Journal of Heuristics (2025) 31:17
https://doi.org/10.1007/s10732-025-09552-7

An effective population-based approach for the partial set covering problem

Ye Zhang[1] · Jinlong He[1] · Yupeng Zhou[1] · Shuli Hu[1] · Dunbo Cai[2] · Naiyu Tian[3] · Minghao Yin[1]

Received: 22 May 2024 / Revised: 23 January 2025 / Accepted: 29 January 2025 / Published online: 8 February 2025

Abstract
The partial set covering problem (PSCP) is a significant combinatorial optimization problem that finds applications in numerous real-world scenarios. The objective of PSCP is to encompass a minimum number of subsets while ensuring the coverage of at least n elements. Due to its NP-hard nature, solving large-scale PSCP efficiently remains a critical issue in computational intelligence. To effectively tackle this challenge, we delve into a population-based approach that incorporates a modified tabu search, thereby striking a delicate balance between exploration and exploitation. To further enhance its efficacy, we employ the multiple path-relinking strategy and the fix-and-optimize process. Finally, the dynamic resource allocation scheme is utilized to save computing efforts. Comparative experiments of the proposed algorithm were conducted against three state-of-the-art competitors, across two distinct categories,

✉ Yupeng Zhou
zhouyp605@nenu.edu.cn

✉ Minghao Yin
ymh@nenu.edu.cn

Ye Zhang
zhangy923@nenu.edu.cn

Jinlong He
hejl104@nenu.edu.cn

Shuli Hu
husl903@nenu.edu.cn

Dunbo Cai
caidunbo_yewu@cmss.chinamobile.com

Naiyu Tian
naiyutian@126.com

[1] College of Information Science and Technology, Northeast Normal University, Changchun 130117, China

[2] Center for Technology Research and Innovation, China Mobile (Suzhou) Software Technology Company Ltd., Suzhou 215000, China

[3] Nanjing Research Institute of Electronic Engineering, Nanjing 210007, China

encompassing 150 instances. The results significantly underscore the profound effectiveness of our proposed algorithm, as evidenced by the updating of 67 best-known solutions. Moreover, we conduct an in-depth analysis of the key components inherent to the algorithm, shedding light on their respective influences on the whole performance.

Keywords Partial set covering · Tabu search · Multiple path-relinking · Fix-and-optimize · Dynamic resource allocation

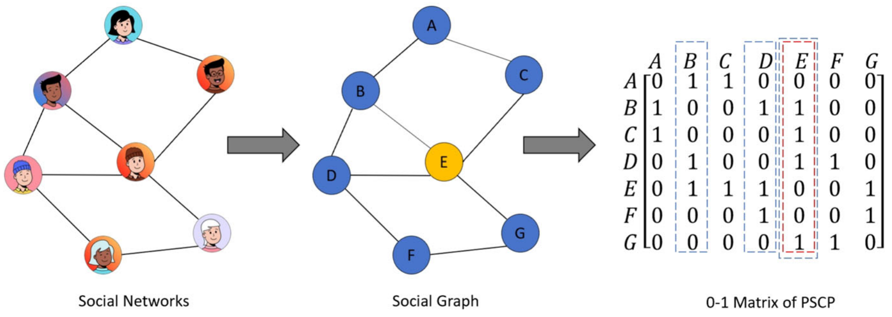
1 Introduction

Covering problems are a fundamental class of optimization problems in mathematics and computer science that involve finding a minimum-cost or minimum-size collection of sets that completely cover a given universe. In the classical set covering problem (SCP), we are given a universe set and a collection of subsets of this universe. The goal is to select a minimum number of subsets such that their union covers the entire universe. This problem finds applications in various fields, including logistics Boschetti and Maniezzo (2015); Alfieri et al. (2007), scheduling Kritter et al. (2019); Yaghini et al. (2015), network design Liao and Ting (2017); Benhaya et al. (2021), etc. SCP also demonstrates its value in resource optimization, showcasing its capability to support comprehensive coverage and optimal configuration under constrained conditions.

Several exact algorithms have been developed to solve the Set Covering Problem (SCP), such as those proposed by Balas and Carrera (1996) and Beasley and Jörnsten (1992), which can guarantee optimal solutions given sufficient runtime. However, these approaches tend to be inefficient, particularly when dealing with large solution spaces. To address this, approximation algorithms have been introduced Hochbaum (1982); Bansal et al. (2010). Unfortunately, in practical applications, these algorithms often fail to deliver satisfactory results in terms of solution quality. As a result, significant research has focused on heuristic approaches, including genetic algorithm Beasley and Chu (1996), simulated annealing Brusco et al. (1999), artificial bee colony Crawford et al. (2014), local search Gao et al. (2014), Gao et al. (2015), Luo et al. (2022), etc. These efforts highlight the considerable attention SCP has garnered and the extensive research conducted in this area.

As the era of big data continues to advance, the massive volume of data is expanding at an unprecedented rate, presenting a more formidable challenge in finding optimal solutions within the constraints of limited computing resources and time. In response to this, the partial set covering problem, a modified version of the well-known minimum set covering problem, has emerged as a viable solution Daskin and Owen (1999); Liu et al. (2022). The partial set covering problem aims to find a minimum-size collection of subsets that covers at least a certain percentage, or fraction, of the universe. This variant takes into account scenarios where covering the entire universe is not necessary, but rather partial coverage is sufficient.

To provide a concrete illustration of the PSCP, Fig. 1 shows an example within the context of social networks. The PSCP can be applied to model the spread of information or influence among individuals Kempe et al. (2003). Consider a social network



$$\begin{array}{c|ccccccc} & A & B & C & D & E & F & G \\ \hline A & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ B & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ C & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ D & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ E & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ F & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ G & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}$$

Fig. 1 A scenario of PSCP within social networks

where each individual is represented as a node, and the connections between individuals are represented as edges. This network can be referred to as a social graph. To formulate this problem as a PSCP instance, we can transform the social graph into a 0-1 matrix, in which each subset represents a group of individuals that can propagate the information to others within the group. Information spreading can be viewed as the process of selecting a subset of individuals to initially receive the information, with the aim of maximizing its overall reach within the network. By solving the PSCP, we can determine the minimum number of individuals needed to reach a certain percentage of the total population. This allows us to optimize the selection process and identify the most influential individuals who can efficiently propagate the information throughout the network, while minimizing the overall dissemination effort. By studying such partial set covering models, we can gain insights into effective strategies for information dissemination, viral marketing, and other related phenomena in social networks.

