Solving the d-Distance p-Packing Problem via Integer Linear Programming and Approximation Algorithms

Zahra Hamed-Labbafian [a,*] Sandi Klavžar [b,c,d,†] Mostafa Tavakoli [a,‡]

[a] Department of Applied Mathematics, Faculty of Mathematical Sciences, Ferdowsi University of Mashhad, Mashhad, 91775 Iran

[b] Faculty of Mathematics and Physics, University of Ljubljana, Slovenia

[c] Institute of Mathematics, Physics and Mechanics, Ljubljana, Slovenia

[d] Faculty of Natural Sciences and Mathematics, University of Maribor, Slovenia

Abstract

Let $G = (V(G), E(G))$ be a graph. A set $S \subseteq V(G)$ is a d-distance p-packing if every vertex of G is within distance at most d from at least one vertex in S, and the distance between every pair of distinct vertices in S is greater than p. The d-distance p-packing number $\gamma_d^p(G)$ of G is the minimum cardinality of such a set. Since determining $\gamma_d^p(G)$ is an NP-hard problem, designing efficient solution approaches is essential, particularly for large-scale graphs. In this paper, three complementary approaches for solving this problem are investigates. First, an integer linear programming (ILP) model is proposed. Next, two greedy initialization heuristics based on the betweenness centrality and the minimum eccentricity are introduced to rapidly construct feasible initial solutions. These initial solutions are then refined by a tabu search metaheuristic embedded within a binary search framework to obtain smaller feasible d-distance p-packing sets. Computational experiments demonstrate that the proposed tabu search algorithm consistently produces high-quality solutions and matches the optimal solutions whenever the exact values are available, while remaining effective for larger graph instances.

Keywords: d-distance p-packing set; d-distance p-packing number; integer linear programming; tabu search algorithm
AMS Subj. Class. (2020): 05C12, 05C85

*Email: hamedlabbafianzahra@gmail.com
†Email: sandi.klavzar@fmf.uni-lj.si
‡Corresponding author, Email: m_tavakoli@um.ac.ir

1 Introduction

Graph domination theory is one of the central topics in graph theory and has attracted considerable attention due to its numerous applications in network design, facility location, wireless communication, social network analysis, and combinatorial optimization. Over the past decades, many variants of domination have been introduced to model different practical requirements encountered in optimization problems on graphs. To start with, we recommend the following recent book to readers [13].

One of the most important generalizations of classical domination is *distance domination*. Let d be a positive integer. A set $D \subseteq V(G)$ is called a *d-distance dominating set* if every vertex of G is within distance at most d from at least one vertex of D. The minimum cardinality of such a set is called the *d-distance domination number* and is denoted by $\gamma_d(G)$. Distance domination has been extensively studied from both structural and algorithmic perspectives, see the survey [14]. Among these studies, we would like to highlight the work of Hansberg et al. [12] on distance domination and distance irredundance; Henning and Lichiardopol [15] on extremal bounds under degree constraints; and Tian and Xu [18] on distance domination-critical graphs.

Another fundamental concept is graph packing, where selected vertices are required to satisfy prescribed pairwise distance constraints. Packing models naturally arise in applications involving interference avoidance, resource allocation, and communication networks. Consequently, combining domination and packing requirements has become an important research direction in graph optimization. For recent research, see articles [9, 10] (with the same title!).

The above two concepts above were intertwined in 1994 by Beineke and Henning [3] under the name (p, d)-domination number. In the series of papers [5–7], with the intention of placing this general concept within the trends of contemporary graph domination theory, the concept was renamed to the d-distance p-packing domination number. In this problem, the selected vertices must simultaneously dominate the graph within distance d while remaining pairwise separated by a distance greater than p. Formally, a set $S \subseteq V(G)$ is called a *d-distance p-packing*, if every vertex of G is at distance at most d from some vertex of S, and every two distinct vertices of S are at distance greater than p. The minimum cardinality of such a set is called the *d-distance p-packing domination number* and is denoted by $\gamma_d^p(G)$. Special cases include the already mentioned d-distance domination number γ_d^0, the independent domination number γ_1^1 [13], the d-distance independent domination number γ_d^1 [8, 14], the lower packing number γ_2^2, and its generalization, the d-independent d-domination number γ_d^d [16].

Theoretical properties of this parameter have recently received considerable attention. The original paper [5] established fundamental properties, investigated computational complexity, and characterized optimal solutions for several classes of trees. Further developments include the study of strong product graphs [7] and additional structural results for trees and bipartite graphs [6]. Despite these advances, relatively little attention has been devoted to practical algorithms capable of solving the problem efficiently on

medium- and large-scale graphs.

The problem naturally appears in facility location and service network design. Consider an urban transportation network represented by a graph in which vertices correspond to districts and edges represent direct road connections. Public facilities such as fire stations, medical centers, or emergency service units should collectively cover the entire city while remaining sufficiently separated to avoid unnecessary redundancy and inefficient allocation of resources. This optimization objective can be naturally formulated as a d-distance p-packing problem.

In this paper, we propose three complementary approaches for solving the d-distance p-packing problem and compare their performance.

