procedure to the set of offspring.
10:     (Evaluation) Evaluate the fitness of each offspring.
11:     (Survivor selection) Create a new generation of $\mu$ individuals using deterministic fitness-based replacement.
12: until complete $\tau$ generations.
13: return the best individual of the last generation.
```

Step 3 of Pseudo-code 1 generates an initial population of μ random binary vectors as solutions of the input graph G. Encoding and storing a candidate solution as a random $\vec{x}$ require $O(n)$ time; then, encoding the entire initial population requires $O(\mu n)$ time. A random binary vector at this stage is, in general, not a solution for the 2-packing set problem. To guide the search towards a feasible region for the problem, we use the penalty function approach of Back and Khuri (1994) implemented on the fitness function expressed in (1). Based on this function, Steps 4 and 10 evaluate the fitness for each individual $\vec{x}$ of the population in $O(\mu n)$ time,

$$f(\vec{x}) = \sum_{i=1}^{n} (x_i - n \cdot x_i \cdot \sum_{j=1}^{n} x_j c_{ij}). \qquad (1)$$

Let v be a vertex of V in graph G, and assume that $v \in S$. We denote as $S(v)$ ($S^2(v)$) the set of vertices in $N(v)$ ($N^2(v)$) belonging to S. Therefore, the set $C_v = \{S(v) \cup S^2(v)\}$ includes the vertices in conflict with vertex v. In the fitness function expressed in (1), the penalty increases proportionally to the cardinality of C_v. Additionally, the penalty also depends on the closeness of the vertices that cause the conflict, in such a way that the fitness function penalizes more heavily the vertices in $S(v)$ than the vertices in $S^2(v)$. This is only possible due to the extension of the adjacency matrix computed by Steps 1 and 2 of Pseudo-code 1. Figure 1 shows three different candidate solutions for a 2-packing set. Solid vertices (of a candidate solution) are assumed to belong to the 2-packing set whereas hollow vertices are not. Using our fitness function, the configuration of Fig. 1(a) has a penalty greater than that of Fig. 1(b), and for the configuration of Fig. 1(c) there is no penalty.

Regarding evolutionary operators, the *gene2pack* algorithm uses a roulette wheel method in combination with the linear dynamic scaling technique (Step 6 of Pseudo-code 1), where the probability of an individual to be a parent is proportional to its fitness value. It is acknowledged that *gene2pack* can use other parent selection methods (e.g., tournament selection), although preliminary trials showed no significant difference (see Section 5.4). The implementation of the roulette wheel method requires $O(\mu n^2)$ time in each generation. The algorithm uses the well-known operators (one-point crossover and bit-flip mutation) to recombine pairs of individuals from the pool of selected individuals (Step 7) and to flip the value of every bit of $\vec{x}$ with probability P_m (Step 8), respectively. Steps 7 and 8 both require $O(\mu n)$ time for recombining and mutating the entire population in each generation. The algorithm uses a fitness-based replacement scheme (implemented with a sorting algorithm in $O(\mu \log \mu)$ time) to generate a new population from the union of the μ individuals of the previous generation and the λ offspring (Step 11). This scheme introduces elitism into the evolutionary process and guarantees the survival of the fittest individual. The *gene2pack* algorithm iterates until it completes τ generations. After that, the algorithm returns the best

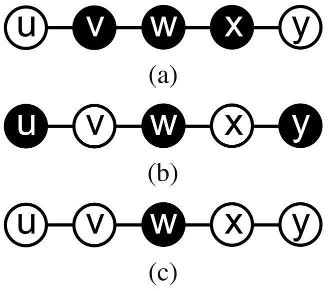


Fig. 1. Example of the evaluation of individuals by the fitness function for the 2-packing set problem.

Algorithm 2. *Local_Improvement* procedure.

```
Input: A genotype $\vec{x}$ of an individual from the population.
Output: An improved genotype $\vec{x}'$.
 1: Decode genotype $\vec{x}$ to a phenotype $S$
 2: if $S = \emptyset$ then
 3:    Randomly choose a vertex from $V$ and insert it into $S$.
 4: end if
 5: Build set $A = \{v | v \in V$ and $t(v) \geq 2\}$.
 6: for each vertex $v \in A$ do
 7:    if $v \in S$ then
 8:       $S = S \setminus v$
 9:    end if
10:    Remove from $S$ one neighbor vertex of $v$.
11: end for
12: Encode $S$ to a new genotype $\vec{x'}$.
13: return $\vec{x'}$
```

individual found in the last generation (Step 13).

Step 9 of Pseudo-code 1 introduces a novel local improvement procedure for the *gene2pack* algorithm. It allows this algorithm to generate higher quality solutions for the maximum 2-packing set problem. Pseudo-code 2 shows the proposed local improvement procedure.

The procedure takes as input a genotype $\vec{x}$ and returns an improved version $\vec{x}'$ (if possible, if not then $\vec{x}' = \vec{x}$). The procedure decodes $\vec{x}$ as a candidate solution S for the 2-packing set problem in $O(n)$ time (Step 1 of Pseudo-code 2). If S is empty, it randomly chooses a vertex $v \in V$ and inserts v into S in $O(1)$ time (Steps 2–4 of Pseudo-code 2). Step 5 of Pseudo-code 2 identifies a set A of vertices with a *tightness* of 2 or more. We say that the *tightness* of any vertex v, denoted as $t(v)$, is the number of vertices in S belonging to the closed neighborhood of v, i.e., $t(v) = |N[v] \cap S|$. Since each vertex evaluates all its neighborhood to determine its tightness, Step 5 can take $O(n^2)$ time for dense graphs. Notice that, in a valid solution for the 2-packing set problem, every vertex $v \in V$ can be $t(v) = 1$ or $t(v) = 0$. Therefore, a way to improve a candidate solution is by deletion of vertices with a tightness 2 or more until a feasible solution is reached. Finally, in Steps 6–11 of Pseudo-code 2, each vertex v of A chooses one neighbor in S and removes it from S in $O(1)$ time. Additionally, if v is in S, v is also removed from it. Applying the *Local_Improvement* procedure to the set of λ offspring takes $O(\lambda n^2)$ time in Step 9 of Pseudo-code 1.