The partial set covering problem poses a formidable optimization challenge, established as NP-hard Hartmanis (1982). This implies that unless P equals NP, the quest for a polynomial-time algorithm to solve PSCP optimally remains elusive. Consequently, researchers have directed their efforts towards crafting approximation algorithms that yield solutions close to the optimal. Although there are numerous solution methodologies available for the classical Set Covering Problem Beasley and Chu (1996); Caprara et al. (2000); Leutwiler and Corman (2023); Emerick et al. (2023), it is important to note that the literature specifically addressing the algorithmic aspects of the PSCP is relatively limited. Inamdar et al. Inamdar and Varadarajan (2018) innovatively transformed the Partial Set Covering Problem (PSCP) into SCP using natural LP relaxation inspired by geometric set systems, setting the stage for further advancements with their elegantly simple approach. Building upon the foundations laid by Inamdar et al., Chekuri et al. Chekuri et al. (2019) enhanced the approximation ratio of PSCP to the remarkable value of $(1 - 1/e)(\beta + 1)$, employing a framework akin to that of Inamdar and Varadarajan (2018). Here, β is the integrality gap of the linear programming formulation of set covering problems. Afterwards, Ran et al. Ran et al. (2021) designed a parallel algorithm for PSCP which yielded a solution with an approximation ratio of at most $h/(1 - 2\varepsilon)$, where h is the maximum number of sets containing the same elements, and $0 < \varepsilon < 0.5$ is a constant.

It is important to note that PSCP shares a structural resemblance with the classical set covering problem, and there exists substantial evidence highlighting the limitations of approximation algorithms for SCP, as detailed in references Luo et al. (2022) and Wang et al. (2021). In the same way, approximation algorithms may not always meet the solution quality requirements of practical applications. Additionally, as the size of the dataset grows, exact algorithms may require extensive computational efforts, rendering them impractical for real-world scenarios. Consequently, heuristic algorithms have emerged as a promising alternative. These algorithms offer a more practical approach to addressing the PSCP by providing efficient and effective solutions, especially for large-scale problems. While a profusion of heuristic methods has been proposed for tackling diverse set covering variants Chaurasia and Kim (2019); Zhou et al. (2023, 2022), to the best of our knowledge, only two heuristic algorithms have been specifically designed to tackle the challenges posed by the PSCP. One algorithm is the iterated tabu search, featuring a dual-level perturbation technique alongside tabu heuristics Bilal et al. (2014). The other is the Argentine Ant System, leveraging varied populations within a local search framework, characterized by flexible configuration verification and a volatilization-fixed weight mechanism Liu et al. (2022). Experimental results suggest that there is room for improvement in both algorithms. This gap presents an opportunity for future research work, where scholars can begin to explore and design more effective heuristic methods to offer high-quality solutions for PSCP instances encountered in real-world settings.

This paper presents a population-based heuristic framework, namely PSSC, to address the partial set covering problem. PSSC achieves a balance between exploration and exploitation. During the exploitation phase, a modified tabu search (MTS) serves as the foundation for conducting searches with varying intensities. To tackle the issue of local optima, a fix-and-optimize technique is applied to fix specific elements, followed by a greedy repair of the incomplete solution. Regarding population interactions, a multiple path-relinking method facilitates the sharing of knowledge among individuals, guiding some solutions towards different directions. Finally, a dynamic resource allocation strategy is employed to distribute computational resources to promising solutions, thereby enhancing overall efficiency. The performance of PSSC is assessed through comprehensive experiments on 150 well-known instances, including 75 previous ones and 75 new difficult ones. Comparisons with the commercial solver CPLEX, the state-of-the-art local search RWLS, and the population-based method AAS demonstrate the effectiveness of PSSC.

The subsequent sections of this manuscript are structured as follows. Section 2 provides an exposition of the foundational concepts and background knowledge pertinent to this study. In Sect. 3, we present the proposed framework and key components of it. The computational results are reported and analyzed in Sect. 4. Finally, in Sect. 5, we draw the conclusions and summarize the key contributions of this paper.

2 Preliminaries

In this section, we formally introduce the key definitions and notations used throughout the paper.

Set covering problem (SCP) is a well-known NP-hard combinatorial optimization problem, which aims to find a minimum-size collection of subsets that covers the entire universe. The SCP is formulated as follows:

$$\min \sum_{i=1}^{m} x_i \tag{1}$$

$$\text{s.t.} \sum_{i=1}^{m} a_{ij} x_i \geq 1, \qquad \forall e_j \in U \tag{2}$$

$$x_i \in \{0, 1\}, \qquad \forall s_i \in M \tag{3}$$

where $U = \{e_1, ..., e_j, ..., e_u\}$ is the universe, a finite set of elements. $M = \{s_1, ..., s_i, ..., s_m\}$ is the set of subsets of U, also known as the covering set. $a_{ij} = 1$ if subset s_i covers element e_j, and $a_{ij} = 0$ otherwise. x_i is a binary variable indicating whether subset s_i is selected or not.

Partial set covering problem (PSCP) is a variant of the SCP that aims to find the smallest possible collection of subsets that together cover at least a specified percentage, or fraction, of the universe. PSCP can be formulated as follows:

$$\min \sum_{i=1}^{m} x_i \tag{4}$$

$$\text{s.t.} \sum_{s_i \in C_j} x_i \geq z_j, \qquad \forall e_j \in U \tag{5}$$

$$\sum_{e_j \in U} z_j \geq n \tag{6}$$

$$z_j \in \{0, 1\}, \qquad \forall e_j \in U \tag{7}$$

$$x_i \in \{0, 1\}, \qquad \forall s_i \in M \tag{8}$$

where n is the number of desired coverage elements, usually given as a certain percentage of U, C_j is the set collection that covers element e_j. The main distinction between SCP and PSCP lies in their coverage requirements. The SCP mandates that all elements within the universe must be covered, whereas the PSCP only demands that a specified percentage of elements be covered. This makes the PSCP more applicable to various real-world scenarios. It's worth noting that if n equals $|U|$, the PSCP transforms into the SCP.