First, we present an exact solution method based on an integer linear programming (ILP) formulation. Although this method guarantees an optimal solution, it becomes computationally expensive and practically infeasible for large and complex graphs due to its high computational complexity.

Next, we introduce a simple greedy procedure that is used solely as an initialization heuristic. Its purpose is to rapidly construct an initial feasible d-distance p-packing, which serves as the starting solution for the tabu search algorithm. The greedy procedure is not intended as a competitive solution method and does not guarantee a packing of minimum cardinality.

Finally, we propose a tabu search metaheuristic to improve the initial solution. The tabu search is embedded within a binary search framework, which iteratively reduces the cardinality of the feasible packing while preserving feasibility. Starting from a feasible solution generated by the greedy initialization improves the robustness of the search process and helps avoid premature termination.

At the end we compare the proposed approaches in terms of solution quality (the size of the resulting packing) and running time. Experimental results demonstrate the effectiveness of the proposed tabu search algorithm relative to the greedy initialization and the exact ILP model.

We conclude the introduction with formal definitions of the concepts we need. Throughout this paper, all graphs are finite, simple, and connected. Let $G = (V(G), E(G))$ be a graph. For a nonempty subset $X \subseteq V(G)$, the subgraph induced by X is denoted by $G[X]$. The distance between u and v, denoted by $d_G(u, v)$, is the length of a shortest u, v-path in G. For a vertex $v \in V(G)$ and an integer $d \geq 0$, the closed d-neighborhood of v is $N_d[v] = \{u \in V(G) : \ d_G(u, v) \leq d\}$. Let d and p be nonnegative integers. A set $S \subseteq V(G)$ is called a *d-distance p-packing* if it satisfies the following conditions:

(i) For every vertex $u \in V(G)$, there exists a vertex $s \in S$ such that $d_G(u, s) \leq d$.
(ii) For every pair of distinct vertices $x, y \in S$, $d_G(x, y) > p$.

The first condition guarantees that every vertex of the graph is covered within distance d, whereas the second condition enforces a minimum separation between every pair of selected vertices. The *d-distance p-packing domination number* of a graph G, denoted by $\gamma_d^p(G)$, is the minimum cardinality of a d-distance p-packing of G.

2 Proposed Algorithms

We now describe three algorithmic approaches for solving the problem.

2.1 Exact Solution via Integer Linear Programming

In this section, we present an exact method for solving the d-distance p-packing problem. We formulate the problem as an Integer Linear Programming (ILP) model, which seeks a feasible set $S \subseteq V(G)$ satisfying both the covering and packing conditions while minimizing $|S|$.

Decision Variables

For each vertex $v \in V = V(G)$, we define a binary variable x_v indicating whether vertex v belongs to the set S or not:

$$x_v = \begin{cases} 1; & \text{if } v \in S, \\ 0; & \text{otherwise.} \end{cases}$$

The objective of the model is to minimize the number of selected vertices. Hence, the objective function is defined as $\min \sum_{v \in V} x_v$, where $x_v \in \{0, 1\}$ for every $v \in V$.

To guarantee the d-covering condition, for every vertex $v \in V$, at least one vertex within its d-neighborhood must be selected, that is the covering constraint is written as $\sum_{u \in N_d[v]} x_u \geq 1$.

To ensure the p-packing condition, any pair of distinct vertices $u, v \in V$ whose distance is at most p cannot be selected simultaneously. This condition is enforced by the constraint $x_u + x_v \leq 1$ for every $u, v \in V$ with $u \neq v$ and $d_G(u, v) \leq p$.

The ILP model is summarized as follows:

$$\begin{aligned} \text{Minimize } & \sum_{v \in V} x_v \\ \text{subject to } & \sum_{u \in N_d[v]} x_u \geq 1, \quad v \in V; \\ & x_u + x_v \leq 1, \qquad u, v \in V,\ d_G(u, v) \leq p,\ u \neq v; \\ & x_v \in \{0, 1\}, \qquad v \in V. \end{aligned}$$

Complexity Analysis

Let $n = |V(G)|$ and $m = |E(G)|$. In the preprocessing, the shortest-path distances between all pairs of vertices are computed to determine the d-neighborhoods and the vertex pairs whose distance is at most p. Using the Floyd–Warshall algorithm, this preprocessing requires $O(n^3)$ time, while using Breadth-First Search (BFS) from every vertex, it requires $O(n(n + m))$ time for unweighted graphs.

The ILP model contains one binary decision variable for each vertex; therefore, the total number of variables is n. The covering constraints introduce n constraints, one for each vertex. In addition, for every pair of vertices satisfying $d_G(u, v) \leq p$, one packing constraint is added. In the worst case, this yields $O(n^2)$ packing constraints. Consequently, the model contains n binary variables and $O(n^2)$ constraints.

Although the size of the mathematical model is polynomial in the number of vertices, solving the resulting binary ILP is NP-hard. Hence, the worst-case running time is exponential in n, depending on the branch-and-bound (or branch-and-cut) search tree explored by the ILP solver. Therefore, the proposed exact approach is suitable for small and medium-sized graph instances, whereas larger instances motivate the use of metaheuristic methods such as tabu search.