In summary, the overall execution time of the *gene2pack* algorithm is $O(n^3 + \mu n + \tau(\mu n^2 + \mu \log \mu))$.

Figure 2(a) shows an instance graph G with a candidate solution $S = \{r, s, t, w, x, y\}$ for the 2-packing problem. Notice that vertices r, s, and t have a tightness of 1, w and x have a tightness of 2, and the remaining vertices have a tightness of 3. Figure 2(b) shows the same graph after the execution of the local improvement procedure, whose candidate solution is $S' = \{r, t\}$. It is important to note that the candidate solution S' is not a valid 2-packing set; however, it is a better candidate solution than S according to the fitness function.

Next, we prove that the local improvement procedure always improves the fitness of the offspring when there are vertices in conflict.

Let $C_{\vec{x}}$ be the set of vertices in conflict for a 2-packing set solution for individual $\vec{x}$. We say that a vertex v is *valid* in the 2-packing set if $v \notin C_x$. Observe that, when $C_{\vec{x}} = \emptyset$, the 2-packing set is valid. The next results follow from our local improvement procedure.

Lemma 1. *After a local improvement to an individual $\vec{x}$ with vertices in conflict, the set $C_{\vec{x}}$ decreases in cardinality.*

Proof. Let $|C_{\vec{x}}^-|$ be the cardinality of $C_{\vec{x}}$ before local improvement, and let $|C_{\vec{x}}^+|$ be the cardinality of $C_{\vec{x}}$ after local improvement. Notice that all the vertices in $C_{\vec{x}}$ have at least one neighbor in A. After Steps 8 and 10 of Pseudo-code 2, each vertex in A removes one or two vertices from $C_{\vec{x}}$. Thus $|C_{\vec{x}}^-| > |C_{\vec{x}}^+|$. ■

Lemma 2. *For a valid 2-packing set, no local improvement is possible.*

Proof. Since $t(v) < 2$ for all v in $\vec{x}$, the set A is empty. Thus, no local improvement is possible. ■

Lemma 3. *The local improvement procedure does not remove valid vertices.*

Proof. We prove by contradiction. Assume that the local improvement procedure removes a valid vertex v. This is due to the following two cases:

Case a. Vertex v is in the set A. If v is in A, then $t(v) \geq 2$, which is a contradiction.

Case b. Vertex v is not in the set A. Then $\exists u \in N(v)$, such that $t(u) \geq 2$, i.e., $u \in A$. Then, u has at least two neighbors in the 2-packing set, v and some vertex w. Thus, v is not a valid vertex, which is a contradiction.

By contradiction of *Case a* and *Case b*, the result follows. ■

Theorem 1. *The local improvement procedure increases the fitness for each individual with vertices in conflict.*

Proof. The result follows from Lemmas 2 and 3. ■

5. Evaluation

Experiments were conducted to evaluate the performance of *gene2pack* (described in Section 4) for the maximum 2-packing set problem. *Gene2pack* was implemented using DEAP, an open-source evolutionary computation framework (Fortin *et al.*, 2012), version 1.2.2. The experiments were carried on a Mac Pro with the following specifications: 2.7 GHz 12-Core Intel Xeon E5 and 64 GB 1866 MHz DDR3 with a macOS High Sierra operating system, version 10.13.1.

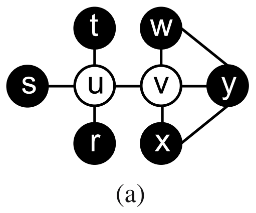
5.1. Objectives.

We conducted three series of experiments to evaluate the effectiveness of *gene2pack*.

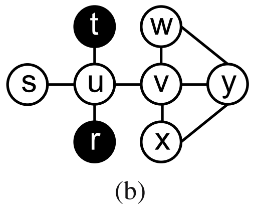
The first series was designed to evaluate the effectiveness of *gene2pack* on arbitrary graphs, namely, connected Erdös–Rényi graphs. We selected these because they are well-known and commonly used to evaluate graph algorithms (Gregor and Lumsdaine, 2005).

The second series of experiments was designed to evaluate the effectiveness of *gene2pack* on cycle graphs of linearly increasing order. A cycle graph (also known as a circular or ring graph) is a simple connected graph where all vertices have a degree 2. We selected cycle graphs because the cardinality of a maximum 2-packing set of a cycle graph C can be mathematically computed as $\lfloor |C|/3 \rfloor$, where $|C|$ is the order of the cycle graph. In doing so, we were able to establish a baseline against which to measure the effectiveness of *gene2pack*.

The third series of experiments was designed to evaluate the effectiveness of *gene2pack* on randomly generated cactus graphs of linearly increasing order. A cactus graph is a connected outerplanar graph where any two simple cycles share at most one vertex. We selected cactus graphs because there is a polynomial-time algorithm to compute the cardinality of maximum 2-packing sets on cactus graph (see Flores-Lamas *et al.*, 2018).