Maximum set k-covering problem (MSKCP) is a generalization of the set covering problem that aims to find a collection of k subsets that maximizes the number of covered elements in the universe. The MSKCP can be formulated as follows:

$$\max \sum_{e_j \in U} \sum_{i=1}^{m} a_{ij} x_i \tag{9}$$

$$\text{s.t.} \sum_{i=1}^{m} x_i = k \tag{10}$$

$$x_i \in \{0, 1\} \qquad \forall s_i \in M \tag{11}$$

where k is the number of selected subsets. The constraint (10) ensures that exactly k subsets are selected. We introduce MSKCP for the reason that PSCP is effectively solved by iteratively solving the MSKCP. Starting with a large value of k, the MSKCP is solved to find the subset that maximizes the coverage. If the coverage requirement of PSCP is met, the value of k is decremented, and the corresponding MSKCP is solved again. This process continues until the cut-off condition is met. To the best of our knowledge, this research represents the first attempt to convert the task of solving PSCP into a series of stages of solving the MSKCP.

After giving the definitions, we further introduce the reduction rules for solving the set covering problem. Within SCP, two critical concepts are pivotal: column domination and column inclusion. Column domination occurs when a column (subset) can be removed from the solution without impacting coverage, while column inclusion indicates that a column (subset) is essential for covering a specific element. It is evident that the initial rule determines certain subsets that are not required to be included, whereas the latter rule determines subsets that must be included. The details are as follows:

Column domination rule: If the set of rows covered by a column j is also covered by another column j' with a lower cost, then we say that j is dominated by j'. Dominated columns can be removed from M.

Column inclusion rule: If a specific row is covered by only one unique column, then this indispensable column is included in every solution.

These rules are particularly useful for solving the SCP, especially in large-scale instances. In the case of the PSCP, only the first rule remains applicable since not all columns are necessarily required to be covered. Consequently, the proposed algorithm utilizes the column domination rule to reduce the size of PSCP instances before commencing the search procedure.

As the proposed algorithm heavily relies on the score value of subset s to guide the search, we provide the calculation of $score(s)$ as follows:

$$score(s) = \begin{cases} f(S) \cup \{s\} - f(S), & s \notin S \\ f(S) \setminus \{s\} - f(S), & s \in S \end{cases} \tag{12}$$

where $f(S)$ represents the objective function evaluating the solution S in PSCP.

Now, we give an example of PSCP in Fig. 2 to make the problem comprehensible. Given a 0–1 matrix of PSCP, the columns c_1, c_2, c_3 and c_4 represent subsets of the set system (U, M), and the rows e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9 and e_{10} represent elements in U respectively. The term a_{ij} with a mean value of 1 in the matrix means that row i is covered by column j, that is, the subset c_j covers element e_i. Suppose that $n = 9$, the objective of PSCP is to select the minimum number of subsets such that at least 9 elements are covered. Initially, when the candidate solution S is empty, we select

$$\begin{array}{c|cccc|} & c1 & c2 & c3 & c4 \\ e1 & 1 & 1 & 0 & 0 \\ e2 & 1 & 0 & 0 & 1 \\ e3 & 1 & 1 & 0 & 1 \\ e4 & 0 & 0 & 1 & 0 \\ e5 & 1 & 0 & 1 & 0 \\ e6 & 0 & 0 & 1 & 1 \\ e7 & 0 & 0 & 0 & 1 \\ e8 & 0 & 1 & 0 & 1 \\ e9 & 1 & 0 & 0 & 1 \\ e10 & 0 & 0 & 1 & 0 \end{array} \quad \begin{array}{c|ccc:c:|} & c1 & c2 & c3 & c4 \\ e1 & 1 & 1 & 0 & 0 \\ e2 & 1 & 0 & 0 & 1 \\ e3 & 1 & 1 & 0 & 1 \\ e4 & 0 & 0 & 1 & 0 \\ e5 & 1 & 0 & 1 & 0 \\ e6 & 0 & 0 & 1 & 1 \\ e7 & 0 & 0 & 0 & 1 \\ e8 & 0 & 1 & 0 & 1 \\ e9 & 1 & 0 & 0 & 1 \\ e10 & 0 & 0 & 1 & 0 \end{array} \quad \begin{array}{c|cc:c:c:|} & c1 & c2 & c3 & c4 \\ e1 & 1 & 1 & 0 & 0 \\ e2 & 1 & 0 & 0 & 1 \\ e3 & 1 & 1 & 0 & 1 \\ e4 & 0 & 0 & 1 & 0 \\ e5 & 1 & 0 & 1 & 0 \\ e6 & 0 & 0 & 1 & 1 \\ e7 & 0 & 0 & 0 & 1 \\ e8 & 0 & 1 & 0 & 1 \\ e9 & 1 & 0 & 0 & 1 \\ e10 & 0 & 0 & 1 & 0 \end{array}$$

Fig. 2 An example of PSCP with 4 subsets and 10 elements

subset c_4 to be included in the candidate solution set, and update the score values of the remaining subsets accordingly. Consequently, subset c_3 is added to the candidate solution set since it possesses the highest score value of 3. Following this, the constraint of the PSCP is satisfied, and the optimal solution is $S = \{c_3, c_4\}$. It is important to note that this is a simple example that can be calculated manually. However, when dealing with real instances, efficient algorithms are required to solve the problem effectively.

3 The proposed algorithm for PSCP

In this section, we present the effective population-based algorithm, denoted as PSSC, for solving PSCP. The overall framework is provided, followed by a detailed description of its key components for a comprehensive understanding of this method.

3.1 The general framework

Population-based search is a set of optimization algorithms that utilize the power of swarm intelligence or evolutionary principles to solve a wide range of optimization problems Wang and Singh (2008); Prügel-Bennett (2010); Babalola et al. (2020). This approach searches for new candidate solutions through iterative interactions between individuals. In this paper, we present a population-based approach to minimize the selected sets that can cover no less than n elements in PSCP. Within the overall framework, the PSSC algorithm includes steps such as initialization, modified tabu search, multiple path-relinking, fix-and-optimize strategy, and dynamic resource allocation.