Since the proposed tabu search algorithm is embedded within a binary search framework, an initial feasible solution is required to establish a valid upper bound for the search. In the d-distance p-packing problem, however, a graph may admit no feasible solution of a given size k, while feasible solutions of smaller sizes do exist. Consequently, starting the binary search with a trivial upper bound such as $|V(G)|$ may cause the first feasibility test to fail, preventing the search from effectively exploring the solution space.

To address this issue, we employ a simple greedy procedure solely as an initialization heuristic. The purpose of this procedure is not to provide a competitive solution method for the d-distance p-packing problem, but rather to construct an initial feasible solution that serves as the starting point of the tabu search. Beginning from this feasible solution enables the tabu search to iteratively reduce its size while maintaining feasibility, thereby avoiding premature termination and improving the convergence of the overall search process.

It should be emphasized that the greedy procedure is used exclusively for initialization and is not intended as an independent algorithm for solving the problem. Indeed, as demonstrated in the experimental results, both the proposed tabu search and the exact ILP formulation consistently produce superior solutions.

For a small number of benchmark instances, the greedy initialization was unable to construct a feasible solution due to the characteristics of the corresponding parameter settings. In these cases, an initial feasible upper bound was estimated from the solutions of neighboring (d, p) parameter combinations and was used to initialize the tabu search directly. This strategy allowed the search process to proceed successfully without relying on the greedy initialization.

2.2 Greedy Initialization for Tabu Search

Throughout this section, we assume that $p \leq d$, since the proposed greedy initialization is designed for this setting. For instances with $p > d$, the greedy procedure is not guaranteed to construct a feasible initial solution.

The greedy procedure incrementally constructs a set $S \subseteq V = V(G)$ by selecting vertices according to a predefined heuristic while maintaining the packing constraint. After selecting a vertex, all vertices within distance d are removed from further consideration,

and the process continues until no candidate vertices remain.

2.2.1 Algorithm Description

The algorithm maintains two sets:

- S: the current d-distance p-packing being constructed.
- A: the set of *available* vertices that have not yet been covered or removed.

Initially, $S = \emptyset$ and $A = V$. At each iteration, a vertex $u \in A$ is selected according to a given *selection strategy*. The algorithm then checks whether adding u to S violates the p-packing condition, i.e., whether there exists $s \in S$ such that $d_G(u, s) \leq p$. If no such vertex exists, then u is added to S, and all vertices within distance d from u are removed from A (since they are now covered). If adding u violates the packing condition, u is simply discarded and removed from A without being added to S. The process continues until A becomes empty.

Finally, the algorithm verifies the d-domination condition: every vertex in V must be within distance d from some vertex in S. If this holds, the algorithm returns S and its size; otherwise, it reports failure.

2.2.2 Selection Strategies

We implement two different heuristics for selecting the next vertex $u \in A$.

Betweenness Centrality Strategy

Select a vertex with maximum betweenness centrality in the induced subgraph $G[A]$. Betweenness centrality measures the fraction of shortest paths between pairs of vertices that pass through a given vertex. In $G[A]$, for $v \in A$, it is defined as:

$$c_B(v) = \sum_{s \neq v \neq t \in A} \frac{\sigma_{st}(v)}{\sigma_{st}},$$

where σ_{st} is the number of shortest s, t-paths, and $\sigma_{st}(v)$ is the number of such paths containing v. This strategy favors vertices that act as bridges or hubs in the remaining graph.

Minimum Eccentricity Strategy

Select a vertex with minimum eccentricity in the induced subgraph $G[A]$, where the eccentricity of $v \in A$ is:

$$\mathrm{ecc}_A(v) = \max_{w \in A} \mathrm{d}_G(v, w).$$

The minimum eccentricity over A is the *radius* of $G[A]$. Vertices achieving the radius are called *central vertices*. This strategy chooses a vertex that is centrally located, potentially covering the remaining vertices more evenly.

In case of ties, both strategies randomly choose among the candidate vertices to increase diversity.

2.2.3 Complexity Analysis

Let $n = |V(G)|$ and $m = |E(G)|$. Precomputation of all-pairs shortest paths requires $O(n(m + n \log n))$ using Dijkstra's algorithm (or $O(n^3)$ with Floyd-Warshall). Each iteration removes at least one vertex from A, so at most n iterations are performed. In each iteration:

- **Betweenness strategy**: $O(|A| \cdot (m_A + |A| \log |A|))$ using Brandes' algorithm, where m_A is the number of edges in $G[A]$.
- **Eccentricity strategy**: $O(|A|^2)$ to compute all eccentricities.

Thus, the overall complexity is dominated by the betweenness strategy, leading to $O(n^2 m)$ in the worst case. In practice, for moderate n, the algorithm runs quickly.

The algorithm steps are presented in Algorithm 1.

The greedy initialization procedure does not guarantee a minimum-cardinality d-distance p-packing. Its primary purpose is to rapidly construct an initial feasible solution for the tabu search algorithm. As demonstrated in the computational experiments, the subsequent tabu search substantially improves the quality of this initial solution while retaining its feasibility.

Therefore, in the next section, we present the tabu search algorithm.