(a)



(b)

Fig. 2. Example of the execution of the local improvement function for an individual $\vec{x}$: the individual $\vec{x}$ before the local improvement (a), the resulting individual $\vec{x}'$ after the local improvement (b).

5.2. Benchmarks. We used three benchmarks to evaluate the effectiveness of *gene2pack*: (i) a comparison with an extended version of a genetic algorithm to compute maximum 2-packing sets on arbitrary graphs proposed by Back and Khuri (1994), hereafter referred to as *the extended BK genetic algorithm*; (ii) a comparison with the optimal solution on Erdös–Rényi graphs and cycle graphs; and (iii) a comparison with an exact algorithm to compute maximum 2-packing sets on cactus graphs proposed by Flores-Lamas *et al*. (2018), hereafter referred to as the *Flores-Lamas algorithm*.

- *Comparing against the extended BK genetic algorithm for arbitrary graphs.* The genetic algorithm proposed by Back and Khuri (1994) is focused on solving the maximum independent set problem; however, we adapted it for the maximum 2-packing set one. The modification consisted of penalizing unfeasible individuals for every vertex that does not belong to a maximum 2-packing set (instead of a maximum independent set). The algorithm proposed by Back and Khuri was extended and implemented for comparison because it (i) also uses an evolutionary approach, (ii) solves a very similar problem, and (iii) could be adapted for the maximum 2-packing set problem relatively easily while keeping its algorithmic structure. With this adaptation, the *extended BK genetic algorithm* is capable of solving the maximum 2-packing set problem. Notice that this algorithm does not include a local improvement procedure. It should be noted that we re-implemented the genetic algorithm proposed by Back and Khuri also using DEAP (Fortin *et al*., 2012). It is worth mentioning that we did not select a genetic algorithm approach (or any other approach) specifically designed for the maximum 2-packing set problem for arbitrary graphs as a benchmark because, to the best of our knowledge, we are the first to address the maximum 2-packing set problem for arbitrary graphs. We would like to indicate that we explored adapting other methods (specifically designed for the maximum independent set problem) such as that of Lamm *et al*. (2017), based on graph reduction rules, and of Andrade *et al*. (2012) and Nogueira *et al*. (2018), based on local search methods. However, these extensions imply major and complex adaptations, which may have resulted in completely different approaches.

- *Comparing against optimal solutions on Erdös-Rényi graphs and cycle graphs.* With respect to Erdös–Rényi graphs (see Section 5.3 for a description of their creation), we computed their maximum 2-packing sets by utilizing a brute-force algorithm. It was implemented using (i) the C programming language to prevent the overhead derived from interpreting code and (ii) openMP to take advantage of the multicore processor where the experiments were carried on. In addition, it should be noted that, due to the exponential growth in execution time when solving NP-hard problems using brute-force algorithms, we obtained maximum 2-packing sets of only relatively small arbitrary graphs ranging in order from 20 to 40 vertices. To illustrate this exponential growth in execution time, the parallel brute-force algorithm obtained maximum 2-packing sets of 40-vertex graphs in approximately 3 to 4 days and, given that the brute-force algorithm faces exponential growth as the number of vertices increases, obtaining a maximum 2-packing set of a 50-vertex graph may take 3,072 days approximately using the same computer where the experiments were carried out. With respect to cycle graphs, as explained in Section 5.1, we obtained the cardinality of their maximum 2-packing sets using a simple mathematical expression ($\lfloor |C|/3 \rfloor$).

- *Comparing against the Flores-Lamas algorithm for cactus graphs.* The polynomial-time algorithm proposed by Flores-Lamas *et al*. (2018) was designed for cactus graphs. Flores-Lamas *et al*. (2018) solve the maximum 2-packing set problem for a unicyclic graph, and then generalize the solution to cactus graphs composed of simple cycles. It should be noted that the *Flores-Lamas algorithm* cannot be fully compared with *gene2pack* because the former works only for cactus graphs and obtains optimal solutions, whereas the latter works for arbitrary graphs (which also include cactus graphs) and obtains (sub)optimal solutions. Nevertheless, the results obtained from the *Flores-Lamas algorithm* enabled us to establish a baseline against which to measure the effectiveness of *gene2pack* for relatively large graphs of up to 1,000 vertices.

5.3. Input data source: Erdös–Rényi and cactus graphs. With respect to Erdös–Rényi graphs, we created a database of connected random graphs ranging in order from 20 to 40 vertices. This type of random graphs has been proved to efficiently describe the structure of some real-world distributed systems (Adamic and Huberman, 2000); e.g., they adequately simulate network motifs in biological networks (Amaral *et al*., 2000; Alon, 2007; Knudsen and Wiuf, 2008). To avoid biased results due to specific structures of random graphs, we generated 50 graphs for each order category, for an overall total of 1,050 connected arbitrary graphs. To ensure that an Erdös–Rényi graph $G = (V, E)$ is connected, we attempted to compute a spanning tree $T = (V_T, E_T)$ such as $|V| = |V_T|$ and rejected the graph if it was not possible. We computed all maximum 2-packing sets for all the connected Erdös–Rényi graphs by using our parallel brute-force algorithm described in Section 5.2.

We also created 20 random cactus graphs ranging from 50 to 1,000 vertices (in steps of 50) using the procedure described by Trejo-Sánchez *et al*. (2018). Unlike Erdös–Rényi graphs, cactus graphs have a relatively regular structure, and that is why the cardinality of their maximum 2-packing sets increased linearly with the number of vertices (see Fig. 6). Hence, we did not create multiple random cactus graphs (as we did for Erdös–Rényi graphs).