The details of the PSSC algorithm are described in Algorithm 1. First, the relevant parameters are initialized (lines 1–2). Then, the candidate solution S is constructed by continuously selecting subsets with the highest score value from the unselected set pool until the number of covered elements is not less than n (lines 3–5). The objective function value $f(S)$ becomes the upper bound of PSCP, and the aim of PSSC is to seek $f(S) - 1$ subsets constantly to meet the constraint in Eq. (6) until the stopping criterion is met. Then, the whole population P is initialized by $Init$ that will be introduced in

Algorithm 1: PSSC framework

```
   Input: Population size $|P|$, elements to be covered $n$, backtrack step $\gamma$, search depth $\theta$.
   Output: The best solution $S^*$ found.
1  $mode \leftarrow 0, cnt \leftarrow 0, iter \leftarrow 0, HC \leftarrow 0$;
2  $S \leftarrow \emptyset, Mark \leftarrow \emptyset$;
3  while $S$ is not feasible do
4      $s \leftarrow \underset{s \in M \setminus S}{\operatorname{argmax}}\, score(s)$;
5      $S \leftarrow S \cup \{s\}$;
6  $S^*, P \leftarrow Init(p, f(S) - 1)$;              // Section 3.2
7  while elapsed time $<$ cutoff time do
8      for each $P_d \in P$ do
9          if $P_d \notin Mark$ then
10             $P_d \leftarrow MTS(P_d, M)$;            // Section 3.3
11         if $P_d.fit >= S^*.fit$ then
12             $cnt \leftarrow 0$;
13             $HC_d \leftarrow HC_d + 1$;
14             $S^* \leftarrow P_d$;
15         if $P_d.fit >= n$ then
16             $HC_d \leftarrow 0$;
17             $S^*, P \leftarrow Init(n, f(P_d) - 1)$;          // Section 3.2
18     $cnt \leftarrow cnt + 1$;
19     if $cnt = 40$ then
20         $cnt \leftarrow 0$;
21         $P_d \leftarrow \underset{P_d \in P}{\operatorname{argmax}}\, HC_d$;
22         $S \leftarrow FO(P_d, f(S^*) * \theta, \gamma)$;          // Section 3.6
23         if $S.fit >= S^*.fit$ then
24             $S^* \leftarrow S$;
25         if $S.fit >= n$ then
26             $HC_t \leftarrow 0$;
27             $S^*, P \leftarrow Init(n, f(S) - 1)$;          // Section 3.2
28     $mode, P \leftarrow MPR(mode, S^*, P, Mark)$;          // Section 3.4
29     if $iter = 5$ then
30         $iter \leftarrow 0$;
31         $Mark \leftarrow DRA(Mark, R_{max}, HC)$;          // Section 3.5
32     $iter \leftarrow iter + 1$;
33 return $S^*$;
```

Sect. 3.2 (line 6). The algorithm proceeds with a series of iterations (lines 7–32). In each iteration, MTS is performed on each individual to generate new solutions, and the population is updated accordingly (lines 8–10). If the fitness value of the obtained individual P_d surpasses that of the current optimal solution S^*, then S^* is updated, and the hit count array is incremented by 1 (lines 11–14). If an individual satisfies the constraint condition of PSCP, it indicates that a candidate solution has been found with the objective function of $f(P_d)$ and will seek better solutions with $f(P_d) - 1$ (lines 15–17). In such cases, the $Init$ function is called to reset and proceed to the next cycle of search. Note that we use the objective function of MSKCP in Eq. (9) as the fitness function and use the objective function of PSCP in Eq. (4) as the final f value.

In every 40 iterations, the fix-and-optimize method is utilized to fix some subsets and try to explore new improved solutions of the well-performed individual (lines 19–22). If a better solution is found, then update the best solution S^* (lines 23–24). Moreover, if this solution is feasible, continue to search the space with $f(S) - 1$ (lines 25–27). If the above condition is not met, the Multiple Path-Relinking strategy is employed to ensure diversity among individuals in the population (line 28). In light of the concern regarding increased computational resource consumption by the population, which can potentially result in inferior search performance within a specified time period, we propose the dynamic resource allocation strategy. This strategy aims to allocate computing resources effectively among the individuals, thereby enhancing overall search capabilities. After the completion of the cycle, the algorithm invokes the DRA to reassign computing resources based on the performance of each individual within the population during that cycle (lines 29–32). Once the termination condition is satisfied, the algorithm returns the best solution obtained during the search process (line 33).

Algorithm 2: Initialization procedure–$Init$

```
Input: Desired coverage elements $n$, upper bound of the selected subsets $k_{max}$.
Output: Local best solution $S'$, initialized $Populatio$.
1 $Population = \{P_0, P_1, ..., P_{|P|}\} \leftarrow \emptyset; t = 0$;
2 $P_t \leftarrow$ add the set $s$ with maximum $score(s)$ with no more than $k_{max}$ times until at least $n$ elements
  are covered;
3 update $Population$;$t = t + 1$;
4 while $t < |P|$ do
5     $t = t + 1$;
6     $P_t \leftarrow$ randomly add one set $s$ with no more than $k_{max}$ times until at least $n$ elements are covered;
7     update $Population$;
8 $S' \leftarrow \underset{P_d \in Population}{\text{argmin}} f(P_d)$;
9 return $S'$, $Population$;
```

3.2 Population initialization

In population-based optimization approaches, the initialization process holds significant importance in determining the quality and diversity of the initial population. When addressing the PSCP, a suitable population initialization procedure is crucial to establish a solid foundation for an effective optimization process. The objective of this procedure is to generate an initial population that possesses favorable characteristics for solving the PSCP.

Algorithm 2 presents the pseudocode for this population initialization procedure. First, an empty population is created, and a counter variable t is set to 0 to traverse the population (line 1). In the greedy mode (line 2), an individual is generated by selecting the set with the maximum value of the $score$. This process continues no more than k_{max}

times until the individual becomes feasible, satisfying Eq. (5). Then, the population is updated and k_{max} will also be updated with a smaller value if possible (line 3). To strike a balance between exploration and exploitation, the remaining individuals are initialized in random mode by selecting a random set with no more than k_{max} times until at least n elements are covered (lines 4–7). This random mode generates $|P| - 1$ individuals. Once all individuals have been initialized, the best solution S' is recorded (line 8). Finally, the best solution S' along with the entire population is returned for the subsequent stages (line 9). By employing this appropriate population initialization procedure, the proposed algorithm begins with a diverse and promising initial population, laying the groundwork for subsequent search and improvement operations.