2.3 Tabu Search Metaheuristic for d-distance p-packing

In this section, we present a tabu search metaheuristic to approximate the minimum d-distance p-packing number $\gamma_d^p(G)$. Tabu search is a local-search-based metaheuristic that explores the solution space by moving from a current solution to a neighboring solution at each iteration. Unlike the exact ILP method, which is computationally prohibitive for large graphs, tabu search provides a fast approximate solution. The algorithm prevents cycling by maintaining a tabu list of recently performed moves that are forbidden for a fixed number of iterations, thereby avoiding repetitive loops.

To maintain diversity and randomness of the algorithm, only the size of the set S obtained from the greedy algorithm (i.e., $k_{\text{start}} = |S_{\text{greedy}}|$) is used as input, not the set S_{greedy} itself. In this way, the tabu search algorithm is not dependent on a specific greedy solution and can explore a much wider search space. Each candidate solution in this problem is represented as a binary vector of length $n = |V(G)|$: $\mathbf{x} = (x_1, \ldots, x_n)$ with $x_i \in \{0, 1\}$, where $x_i = 1$ indicates that vertex v_i belongs to the set S and $x_i = 0$ indicates

Algorithm 1 Greedy Initialization for Tabu Search

Require: Graph $G = (V, E)$, integers $d \geq 0$, $p \geq 0$, selection strategy *strategy* $\in$ $\{\text{betweenness}, \text{min\_eccentricity}\}$.

Ensure: An initial feasible d-distance p-packing set S (or None, ∞ if no feasible solution exists).

Precompute all-pairs shortest paths distances $d_G(u, v)$ for all $u, v \in V$.
Initialize $S \leftarrow \emptyset$.
Initialize $A \leftarrow V$.
while $A \neq \emptyset$ **do**
  Select a vertex $u \in A$ according to the given *strategy* in the induced subgraph $G[A]$.
    (In case of tie, choose randomly among candidates.)
  Check the p-packing condition:
  if $\exists s \in S$ such that $d_G(u, s) \leq p$ **then**
    Reject u: $A \leftarrow A \setminus \{u\}$.
  else
    Accept u: $S \leftarrow S \cup \{u\}$.
    Remove all vertices within distance $\leq d$ from u from A:
      $A \leftarrow A \setminus \{v \in A : \ d_G(u, v) \leq d\}$.
  end if
end while
Verify the d-covering condition:
if $\bigcup_{s \in S} \{v \in V : \ d_G(s, v) \leq d\} = V$ **then**

  return S and $|S|$.
else

  return (None, ∞).
end if

its absence. The size of the set S is equal to the number of ones in the vector, i.e., $|S| = \sum_{i=1}^{n} x_i$. For a fixed size k (where $k < k_{\text{start}}$), the tabu search algorithm generates an initial solution that is not necessarily feasible. This initial solution is constructed randomly: simply select k distinct vertices from the graph uniformly at random and place them into the set. This initial solution may violate the d-covering condition, the p-packing condition, or both. The tabu search algorithm proceeds to find the smallest feasible k (i.e., $\gamma_d^p(G)$) as follows. First, we take $k_{\text{start}} = |S_{\text{greedy}}|$ as an upper bound. Then we start the algorithm with $k = k_{\text{start}} - 1$. For this k, the tabu search algorithm is executed. If a solution with zero cost (a feasible solution) is found, we decrease the value of k and continue the search. For greater efficiency, we use a binary search method: we consider the search range between 1 and k_{start}, and by testing middle values, we quickly converge to the smallest feasible k. For each candidate value of k, the tabu search algorithm is

executed independently and attempts to find a feasible set of that size. After finding the smallest k for which a feasible solution exists, that value is reported as an approximation of $\gamma_d^p(G)$.

The overall procedure of the tabu search algorithm is as follows. The algorithm begins with an initial solution (a binary vector of length n with exactly k ones) generated randomly. This initial solution is evaluated using the cost function described in Subsection 2.3.1. If the cost value is zero, an optimal feasible solution has been found, and the algorithm terminates, returning the corresponding d-distance p-packing set. Otherwise, the algorithm generates the neighbors of the current solution. The neighborhood of a solution S consists of all solutions that can be obtained by swapping exactly one vertex: remove one vertex from S and add one vertex from $V(G) \setminus S$. In other words, each neighbor is obtained by a single swap move (u, v) where $u \in S$ and $v \notin S$. All generated neighbors are evaluated using the same cost function. Among them, the best neighbor (the one with the smallest cost value) that is not in the tabu list is selected. If the best neighbor has a cost lower than the current best-known solution, it is accepted as the new current solution S. The move (swap) corresponding to this neighbor is added to the tabu list to prevent revisiting the same solution in the near future.

The tabu list keeps track of recent moves to avoid going around in circles. Each move stays in the tabu list for a certain number of steps (called tabu tenure). After that number of steps, the move is automatically removed from the tabu list and can be used again. However, there is an exception: even if a move is still in the tabu list, it can be accepted if it leads to a solution that is better than the best solution found so far. This exception is called the aspiration criterion. In short, a move leaves the tabu list either when its tabu time is over, or immediately if it gives a better solution than the current best.