It is worth mentioning that, to the best of our knowledge, this is the first database containing connected Erdös–Rényi and cactus graphs with the corresponding cardinality of their maximum 2-packing sets that can be used for benchmarking purposes. This database of graphs (described using GML, a portable file format for graphs) and a summary can be downloaded from `https://github.com/trejoel/Gene2Pack/tree/amcs`.

5.4. Experimental settings. In the first series of experiments both the proposed *gene2pack* algorithm and the *extended BK genetic algorithm* were evaluated using the database of connected Erdös–Rényi graphs described in Section 5.3. In addition, in order to evaluate the effectiveness of the *gene2pack* algorithm as the number of generations and individuals increased, we conducted experiments using populations of (i) 50 individuals evolved for 50 generations, (ii) 100 individuals evolved for 100 generations, and (iii) 200 individuals evolved for 200 generations. In spite of the challenge to accurately estimate the effects of any change in the operator and parameter settings of the algorithm, we performed several preliminary trials to find the best settings consistently, which are reported in Table 1. For instance, even though our experiments were executed using roulette selection, we also conducted experiments using tournament selection for 50 Erdös–Rényi graphs of an order of 40. In this series of experiments, *gene2pack* was executed 50 times for each graph using roulette selection and 50 times for each graph using tournament selection, resulting in a total of 5,000 executions. However, the result of a two-tailed t-test showed that there was not a significant difference ($p > 0.05$) between the cardinalities of the maximum 2-packing sets obtained by *gene2pack* using roulette selection and those obtained using tournament selection (with a tournament size of 3). Since both *gene2pack* and the *extended BK genetic algorithms* make use of random search techniques to explore the solution space, each algorithm was executed 50 times for each connected arbitrary graph. This resulted in a total of 157,500 executions of each algorithm.

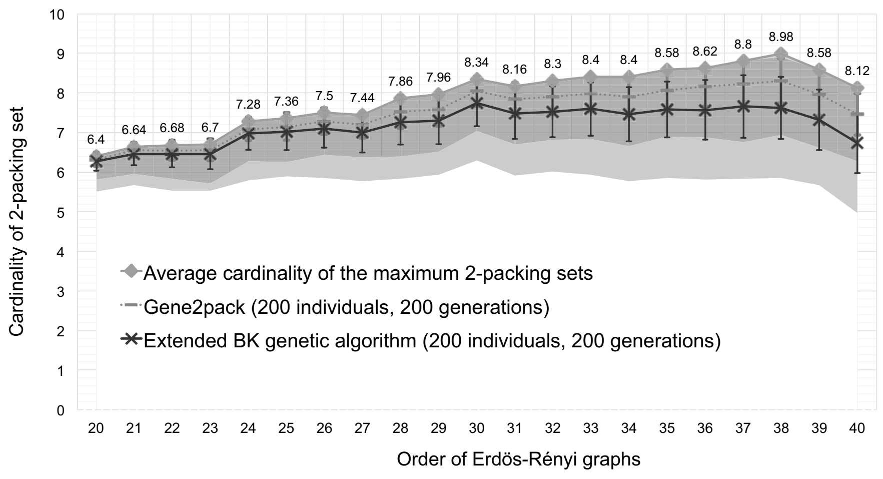
In the second series of experiments, both the proposed *gene2pack* algorithm and the *extended BK genetic algorithm* were evaluated using cycle graphs of an order ranging from 50 to 1,000 in steps of 50. In the third series of experiments, the *gene2pack* algorithm, the *extended BK genetic algorithm*, and the *Flores-Lamas algorithm* were evaluated using cactus graphs (of an order ranging from 50 to 1,000 in steps of 50) described in Section 5.3. It should be noted that we explored the performance of *gene2pack* using cycle and cactus graphs of up 1,000 vertices because these graphs were sufficiently large to guarantee statistical significance and allow us to observe clear patterns in the results. The control parameters of the genetic algorithms were the same as those reported for the first series of experiments. As in the first series, the *gene2pack* algorithm and the *extended BK genetic algorithm* were executed 50 times for each graph. This resulted in a total of 3,000 executions of each algorithm for the second and third series of experiments.

5.5. Performance measures. The performance measures are (i) the average cardinality of the 2-packing sets obtained by the algorithms and (ii) the number of times a maximum 2-packing set was obtained by the algorithms.

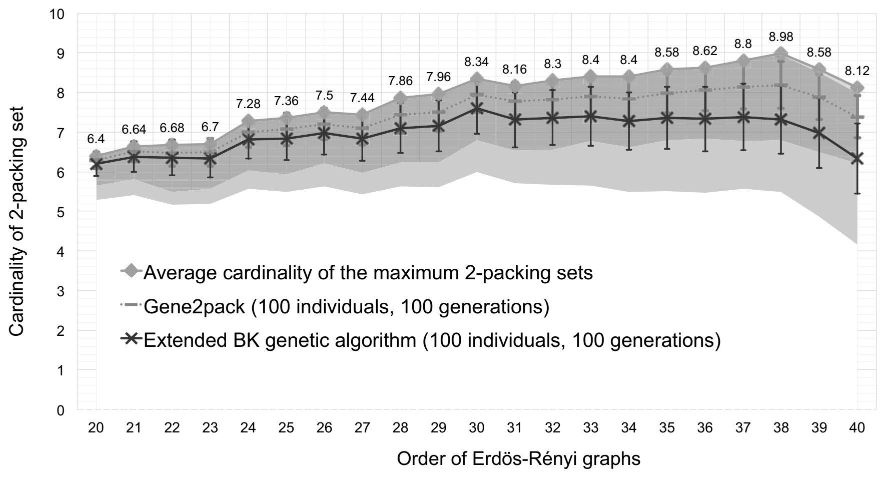
5.6. Results and analysis. Empirical results are shown in Figs. 3–6. Figure 3 depicts the average cardinality of the 2-packing sets obtained by the *gene2pack* algorithm and the *extended BK genetic algorithm* as well as the average cardinality of the maximum 2-packing sets (grouped by order) for connected Erdös–Rényi graphs. Also regarding connected Erdös–Rényi graphs, Fig. 4 shows the number of maximum 2-packing sets obtained by the *gene2pack* algorithm and the *extended BK genetic algorithm* as well as the number of possible maximum 2-packing sets grouped by graph order. Figure 5 shows results related to the cycle graphs also grouped by graph order; in particular, it shows the cardinality of the corresponding maximum 2-packing sets and the average cardinality of the 2-packing sets obtained by the *gene2pack* algorithm and the *extended BK genetic algorithm* for cycle graphs. Figure 6 shows results related to the cactus graphs grouped by graph order, in particular, it shows the cardinality of the corresponding maximum 2-packing sets obtained by the *Flores-Lamas algorithm*

Table 1. Configuration set for the *gene2pack* algorithm.