3.3 The modified tabu search

The systematic change of neighborhood within a local search algorithm presents a straightforward yet highly effective metaheuristic for solving combinatorial optimization problems Mladenović and Hansen (1997). Employing a methodology termed Modified Tabu Search (MTS), this approach systematically ventures through diverse neighborhoods adjacent to a particular solution in a sequential manner. This process facilitates the algorithm's ability to escape local optima, thereby discovering better solutions. To clarify, we define three distinct neighborhood sets: $N_1(x) = RS(S, R = 1, TS)$, $N_2(x) = RS(S, R = 2, TS)$ and $N_3(x) = RS(S, R = 3, TS)$. By changing the neighborhood structure, the algorithm can explore the search space efficiently and balance exploration and exploitation. This flexibility allows the MTS to adapt to different problem instances and their unique characteristics, rendering it a robust and flexible optimization framework.

We utilize MTS to improve the quality of individual solutions in the population. Algorithm 3 describes the detailed process of the modified tabu search for PSCP. Given the current solution S, the algorithm initializes the local best solution S' as S, sets the search intensity R to 1, and initializes the set of feasible solutions $\hat{S}$ to an empty set (line 1). The algorithm then proceeds with variable neighborhood descent at different intensities to find the optimal solution S'. To be specific, the algorithm executes multiple rounds, each of which begins with setting the search intensity to 1 and recording the objective function of S' which serves as the lower bound of the solutions obtained through the search. Subsequently, the algorithm performs a neighborhood search with the search intensity R. It resets the tabu list TS to an empty set, initializes the iteration counter $iter$ to 0, and sets the tabu value $tabu(s) = -1$ for any set $s \in M'$. Additionally, the algorithm sets the counter t to 0, which keeps track of the number of iterations in which S has not been updated (lines 3–5). The algorithm continues to iterate unless t is not less than ϵ. At the beginning of each iteration, it identifies all sets in the tabu state and adds them to the tabu list TS. A set s is considered in tabu state only if the current iteration number $iter$ is less than or equal to $tabu(s)$. In this case, s is not allowed to be removed from or added to the current solution S. Otherwise, s is in free state and can be removed from or added to S (line 7). The refined search (RS) procedure is then invoked to perform a neighborhood

Algorithm 3: Modified Tabu Search procedure–MTS

```
Input: Initial solution $S$, candidate subset pool $M'$.
Output: Local best solution $S'$ of one individual obtained during the search.
1  $S' \leftarrow S, \hat{S} \leftarrow \emptyset, R \leftarrow 1$;
2  while $R <= 3$ and $\hat{S} = \emptyset$ do
3      $S \leftarrow S', TS \leftarrow \emptyset, iter \leftarrow 0, t \leftarrow 0$;
4      for each $s \in M'$ do
5          $tabu(s) \leftarrow -1$;
6      while $t < \epsilon$ do
7          $TS \leftarrow \{s \mid s \in M' \wedge tabu(s) <= iter\}$;
8          $S_2, \hat{S} \leftarrow RS(S, R, TS)$;
9          if $f(S_2) < f(S')$ then
10             $S' \leftarrow S_2, t \leftarrow 0, R \leftarrow 1$;
11         else
12             $t \leftarrow t + R$;
13             $R \leftarrow R + 1$;
14         for each $s \in (S \cup S_2) \setminus (S \cap S_2)$ do
15             $tabu(s) \leftarrow iter + \zeta$;
16         $S \leftarrow S_2, iter \leftarrow iter + R$;
17 return $S'$;
```

search on S under the current intensity R, obtaining the feasible solution as well as best solutions (line 8). The solution S_2, obtained by the refined search under the current R, represents the R-neighborhood of S. If the objective function of S_2 is better than S', then S' will be updated to S_2 and the counter t is reset to 0, R reset to 1 (lines 9–10). Otherwise, the counter t is incremented by R, and R will increase by 1 (lines 11–13). Before the current iteration stops, the algorithm increases the tabu value of all sets that have been removed from or added to S during this iteration by ζ (a parameter controlling the degree of tabu). This ensures that these sets will be in the tabu state for several subsequent iterations (lines 14–15). Afterwards, S is updated to S_2, $iter$ is increased by R (line 16). After each round of iteration, the algorithm checks if $t > \epsilon$ and $\hat{S}$ is empty. If this condition is true, it indicates that the algorithm has not found a better solution in more than ϵ iterations and should stop. Otherwise, it proceeds to the next round of iteration. Through the application of MTS, PSSC can effectively tackle the PSCP by leveraging the exploitation capabilities of this technique.

Now, we further give the details of the procedure RS in Algorithm 4, which serves as the searching engine of MTS. Given the input solution S, search intensity R, and tabu list TS, the RS algorithm initiates by setting S' as S, $\hat{S}$ as an empty set, and the operation count r to 1 (line 1). RS is then divided into two phases, that is, the removal phase and joining phase. In the removal phase, r sets are selected from S and removed from it. In each iteration, a non-tabu set $s \in S$ is eliminated, resulting in a new solution $S \setminus \{s\}$. This set of new solutions is referred to as S', which is then updated to the new solution with the best f value among all S' (lines 3–4). If S' represents a feasible solution, $\hat{S}$ is updated to S' (lines 5–6). In the joining phase, r sets are chosen from S and added back to it. In each iteration, a set $s \in S$ that is both non-tabu and not part