The algorithm then repeats the process from the beginning with this new current solution. This iterative process continues until a stopping condition is met. The stopping conditions in this problem are: (i) a solution with zero cost (a fully feasible d-distance p-packing) is found, or (ii) the maximum number of iterations is reached. The algorithm steps are presented in Algorithm 2.

Algorithm 2 Tabu search algorithm for d-distance p-packing

Require: $G = (V, E)$, integers d, p, max iterations $Maxit$, tabu tenure t_{tenure}, set size k.
Ensure: A d-distance p-packing set S and its size $|S|$ (or best feasible solution found).

Precompute all-pairs shortest paths distances $\text{dist}_G(u, v)$ for all $u, v \in V$.
Precompute violation pairs (distance $\leq p$) and domination sets (distance $\leq d$).
Generate a random initial solution S of size k (binary vector with exactly k ones).
Compute $\text{cost}(S) = (\#\text{packing violations}) + (\#\text{undominated vertices})$.
$best\_solution \leftarrow S$
$best\_cost \leftarrow \text{cost}(S)$
Initialize an empty tabu list (FIFO queue) of size t_{tenure}
$count \leftarrow 1$
while $count \leq Maxit$ **do**
    Generate neighbors of S by swapping one vertex:
        $\mathcal{N}(S) = \{(S \setminus \{u\}) \cup \{v\} : u \in S, v \notin S\}$ (or sample randomly if too many).
    Using cashing, evaluate all neighbors.
    Sort neighbors by cost (ascending) and select the best one.
    Let $best\_neighbor$ be the best neighbor and $move$ its corresponding swap (u, v).
    if $move$ is not in tabu list **then**
        Accept $best\_neighbor$ as the new solution.
    else if $best\_neighbor\_cost < best\_cost$ **then**
        Accept $best\_neighbor$ as the new solution (aspiration criterion overrides tabu).
    else
        Reject $best\_neighbor$ and select the next best neighbor not in tabu list (or until a valid neighbor is found).
        If no valid neighbor exists, break the loop.
    end if
    if a neighbor was accepted **then**
        $S \leftarrow best\_neighbor$
        $S_{cost} \leftarrow best\_neighbor\_cost$
        if $S_{cost} < best\_cost$ **then**
            $best\_solution \leftarrow S$
            $best\_cost \leftarrow S_{cost}$
        end if
        Add $move$ to the tabu list (FIFO).
        if tabu list size $> t_{tenure}$ **then**
            Remove the oldest move from the tabu list.
        end if
    end if
    $count \leftarrow count + 1$
end while
Return $best\_solution$ with the best (smallest) cost as the best solution.
    (If $best\_cost = 0$, a feasible d-distance p-packing is found.)

2.3.1 Incremental Matrix-Based Fitness Evaluation

To efficiently evaluate neighboring solutions during the tabu search procedure, we employ an incremental matrix-based representation of the objective function. Instead of recomputing the fitness value from scratch after each neighborhood move, only the components affected by the performed move are updated. This significantly reduces the computational cost of evaluating candidate solutions.

Coverage Matrix

Let $n = |V(G)|$ and let S denote the current solution. We define a binary matrix $D \in \{0,1\}^{n\times n}$, whose entries are given by

$$D_{ij} = \begin{cases} 1; & \text{if } i \in S \text{ and } d_G(i,j) \le d, \\ 0; & \text{otherwise,} \end{cases}$$

that is, $D_{ij} = 1$ indicates that the selected vertex i covers vertex j.

The covering constraint is verified by inspecting each column of D. A vertex is considered covered whenever its corresponding column contains at least one nonzero entry. Consequently, the coverage penalty is defined as

$$\text{Penalty}_D = \sum_{u\in V(G)} I\left(\sum_{i=1}^{n} D_{iu} = 0\right),$$

where $I(\cdot)$ denotes the indicator function. Hence, Penalty_D equals the number of uncovered vertices.

Conflict Matrix

To represent violations of the packing constraint, we introduce another binary matrix $P \in \{0,1\}^{n\times n}$, whose entries are defined as

$$P_{ij} = \begin{cases} 1; & i,j \in S,\ i \ne j,\ d_G(i,j) \le p, \\ 0; & \text{otherwise.} \end{cases}$$

Since the distance relation is symmetric, matrix P is symmetric as well. Each nonzero entry indicates that the corresponding pair of selected vertices violates the required packing distance. The corresponding packing penalty is computed as

$$\text{Penalty}_P = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} P_{ij},$$

where the factor $\frac{1}{2}$ compensates for counting every conflicting pair twice.

Fitness Function

The quality of a candidate solution is evaluated by combining the violations of both constraints. The fitness function is defined as $\text{fitness}(S) = \text{Penalty}_D + \text{Penalty}_P$. A solution is feasible if and only if $\text{fitness}(S) = 0$, which indicates that both the covering and packing constraints are simultaneously satisfied.

Neighborhood Move

The neighborhood structure preserves the cardinality of the current solution. Given a solution S, one selected vertex $u \in S$ is removed and one unselected vertex $v \notin S$ is inserted, producing the neighboring solution $S' = (S \setminus \{u\}) \cup \{v\}$. Instead of reconstructing the matrices D and P after each move, only the entries affected by the exchanged vertices are updated.