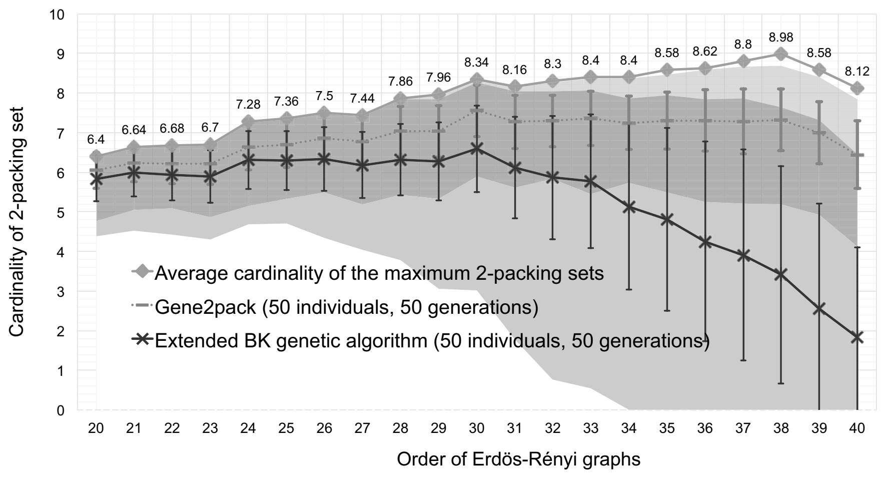
Parameter	Possible values
$(\mu\ ,\ \tau)$: Population size (μ) and number of generations (τ) as completion criterion	$\{(50,50),(100,100),(200,200)\}$
Crossover rate (P_c)	0.9
Mutation rate (P_m)	0.3
Survival selection	Fitness-based $(\mu+\lambda)$



(a)



(b)



(c)

Fig. 3. Overall performance evaluation on Erdös–Rényi graphs.

as well as the average cardinality of the 2-packing sets obtained by the *gene2pack* algorithm and the *extended BK genetic algorithm*. The error bars and error bands of Figs. 3, 5, and 6 represent (i) the standard deviation and (ii) the maximum and minimum values, respectively. In general, the results obtained by the algorithm with the local improvement procedure (i.e., the *gene2pack* algorithm) are better than those obtained by the algorithm with no local improvement (i.e., the *extended BK genetic algorithm*).

From these empirical results (shown in Figs. 3–6), we draw four observations.

Observation 1. *Performance on Erdös–Rényi graphs.* By (on average) obtaining 2-packing sets of higher cardinality for connected Erdös–Rényi graphs of order ranging from 20 to 40, the *gene2pack* algorithm outperformed the *extended BK genetic algorithm* (see Fig. 3).

Analysis. For connected Erdös–Rényi graphs, the average cardinality of the 2-packing sets found by the *gene2pack* algorithm and the *extended BK genetic algorithm* were 7.28 and 6.49, respectively. On average, regardless of the number of individuals and of generations they evolved for, the *gene2pack* algorithm outperformed the *extended BK genetic algorithm* in obtaining 2-packing sets of higher cardinality for each order category of the connected Erdös–Rényi graphs; see Fig. 3. In addition, the *gene2pack* algorithm found 86,518 maximum 2-packing sets (out of the 157,500 experiment runs exploring connected Erdös–Rényi graphs), whereas the *extended BK genetic algorithm* found only 55,258 maximum 2-packing sets (see Fig. 4). However, it is acknowledged that, as shown in Fig. 4, the number of maximum 2-packing obtained by both the *gene2pack* algorithm and the *extended BK genetic algorithm* decreased as the graph order increased. This may be improved increasing the number of generations or the number of individuals (as discussed in Observation 4). The *gene2pack* algorithm achieved better results than the *extended BK genetic algorithm* due to the local improvement procedure specifically designed for the maximum 2-packing set problem. Nonetheless, it should be acknowledged that due to the local improvement procedure, the *gene2pack* algorithm required on average 17.34% more execution time than the *extended BK genetic algorithm*. Finally, it is worth mentioning that we conducted a two-tailed t-test and the results indicate that the means of the cardinality of the 2-packing sets found by the two algorithms are statistically different ($p < 0.001$).

Observation 2. *Performance on cycle graphs.* Whereas, in general, the *extended BK genetic algorithm* obtained feasible solutions for only cycle graphs of order 50 (in addition to two feasible solutions for cycle graphs of order 100), the *gene2pack* algorithm obtained feasible solutions for all the cycle graphs of order ranging from 50 to 1,000 vertices (see Fig. 5).