Algorithm 4: Refinement search–*RS*

```
Input: Initial solution $S$, search intensity $R$, tabu set $TS$, candidate subset pool $M'$.
Output: Best solution $S'$ obtained during the search, the accept solution $\hat{S}$.
1  $S' \leftarrow S, \hat{S} \leftarrow \emptyset, r \leftarrow 1$;
2  while $r <= R$ do
3      $C \leftarrow \{Sp \mid Sp \in S' \setminus \{s\} \wedge s \in M' \wedge s \notin TS\}$;        // Removal phase
4      $S' \leftarrow \operatorname{argmax}_{S' \in C} f(S')$;
5      if $S'.fit >= n$ then
6          $\hat{S} \leftarrow S'$;
7      $r \leftarrow r + 1$;
8  $r \leftarrow 1, U \leftarrow M' \setminus S'$;
9  while $r <= R$ do
10     $C \leftarrow \{Sp \mid Sp \in S' \cup \{s\} \wedge s \notin U \wedge s \notin TS\}$;        // Joining phase
11     $S' \leftarrow \operatorname{argmax}_{S' \in C} f(S')$;
12     if $\hat{S} \notin \emptyset$ then
13         $break$;
14     if $S'.fit >= n$ then
15         $\hat{S} \leftarrow S'$;
16     $r \leftarrow r + 1$;
17 return $S'$, $\hat{S}$;
```

of S' is added, generating a new solution $S \cup \{s\}$. The set of new solutions is denoted as S', which is updated to the new solution with the best f value (lines 10–11). If $\hat{S}$ is not an empty set (indicating the discovery of a feasible solution), the neighborhood search is terminated (lines 12–13). If S' represents a feasible solution, $\hat{S}$ is updated to S' (lines 14–15). Finally, the algorithm returns the best solution S' and the feasible solution $\hat{S}$ (line 17). Note that MTS aims to maximize the number of covered elements using a predefined value of k. It assesses the number of covered elements surpasses a specified threshold, represented by n. If this condition is met, the algorithm iteratively performs the same processes with a decreased value of $k = k - 1$. As a result, PSSC will yield a collection of k subsets, ensuring the coverage of at least n elements while simultaneously striving to minimize k.

3.4 The multiple path-relinking procedure

Path relinking is a powerful optimization technique employed to enhance the efficiency and effectiveness of solving complex optimization problems Glover (1997). It operates by combining solutions from different regions of the search space to construct improved solutions. Initially, the strategy selects both an initial and a target solution. Following this, it methodically examines the pathways linking these solutions, continuously honing and modifying the intermediate solutions encountered during this exploration. By leveraging the information contained in these paths, the path relinking strategy fosters the exchange of valuable traits and characteristics between solutions, leading to the discovery of high-quality solutions. This approach has proven particularly valuable in domains where finding optimal solutions is challenging, such as jobshop scheduling problems Peng et al. (2015), arc routing problems Usberti et al. (2013), maximum satisfiability problems Festa et al. (2005), graph coloring problems Lai et al. (2016), capacitated centered clustering problems Muritiba et al. (2022), etc.

Algorithm 5: Multiple path-relinking procedure–MPR

```
Input: Phase $mode$, current best solution $S^*$, local best solutions for each individual $P$, markers of
       population $Mark$.
Output: $mode$, $P$.
$P_t \leftarrow \emptyset$, $P_d \leftarrow \emptyset$;
switch $mode$ do
    case 0
        $P_d \leftarrow S^*$;
        $P_t \leftarrow roulette()$;
        while $P_t \in Mark$ do
            $P_t \leftarrow roulette()$ ;
        break;          // mode 1
    case 1
        $S \leftarrow \operatorname{argmax}_{P_d \in P} P_d.fit$;
        $P_t \leftarrow \operatorname{argmax}_{P_d \in P \setminus S} P_d.fit$;
        $P_d \leftarrow S^*$;
        break;          // mode 2
    case 2
        $P_d \leftarrow \operatorname{argmax}_{P_d \in P} P_d.fit$;
        $P_t \leftarrow \operatorname{argmin}_{P_d \in P} P_d.fit$;
        break;          // mode 3
$mode \leftarrow (mode + 1) \mod 3$;
$S \leftarrow P_t, U \leftarrow S \setminus P_d, U' \leftarrow P_d \setminus S$;
$l \leftarrow dist(S, P_d)$;
while $dist(S, P_d) > \alpha * l$ do
    $u \leftarrow \operatorname{argmax}_{u \in U} u$;
    $S \leftarrow S \setminus \{u\}, U \leftarrow U \setminus \{u\}$;
    $u' \leftarrow \operatorname{argmax}_{u' \in U'} u' \in U'$;
    $S \leftarrow S \cup \{u'\}, U' \leftarrow U' \setminus \{u'\}$;
    if $\left(\alpha * l < dist(S, P_d) < (1 - \alpha) * l\right) \wedge \left(f(S) > f(P_t)\right)$ then
        $P_t \leftarrow S$;
return $mode, P$ ;
```

The proposed strategy, known as the multiple path-relinking approach outlined in Algorithm 5, enhances the population's search capability by guiding it towards better neighborhoods and ensuring diversity among individuals. In the current stage $mode$, considering the current best solution S^*, the local best solutions of each individual $P = \{P_0, P_1, ..., P_{|P|}\}$, and the marker set $Mark$, we designate the index P_t to identify the individual in the population to be modified, and P_d to represent the guiding solution. To determine which solution to modify and the guidance solution, we alternate between different stages (lines 2–18). If $mode = 0$, we employ a roulette function to randomly select an unmarked individual to be moved towards the optimal solution S^* (lines 3–8). If $mode = 1$, during the search, we identify the indices of the two individuals with the best performance in P and move the second-best individual towards the optimal solution S^* (lines 9–13). If $mode = 2$, we locate the index P_d of the worst performing individual in the current cycle and the index P_t of the best performing individual (lines 14–17). Subsequently, the alternating selection of $mode$ is auto-increasing within module 3 (line 18). Once the solution P_t to be modified and the guidance solution P_d are determined, we direct P_t to search in the direction of P_d (line 19). The distance between two solutions, denoted as S and P_d, is computed by quantifying the number of distinct sets they have in common. This count is then stored in the variable l (line 20). During this process, the unique information of the guiding solution gradually assimilates into and is utilized by the current solution, resulting in multiple intermediate solutions. We eliminate inefficient information by removing the set with the highest f value in $P_t \backslash P_d$ from P_t, and integrate the set with the highest f value in $P_d \backslash P_t$ to approach the guidance solution. If the calculated distance falls within the range of $\alpha \times l$ to $(1 - \alpha) \times l$, where α is a parameter satisfying $0 < \alpha < 1$, the solution S will be selected and added to the collection of solutions P_t (lines 26–27). This procedure is repeated until the generated intermediate solution and the guidance solution exhibit minimal differences (lines 21–27). Finally, we return $mode$ and the newly generated population P (line 28).