For the coverage matrix, removing u clears the corresponding row of D, while inserting v updates only those entries corresponding to vertices within distance d of v. Similarly, for the conflict matrix, removing u eliminates all conflicts involving u by resetting its corresponding row and column. After inserting v, only the distances between v and the vertices of the updated solution are examined to determine whether new packing conflicts arise. Consequently, each neighborhood move requires updating only a small portion of both matrices.

Complexity Analysis

Let $n = |V(G)|$, and let I denote the maximum number of iterations. At each iteration, the tabu search algorithm explores a neighborhood generated by swap moves between one selected and one unselected vertex. When the complete neighborhood is considered, its size is $k(n-k)$, where $k = |S|$. In practice, only a fixed number of neighboring solutions is sampled and evaluated at each iteration, making the computational cost per iteration proportional to the neighborhood sample size.

In a conventional local search implementation, evaluating each neighboring solution requires recomputing the entire fitness function, resulting in a computational complexity of approximately $O(n^2)$ per neighborhood evaluation.

Using the proposed incremental matrix-based evaluation scheme, only the entries affected by the exchanged vertices are updated. Therefore, the complexity of evaluating one neighboring solution is reduced to

$$O(|N_d(u)| + |N_d(v)| + |N_p(u)| + |N_p(v)|)\,,$$

where

$$N_p(x) = \{\, w \in V(G) :\ d_G(x, w) \leq p \,\}.$$

Consequently, if L neighboring solutions are evaluated during each iteration, the overall running time of the tabu search algorithm becomes

$$O(I \cdot L \cdot (|N_d(u)| + |N_d(v)| + |N_p(u)| + |N_p(v)|)) .$$

Since these neighborhoods are typically much smaller than the entire vertex set, the proposed incremental evaluation substantially reduces the computational effort required for neighborhood evaluation. As a result, the tabu search algorithm can explore a significantly larger number of candidate solutions within the same computational time budget.

Although the proposed tabu search algorithm does not guarantee global optimality, it efficiently produces high-quality approximate solutions for medium and large graph instances where exact optimization becomes computationally impractical. The performance of the algorithm depends primarily on parameters such as the tabu tenure, neighborhood size, and the maximum number of iterations.

2.4 Results and Performance Evaluation

To evaluate the correctness and efficiency of the proposed algorithms, experiments were conducted on a 64-bit system equipped with an Intel(R) Core(TM) i5-4300M CPU running at 2.60 GHz. The ILP model was implemented in Python and solved using the CBC (Coin-or Branch and Cut) solver.

Table 1 compares the results obtained by the ILP model, the betweenness centrality (BC) initialization heuristic, the minimum eccentricity (ME) initialization heuristic, and the proposed tabu search algorithm for computing $\gamma_d^p(G)$. Each benchmark graph was evaluated under two different feasible (d, p) parameter settings. Therefore, two rows are reported for each graph, with each row corresponding to one of the selected (d, p) pairs.

To evaluate the correctness of the proposed approach, in the first phase, experiments were conducted on cycle graphs, since the exact values of $\gamma_d^p(G)$ for these graphs are reported in the literature [5]. As observed in the table, the tabu search algorithm successfully obtained the exact optimal values for all these instances. This confirms the correctness of the implementation and the high accuracy of the proposed method. Moreover, in most cases, the approximation algorithms also produced solutions close to the optimal values, although some deviations can be observed in a few instances.

In the second phase, the algorithms were tested on hypercube graphs Q_n, for which the exact values of $\gamma_d^p(G)$ are not available. The results indicate that the tabu search algorithm was able to produce feasible solutions for all tested instances. Although the ILP model successfully solved the instance Q_{10}, it required a considerably longer running time than the proposed tabu search algorithm, which obtained a high-quality solution within a short time. For the larger instance Q_{11}, the ILP model was unable to return a solution within the prescribed time limit, whereas the tabu search algorithm successfully computed a feasible solution. These results clearly demonstrate the superior scalability and computational efficiency of the proposed tabu search approach for large and complex graph instances.

In addition to the cycle and hypercube graphs, we evaluated the proposed algorithms on the graph BN16 (see [1, Figure 1]), the generalized Petersen graph Pet$(12, 2)$ (see [19]),

and the fullerene graphs C_{60}-I_h, C_{100}-D_{5d}-1 and C_{240}-0 selected from the CTCP Program Fullerene Database [17]. For the convenience of the reader, we also refer to [11], where the fullerene graphs $C_{60} - I_h$ and C_{100}-D_{5d}-1 are used as benchmark instances, and to [2], where the C_{240}-0 fullerene is explicitly presented. The latter graphs model the molecular structure of carbon fullerenes and have been extensively studied in graph theory, chemistry, nanotechnology, and materials science. Their highly symmetric yet nontrivial structure makes them a challenging benchmark for optimization problems on graphs. The computational results on fullerene graphs further demonstrate the effectiveness of the proposed algorithms and confirm that the tabu search approach remains robust on this important graph family. The proposed tabu search algorithm matched the exact optimal solution for all tested fullerene graphs except C_{240}-0 For this largest instance, with 240 vertices, it produced a near-optimal solution with an absolute error of only one vertex, while requiring only a fraction of the computational time needed by the exact ILP model.