Analysis. Although the cardinality of a maximum 2-packing set of cycle graphs can be mathematically calculated, neither the *gene2pack* algorithm nor the *extended BK genetic algorithm* were designed taking into account special types of graphs. Regardless of the particularities of cycle graphs and unlike the *extended BK genetic algorithm*, the *gene2pack* algorithm (by using a local improvement procedure) was capable of finding feasible 2-packing sets whose cardinality increased linearly with the number of vertices. It is acknowledged that the 2-packing sets found by the *gene2pack* algorithm were not maximum; however, the larger the graphs, the higher the cardinality of the 2-packing sets.

Observation 3. *Performance on cactus graphs. Gene2pack* obtained 2-packing sets for cactus graphs of up to 1,000 vertices, whereas the *extended BK genetic algorithm* only obtained feasible 2-packing sets for 50-vertex cactus graphs and 100-vertex cactus graphs (although in this case only 3 times out of 150 experiment runs); see Fig. 6.

Analysis. The maximum 2-packing sets of the cactus graphs were obtained using the *Flores-Lamas algorithm*, which is an exact, polynomial-time algorithm. The cardinality of the maximum 2-packing sets of the cactus graphs increased linearly with the graph order. In this regard, unlike the *extended BK genetic algorithm*, the cardinality of the 2-packing sets obtained by *gene2pack* also increased linearly with the graph order. Nevertheless, we acknowledge that the *Flores-Lamas algorithm* outperformed both *gene2pack* and the *extended BK genetic algorithm*. However, whereas *gene2pack* is capable of finding feasible 2-packing sets for Erdös–Rényi graphs (Fig. 3), cycle graphs (Fig. 5), and cactus graphs (Fig. 6), the *Flores-Lamas algorithm* is unable to find 2-packing sets for arbitrary graphs because it relies on assumptions that can only be found in the structure of cactus graphs.

Observation 4. *Impact of generations and population size.* By increasing the number of individuals and of generations these individuals evolved for, the *gene2pack* algorithm found 2-packing sets of higher cardinality.

Analysis. For Erdös–Rényi graphs (Fig. 3), cycle graphs (Fig. 5), and cactus graphs (Fig. 6), the larger the population size and the higher the number of generations, the better the solutions found by *gene2pack*. Moreover, in the case of cycle and cactus graphs, the cardinality of the 2-packing sets obtained increased linearly with the number of vertices. In this regard, it is important to notice that the slope of the results obtained by the *gene2pack* increased as both the number of generations

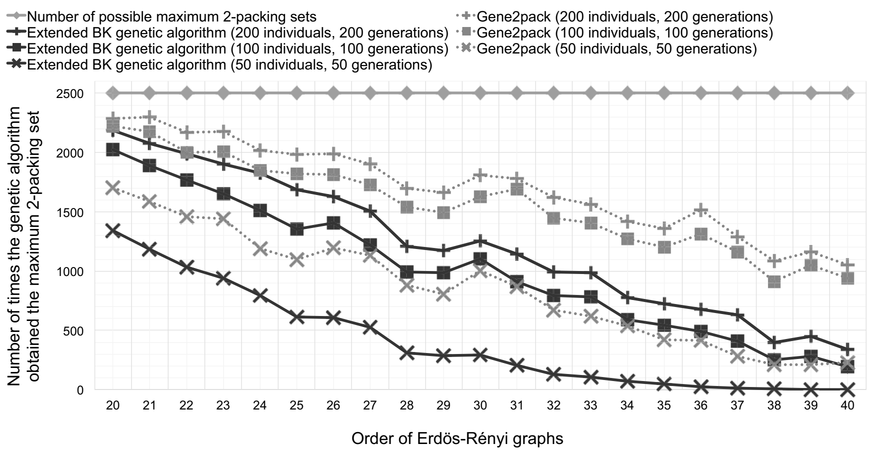


Fig. 4. Number of maximum 2-packing sets obtained for Erdös–Rényi graphs.

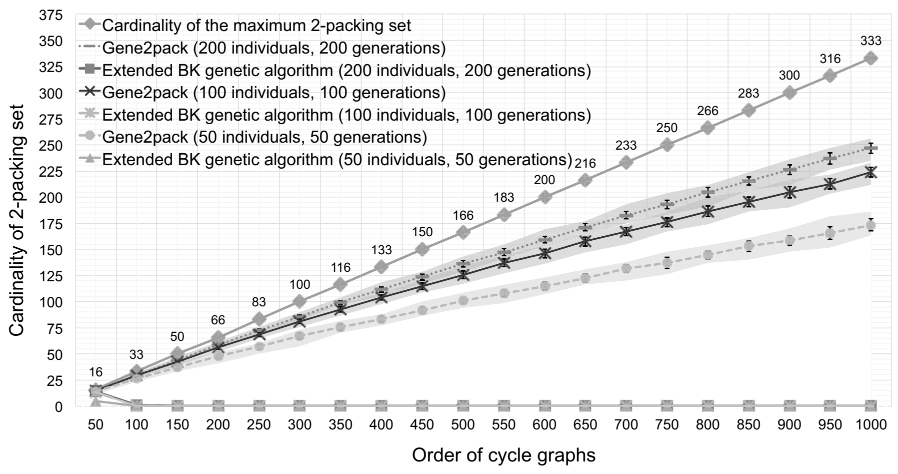


Fig. 5. Overall performance evaluation on cycle graphs.