3.5 The dynamic resource allocation strategy

The proposed dynamic resource allocation (DRA) strategy assesses the performance of each individual within the population based on its search frequency for the optimal solution during the last periodic search process. This performance metric drives the allocation of computational resources through a round-robin approach. Additionally, a subset of individuals with specific labels is excluded from resource allocation in the current cycle, effectively reducing the number of individuals engaged in simultaneous searching. However, to ensure fairness and prevent permanent exclusion, these labeled individuals are re-evaluated and potentially re-labeled at the end of each cycle. This strategy aims to optimize computational resource utilization and promote efficient individual search behaviors.

Algorithm 6 presents a comprehensive depiction of the dynamic resource allocation process, which determines the individual to suspend the search in subsequent iterations, allowing computational resources to be focused on individuals with promising potential. It is worth noting that the dynamic resource allocation (DRA) is **executed every** L iterations to provide an opportunity for underperforming individuals to enhance their performance and develop into promising candidates. This strategy is well-suited for facilitating early rapid convergence while also fostering diversity among individuals. To elaborate further, the current set of markers, denoted as $Mark$, is utilized to prevent the re-marking of individuals that were not explored in the previous L rounds.

Algorithm 6: Dynamic resource allocation procedure–DRA

```
Input: markers of population $Mark$, maximum ratio of the population $R_{mark}$, hit count set of the
       population $HC$.
Output: $Mark$.
1 $Mark' \leftarrow Mark$; $Mark \leftarrow \emptyset$; $k \leftarrow 0$;
2 while $k < R_{mark} \times |P|$ do
3     $d \leftarrow \operatorname{argmin}_{P_d \in U \setminus \{Mark' \cup Mark\}} HC_d$ ;
4     $Mark \leftarrow Mark \cup \{P_d\}$;
5     $k \leftarrow k + 1$;
6 return $Mark$;
```

Additionally, the hit count set, denoted as HC, keeps track of the number of times each individual has updated the global best solution. The first line of the DRA initializes certain parameters. Subsequently, the algorithm searches for the individual that has not discovered the global best solution within L iterations, as well as those that have not been marked in the $\{Mark' \cup Mark\}$ set (lines 2–5). Finally, the updated $Mark$ is returned to the main procedure, ensuring that computational resources are not allocated to individuals present in the $Mark$ set (line 6).

3.6 The fix-and-optimize procedure

The fix-and-optimize (FO) strategy is an efficient approach designed to tackle challenging optimization problems, especially those susceptible to getting trapped in local optima Toledo et al. (2015); Dorneles et al. (2014); Joncour et al. (2023). It leverages the combined power of fixing promising sets and employing a distinct search mode to effectively navigate the search landscape. Specifically, in PSSC, we fix selected sets in the current solution, then remove all unfixed sets and revert the solution to a previous state. Subsequently, we employ the MTS to optimize this infeasible solution, effectively restoring its feasibility. This creates a jump in the search space, allowing exploration of alternative paths.

The pseudo code is reported in Algorithm 7. Firstly, the algorithm initializes two counter variables $k = 0, t = 0$ (line 1) and marks all sets as unfixed (line 3). The loop iterates until k_{max} consecutive iterations are completed without any improvement in the solution (lines 2–17). Each loop involves a backtracking process where the algorithm removes γ sets with minimum score values from the unfixed pool ($S \setminus Fix$) (lines 4–7). Notably, certain promising sets within the candidate solution are fixed and remain untouched unless the best solution (S^*) undergoes an update. Following this removal, the remaining sets in S are marked as fixed (line 8) and cannot be removed again until they are explicitly unfixed. Afterwards, the algorithm attempts to repair the incomplete solution using a greedy heuristic, which involves iteratively adding the set with the highest $score$ value until the solution becomes feasible (lines 9–11). Following that, MTS is called again with the exclusion of any movement pertaining to the fixed sets (line 12). This procedure is seen as a dynamic optimization process, wherein new sets are incrementally added into the foundation of the previously established sets.

Algorithm 7: Fix-and-optimize procedure–FO

```
Input: Candidate solution $S$, current best solution $S^*$, backtrack step $\gamma$, maximum allowed
       iterations without improvement $k_{max}$.
Output: Generated solution $S$.
1  k ← 0;
2  while $k < k_{max}$ do
3      $t \leftarrow 0$; $Fix = \phi$;
4      while $t < \gamma$ do
5          $s' \leftarrow \operatorname{argmin}_{s \in S} score(s)$;
6          $S \leftarrow S \setminus \{s'\}$;
7          $t \leftarrow t + 1$;
8      $Fix \leftarrow S$; // Fix the sets left in $S$;
9      while $S$ is not feasible do
10         $s' \leftarrow \operatorname{argmax}_{s \in M \setminus S} score(s)$;
11         $S \leftarrow S \cup \{s'\}$;
12     $S \leftarrow MTS(S, M \setminus Fix)$;
13     if $f(S) < f(S^*)$ then
14         $S^* \leftarrow S$;
15         $k \leftarrow 0$;
16     else
17         $k \leftarrow k + 1$;
18 return $S$;
```

If the newly generated solution is better than the best solution found, the algorithm updates S^* with S and resets the counter k (lines 13–15). Otherwise, the counter simply increases by 1 (line 17). Finally, the algorithm returns the current solution S (line 18).

4 Experimental results and analysis

In this section, we delve into the detailed design of the experiment and its results. Firstly, we provide a description of the PSCP benchmarks utilized in the experiments. Subsequently, we present a comparison of PSSC with other state-of-the-art methods that address the PSCP. Lastly, we conduct ablation experiments to analyze the contribution of each core algorithmic strategy in PSSC. During our experiments, we execute CPLEX only once with a time limit of 7200 s as it is an exact solver, while each of the other heuristic methods undergoes ten independent runs with a time limit of 100 s for solving each instance. RWLS, AAS, PSSC are all implemented in C++. All experiments are carried out on an Intel(R) Xeon(R) CPU 6326 running at 2.90GHz with 187 GB RAM, operating under the CentOS 7.9 operating system. The specific benchmark instances, along with the code, are available on GitHub.[1]

To tune the parameters for PSSC, we employ the automatic configuration tool irace López-Ibáñez et al. (2016). We randomly select 60 instances from the available examples, and each instance is executed 10 times with a time limit of 100 s per run. The final

[1] https://github.com/LmR308/PSSC.