The greedy algorithms were excluded from the running time comparison, since their computational cost is negligible in practice. Specifically, for small graphs, they terminate in less than one second, while for larger instances they require less than 100 seconds. It is also important to note that the greedy initialization procedure is only applied when the condition $p \leq d$ holds. For instances with $p > d$, the greedy procedure is not guaranteed to construct a feasible initial solution. In such cases, the tabu search algorithm is initialized using an estimated feasible upper bound obtained from neighboring (d, p) parameter combinations. The search then iteratively reduces the size of the solution while preserving feasibility. Consequently, when the gap between the initial solution and the optimal solution is large, more iterations are generally required, resulting in longer running times. This explains the increased execution time observed for some instances in Table 1, particularly when $p > d$.

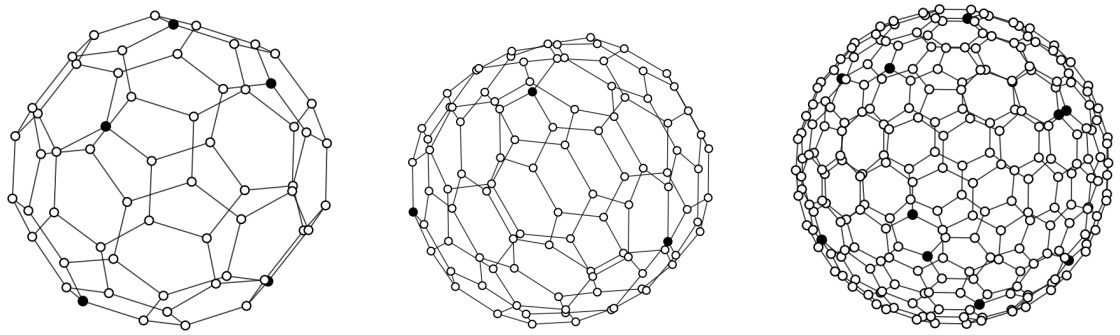
Overall, the experimental results show that the ILP model is suitable for small and medium-sized graphs due to its ability to provide exact solutions in reasonable time. However, as the graph size increases, the computational cost of the ILP model grows rapidly, and it may fail to find a solution within the time limit. In contrast, the tabu search algorithm consistently produces high-quality solutions for all tested instances and, in all cases where the exact values are known, it matches the optimal solutions. Therefore, tabu search can be considered an efficient and reliable method for solving large and complex instances of the d-distance p-packing problem.

Table 1 compares the performance of the exact ILP model, the two greedy initialization heuristics based on betweenness centrality (BC) and minimum eccentricity (ME), and the proposed tabu search algorithm in terms of solution quality and running time. For the fullerene graph C_{60}-I_h with $d = 3$ and $p = 4$, the greedy initialization procedures could not be applied since they are only designed for instances satisfying $p \leq d$. Therefore, the search was initialized directly using the tabu search algorithm.

To determine an appropriate initial solution size, we exploited the result obtained for the previous parameter setting, namely $d = p = 3$, where the optimal solution size was known to be 5. Accordingly, we first attempted to find a feasible solution of size 5 for

the case $d = 3$ and $p = 4$. However, the tabu search algorithm was unable to identify a feasible packing of this size. The initial solution size was therefore increased to 6, and the algorithm successfully found a feasible solution with six vertices in 2.22 seconds. Subsequently, solving the exact ILP model confirmed that $\gamma_3^4(C_{60}\text{-}I_h) = 6$. Hence, the solution obtained by the tabu search algorithm coincides with the optimal solution.

To illustrate the computed optimal d-distance p-packing sets, Fig. 1 presents three representative fullerene graphs together with their corresponding optimal solutions. The black vertices indicate the vertices belonging to the optimal d-distance p-packing set. The optimal packing numbers are $\gamma_3^3(C_{60}\text{-}I_h) = 5$, $\gamma_5^5(C_{100}\text{-}D_{5d}\text{-}1) = 3$, and $\gamma_4^4(C_{240}\text{-}0) = 10$.



(a) $\gamma_3^3(C_{60}\text{-}I_h)= 5$. (b) $\gamma_5^5(C_{100}\text{-}D_{5d}\text{-}1) = 3$. (c) $\gamma_4^4(C_{240}\text{-}0)= 10$.

Figure 1: Optimal d-distance p-packing sets for three benchmark fullerene graphs. Black vertices represent the vertices included in the optimal packing set.

Funding statement

Sandi Klavžar was supported by the Slovenian Research and Innovation Agency (ARIS) under the grants P1-0297, N1-0355, N1-0431, J1-70045.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Authorship contributions statement

References

[1] M. Afkhami, Z. Hamed-Labbafian, S. Klavžar, M. Tavakoli, Packing chromatic number of unitary Cayley graphs of $\mathbb{Z}_n$ and algorithmic approaches to it, Quaest. Math. 49 (2026) 987–999.

[2] K. Balasubramanian, Combinatorics of edge symmetry: Chiral and achiral edge colorings of icosahedral giant fullerenes: C80, C180, and C240, Symmetry 12 (2020) 1308.