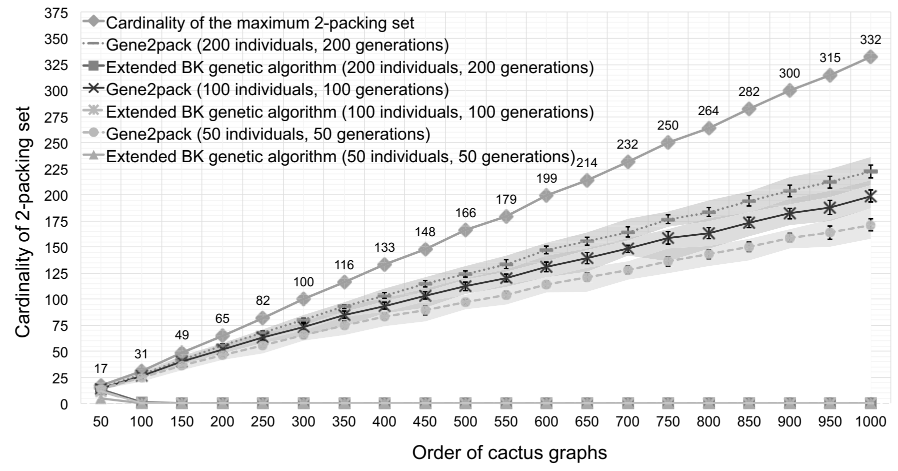


Fig. 6. Overall performance evaluation on cactus graphs.

and the population size increased (see Figs. 5 and 6). These results suggest that the number of generations and/or individuals should be higher for larger graphs in order to obtain 2-packing sets of higher cardinality when using *gene2pack*. In addition, these results also suggest that by adjusting the control parameters of the *gene2pack* algorithm (such as increasing the number of generations), may obtain a maximum 2-packing set for relatively large graphs (of up to 1,000 vertices).

6. Conclusion

The novelty of this work is twofold. First, and most importantly, to the best of our knowledge, it is the earliest research effort to provide a solution for the maximum 2-packing set problem for arbitrary graphs. Second, it contributes the first database containing the cardinality of maximum 2-packing sets of 1,070 graphs, namely, 1,050 Erdös–Rényi graphs and 20 cactus graphs, which can be used for benchmarking purposes. The significance of this research effort is that, by obtaining (maximum) 2-packing sets from arbitrary graphs, our work provides the research community and industry with a tool that can be applied to problems that require distance-two mutual exclusion, e.g., the frequency assignment problem (Hale, 1980).

The main contribution of this work is as follows. We designed and implemented a genetic algorithm for the maximum 2-packing set problem capable of finding feasible 2-packing sets whose cardinality increases linearly with the number of graph vertices. Moreover, the empirical evidence suggests that, as the number of generations and the population size of *gene2pack* increase, the higher the cardinality of the 2-packing sets gets. Furthermore, by using a local improvement procedure, our algorithm outperformed the *extended BK genetic algorithm* (Back and Khuri, 1994) by obtaining 2-packing sets of higher cardinality. In this regard, we also provided a theoretical proof demonstrating that our local improvement procedure increases the fitness for each individual with vertices in conflict.

A potential future research direction is to define graph reduction techniques compatible with the maximum 2-packing set problem. In doing so, our genetic algorithm may find 2-packing sets of higher cardinality in relatively smaller graphs in a relatively shorter time. Another future research direction is to combine our genetic algorithm's local improvement procedure with other neighborhood search heuristics in order to obtain 2-packing sets of higher cardinality.

Acknowledgment

D. Fajardo-Delgado gratefully acknowledges the financial support from Tecnológico Nacional de México (TecNM) under the project 6307.17-P and from PRODEP through the research group ITCGUZ-CA-7. J.O. Gutierrez-Garcia gratefully acknowledges the financial support from Asociación Mexicana de Cultura, A.C.

References

Adacher, L. and Gemma, A. (2017). A robust algorithm to solve the signal setting problem considering different traffic assignment approaches, *International Journal of Applied Mathematics and Computer Science* **27**(4): 815–826, DOI: 10.1515/amcs-2017-0057.

Adamic, L.A. and Huberman, B.A. (2000). Power-law distribution of the world wide web, *Science* **287**(5461): 2115–2115.

Agrawal, A., Klein, P. and Ravi, R. (1995). When trees collide: An approximation algorithm for the generalized Steiner problem on networks, *SIAM Journal on Computing* **24**(3): 440–456.

Alon, U. (2007). *An Introduction to Systems Biology: Design Principles of Biological Circuits*, Chapman & Hall/CRC, Boca Raton, FL.

Amaral, L.A.N., Scala, A., Barthélémy, M. and Stanley, H.E. (2000). Classes of small-world networks, *Proceedings of the National Academy of Sciences* **97**(21): 11149–11152.

Andrade, D.V., Resende, M.G. and Werneck, R.F. (2012). Fast local search for the maximum independent set problem, *Journal of Heuristics* **18**(4): 525–547.

Back, T. and Khuri, S. (1994). An evolutionary heuristic for the maximum independent set problem, *IEEE World Congress on Computational Intelligence, Orlando, FL, USA*, pp. 531–535.

Baran, M. (2018). Closest paths in graph drawings under an elastic metric, *International Journal of Applied Mathematics and Computer Science* **28**(2): 387–397, DOI: 10.2478/amcs-2018-0029.

Eiben, A.E. and Smith, J.E. (2015). *Introduction to Evolutionary Computing*, Natural Computing Series, 2nd Edn, Springer-Verlag, Berlin/Heidelberg.

Feitelson, D.G. (1996). Packing schemes for gang scheduling, *in* D. G. Feitelson and L. Rudolph (Eds), *Workshop on Job Scheduling Strategies for Parallel Processing*, Springer, Berlin/Heidelberg, pp. 89–110.

Flores-Lamas, A., Fernández-Zepeda, J.A. and Trejo-Sánchez, J.A. (2018). Algorithm to find a maximum 2-packing set in a cactus, *Theoretical Computer Science* **725**: 31–51.

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M. and Gagné, C. (2012). DEAP: Evolutionary algorithms made easy, *Journal of Machine Learning Research* **13**(7): 2171–2175.

Gairing, M., Geist, R.M., Hedetniemi, S.T. and Kristiansen, P. (2004a). A self-stabilizing algorithm for maximal 2-packing, *Nordic Journal of Computing* **11**(1): 1–11.

Gairing, M., Goddard, W., Hedetniemi, S.T., Kristiansen, P. and McRae, A.A. (2004b). Distance-two information in self-stabilizing algorithms, *Parallel Processing Letters* **14**(03n04): 387–398.

Garey, M.R. and Johnson, D.S. (2002). *Computers and Intractability*, Vol. 29, WH Freeman New York, NY.

Gregor, D. and Lumsdaine, A. (2005). The parallel BGL: A generic library for distributed graph computations, *Parallel Object-Oriented Scientific Computing (POOSC), Glasgow, UK,* pp. 1–18.

Hale, W.K. (1980). Frequency assignment: Theory and applications, *Proceedings of the IEEE* **68**(12): 1497–1514.

Hochbaum, D.S. and Shmoys, D.B. (1985). A best possible heuristic for the k-center problem, *Mathematics of Operations Research* **10**(2): 180–184.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, University of Michigan Press, Ann Arbor, MI.

Karp, R.M. (1972). Reducibility among combinatorial problems, *in* R.E. Miller *et al.* (Eds), *Complexity of Computer Computations*, Springer, Boston, MA, pp. 85–103.

Knudsen, M. and Wiuf, C. (2008). A Markov chain approach to randomly grown graphs, *Journal of Applied Mathematics* **2008**: 1–14.

Lamm, S., Sanders, P., Schulz, C., Strash, D. and Werneck, R.F. (2017). Finding near-optimal independent sets at scale, *Journal of Heuristics* **23**(4): 207–229.

Manne, F. and Mjelde, M. (2006). A memory efficient self-stabilizing algorithm for maximal k-packing, *in* A.K. Datta and M. Gradinariu (Eds), *Symposium on Self-Stabilizing Systems*, Springer, Berlin/Heidelberg, pp. 428–439.

Mjelde, M. (2004). *k-Packing and k-Domination on Tree Graphs*, Master's thesis, University of Bergen, Bergen.

Newman, M.E.J. (2002). *Handbook of Graphs and Networks*, Wiley-VCH Verlag GmbH & Co. KGaA, Weinheim, DOI: 10.1002/3527602755.

Nogueira, B., Pinheiro, R.G. and Subramanian, A. (2018). A hybrid iterated local search heuristic for the maximum weight independent set problem, *Optimization Letters* **12**(3): 567–583.

Shi, Z. (2012). A self-stabilizing algorithm to maximal 2-packing with improved complexity, *Information Processing Letters* **112**(13): 525–531.

Soto, J.G., Leanos, J., Ríos-Castro, L. and Rivera, L. (2018). The packing number of the double vertex graph of the path graph, *Discrete Applied Mathematics* **247**: 327–340.

Trejo-Sánchez, J. A., Vela-Navarro, A., Flores-Lamas, A., López-Martínez, J.L., Bermejo-Sabbagh, C., Cuevas-Cuevas, N.L. and Toral-Cruz, H. (2018). Fast random cactus graph generation, *in* M. Torres *et al.* (Eds), *International Conference on Supercomputing in Mexico*, Springer, Cham, pp. 129–136.

Trejo-Sánchez, J.A. and Fernández-Zepeda, J.A. (2012). A self-stabilizing algorithm for the maximal 2-packing in a cactus graph, *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), Shanghai, China*, pp. 863–871.

Trejo-Sánchez, J.A. and Fernández-Zepeda, J.A. (2014). Distributed algorithm for the maximal 2-packing in geometric outerplanar graphs, *Journal of Parallel and Distributed Computing* **74**(3): 2193–2202.

Trejo-Sánchez, J.A., Fernández-Zepeda, J.A. and Ramírez-Pacheco, J.C. (2017). A self-stabilizing algorithm for a maximal 2-packing in a cactus graph under any scheduler, *International Journal of Foundations of Computer Science* **28**(08): 1021–1045.

Turau, V. (2012). Efficient transformation of distance-2 self-stabilizing algorithms, *Journal of Parallel and Distributed Computing* **72**(4): 603–612.

Joel Antonio Trejo-Sánchez received his PhD in computer sciences from CICESE in 2014. He is research fellow at CONACyT—Centro de Investigación en Matemáticas. His research interests include combinatorial optimization as well as distributed and parallel computing.

Daniel Fajardo-Delgado received his MSc and DSc degrees from CICESE, Mexico, in 2011. He is currently a titular professor in the Department of Systems and Computation, Instituto Tecnológico de Ciudad Guzmán, Jalisco, Mexico. His research interests include evolutionary algorithms, algorithmic game theory, and self-stabilization.

J. Octavio Gutierrez-Garcia received his PhD in electrical engineering and computer science from CINVESTAV and the Grenoble Institute of Technology, respectively. Currently, he is a tenured professor in the Department of Computer Science at ITAM. His research interests include computational intelligence and distributed systems.

Received: 1 February 2019
Revised: 13 August 2019
Accepted: 7 September 2019