[3] L.W. Beineke, M.A. Henning, Some extremal results on independent distance domination in graphs, Ars Combin. 37 (1994) 223–233.

[4] J.A. Bondy, U.S.R. Murty, Graph Theory, Springer, 2008.

[5] Cs. Bujtás, V. Iršič Chenoweth, S. Klavžar, G. Zhang, The d-distance p-packing domination number: complexity and trees, Aequationes Math. 100 (2026) Paper 25.

[6] Cs. Bujtás, V. Iršič Chenoweth, S. Klavžar, G. Zhang, Revisiting d-distance (independent) domination in trees and in bipartite graphs, Discrete Math. 349 (2026) Paper 114972.

[7] Cs. Bujtás, V. Iršič Chenoweth, S. Klavžar, G. Zhang, On d-distance p-packing domination number in strong products, Ann. Combin. (2026) `doi.org/10.1007/s00026-026-00814-0`.

[8] J.G. Gimbel, M.A. Henning, Bounds on an independent distance domination parameter, J. Combin. Math. Combin. Comput. 20 (1996) 193–205.

[9] Á. Dúcz, A. Gujgiczer, Domination and packing in graphs, `arXiv:2602.18402` (2026).

[10] R. Gómez, J. Gutiérrez, Domination and packing in graphs, Discrete Math. 348 (2025) Paper 114393.

[11] Z. Hamed-Labbafian, M. Tavakoli, Algorithmic methods for determining the strong edge chromatic and Wiener indices of graphs, Fullerenes Nanotubes Carbon Nanostruct. 34 (2026) 899–904.

[12] A. Hansberg, D. Meierling, L. Volkmann, Distance domination and distance irredundance in graphs, Electron. J. Combin. 14 (2007) R35.

[13] T. Haynes, S. Hedetniemi, M.A. Henning, Domination in Graphs: Core Concepts, Springer, Cham, 2023.

[14] M.A. Henning, Distance domination in graphs, in: Topics in Domination in Graphs (T.W. Haynes, S.T. Hedetniemi, M.A. Henning, eds.), Springer, Cham (2020) 205–250.

[15] M. A. Henning, N. Lichiardopol, Distance domination in graphs with given minimum and maximum degree, J. Comb. Opt. 34 (2017) 545–553.

[16] M.A. Henning, O.R. Oellermann, H.C. Swart, Bounds on distance domination parameters, J. Combin. Comput. Inf. Sys. Sciences 16 (1991) 11–18.

[17] P. Schwerdtfeger, L. Wirz, J. Avery, CTCP Program Fullerene Database, Centre for Theoretical Chemistry and Physics, Massey University, New Zealand. Available at: `https://ctcp.massey.ac.nz/index.php?group=&menu=home&page=fullerenes` (Accessed July 30, 2026).

[18] F. Tian, J. M. Xu, Distance domination-critical graphs, Appl. Math. Lett. 21 (2008) 416–420.

[19] E. W. Weisstein, Fullerene, MathWorld–A Wolfram Web Resource. Available at: `https://mathworld.wolfram.com/Fullerene.html` (Accessed July 30, 2026).

Table 1: Comparison of ILP, approximation algorithms, and tabu search for computing $\gamma_d^p(G)$

Graph G	$\lvert V(G)\rvert$	d	p	Known Exact Value	ILP Time (s)	BC	ME	TS Time (s)
C_{20}	20	2	2	4	4 0.07	4	6	4 1.56
C_{20}	20	2	3	4	4 0.08	N/A	N/A	4 1.58
C_{50}	50	3	3	8	8 0.10	8	10	8 1.99
C_{50}	50	3	5	8	8 0.11	N/A	N/A	8 1.92
C_{100}	100	11	10	5	5 0.42	8	7	5 2.07
C_{100}	100	11	11	5	5 0.43	8	7	5 2.35
Q_{10}	1024	4	3	N/A	4 1225.77	8	4	4 10.13
Q_{10}	1024	4	4	N/A	4 1482.13	6	4	4 9.57
Q_{11}	2048	4	3	N/A	Timeout	12	24	8 52.70
Q_{11}	2048	4	4	N/A	Timeout	13	24	8 76.41
BN_{16}	32	3	3	N/A	2 0.94	2	2	2 1.58
BN_{16}	32	3	4	N/A	2 0.09	N/A	N/A	2 1.68
Pet$(12, 2)$	24	2	2	N/A	3 0.11	4	4	3 1.65
Pet$(12, 2)$	24	2	3	N/A	3 0.13	N/A	N/A	3 1.71
C_{60}-I_h	60	3	3	N/A	5 0.67	5	6	5 1.76
C_{60}-I_h	60	3	4	N/A	6 1.60	N/A	N/A	6 2.22
C_{100}-D_{5d}-1	100	5	5	N/A	3 1.49	4	6	3 1.79
C_{100}-D_{5d}-1	100	5	6	N/A	3 2.12	N/A	N/A	3 1.77
C_{240}-0	240	4	4	N/A	10 59.877	15	17	10 57.03
C_{240}-0	240	4	5	N/A	10 58.98	N/A	N/A	10 6.18