Table 1 Tuned PSSC parameters based on the irace tool

Parameters	Ranges	Final values
θ	$\{0.05, 0.1, 0.15, 0.2\}$	0.2
R_{mark}	$\{0.1, 0.2, 0.3, 0.4\}$	0.2
$\|P\|$	$\{10, 15, 20, 25\}$	15
ϵ	$\{100, 200, 300, 400\}$	200
ζ	$\{4, 6, 8, 10\}$	4
α	$\{0.2, 0.3, 0.4, 0.5, 0.6\}$	0.4
L	$\{3, 4, 5, 6, 7\}$	5
γ	$\{500, 750, 1000, 1250\}$	1000

parameter values obtained by irace are presented in Table 1. Furthermore, we conduct experiments on selected instances using all possible combinations of parameters.These experiments show that the optimal solution, obtained with the best parameter settings, significantly outperforms solutions from other parameter configurations. Also, PSSC shows considerable stability based on these parameters. As for the parameters used in RWLS and AAS, they have been tuned in their respective original publications Gao et al. (2014); Liu et al. (2022). So we did not retune them in this paper. To be specific, RWLS is a parameter-free method, which does not need to tune parameters. There are five parameters in AAS, namely the probability parameter P_{rcc} of RCC, the primitive cut-off condition $Iter_0$ of the local search iteration, the growth rate of the cut-off condition q, the initial value μ_0 of the Dynamic Disturbance mechanism and the decrement δ of it. The setting of these parameters is as follows: $P_{rcc} = 0.95$, $Iter_0 = 100$, $q = 10$, $\mu_0 = 0.5$, $\delta = 0.02$.

4.1 Benchmarks

The test instances used in the experiments are obtained from the OR-Library Beasley (1990), which is a well-known test suite utilized for the Set Covering Problem and its variants. For each instance, we performed two sets of experiments, with the value of n set to 90% and 95% of the total number of elements, respectively. Note that in previous work, only the $n90$ benchmark was tested Liu et al. (2022), whereas in this paper, an additional set of 75 difficult instances, namely $n95$, has been tested. It indicates that the experiments can provide a more comprehensive evaluation of the PSSC method by considering a broader range of challenging instances.

4.2 Comparison with other state-of-the-art methods

In this work, PSSC is compared with three state-of-the-art methods to solve PSCP, including a commercial solver CPLEX Hutter et al. (2010), a method for solving SCP called RWLS Gao et al. (2015), and the current best method for solving PSCP namely AAS Liu et al. (2022). All these competitors are introduced as follows:

CPLEX is a well-known commercial solver that is widely used for solving Mixed Integer Programming (MIP) problems. Within the context of PSCP, a sophisticated MIP-based constraint encoding method is implemented with CPLEX version 12.6.3.

RWLS is the most commonly used method for solving SCP problems, using two search operators to perturb candidate solutions and incorporating three main strategies into its local search process.

AAS is the current best algorithm for solving the PSCP. It utilizes a variant of the Ant Colony Optimization (ACO) algorithm Dorigo et al. (1996) and introduces efficient local search techniques such as Relaxed Configuration Checking (RCC) and a volatile fixed-weight mechanism to improve performance.

Tables 2 and 3 present the comparison results between PSSC and these state-of-the-art competitors on 150 benchmark instances, with $n = 90\%$ and $n = 95\%$ of the elements respectively. In each table, "Best" represents the best results obtained within 10 runs, "Avg" denotes the average results, and "Time" indicates the average runtime. The tables clearly demonstrate that PSSC achieves the best performance in terms of both best and average results across all instances. The “Mean” row underscores PSSC’s efficiency, with average runtimes of 10.26 for the $n = 90\%$ benchmark and 5.36 for the $n = 95\%$ benchmark, a notable reduction compared to other methods. Such computational efficiency is particularly advantageous in scenarios where solution speed is critical, emphasizing PSSC’s practical utility. Comparing the $n = 95\%$ benchmark with $n = 90\%$, it is evident that the former presents greater complexity. Table 3 reveals that, while both AAS and PSSC obtain optimal solutions in instances such as scp and scpclr, PSSC consistently achieves these solutions in less time, reflecting its effective search capabilities. Additionally, the "#Best" row reinforces PSSC’s consistency, as it reaches the highest count of optimal solutions, achieving 77 best results across both benchmarks. By contrast, AAS, the next strongest method, secures only 49 and 9 best results for the $n = 90\%$ and $n = 95\%$ cases, respectively. Moreover, PSSC demonstrates remarkable robustness in complex instances, such as the scpcyc series, where it outperforms other methods on both “Best” and “Avg” metrics. These results affirm PSSC’s advancement in addressing challenging cases within the partial set cover problem, consolidating its status as a highly effective and reliable solution method.

In addition, we conduct statistical tests to estimate the difference between PSSC and its competitors. Tables 4 and 5 report the p-values of Wilcoxon signed-rank tests for all pairwise comparisons.The p-values for all pairwise comparisons between PSSC and each of its competitors show that in these pairwise comparisons, the p-values for PSSC are less than 0.05. It indicates that the improvements of PSSC over other methods are statistically significant. The terms R^+_{best} and R^+_{avg} represent the number of instances in which the PSSC method notably excelled beyond other methodologies, specifically regarding the highest quality outcomes and the mean results, respectively. Conversely, R^-_{best} and R^-_{avg} are the opposite of R^+_{best} and R^+_{avg}, respectively. These indicators reflect the count of scenarios where the PSSC approach failed to outmatch the performance of alternative methods, both in terms of best results (R^-_{best}) and average results (R^-_{avg}). From the results, we can observe that PSSC has updated a total number of 67 new lower bounds of the benchmarks.

Table 2 Comparative results on benchmark n90

Instance	CPLEX		RWLS			AAS			PSSC		
	UB	Optimal time	Best	Avg	Time	Best	Avg	Time	Best	Avg	Time
scp41	**30**	216.64	37	37.5	51.25	**30**	**30**	1.06	**30**	**30**	0.27
scp42	**29**	47.98	34	35.2	59.39	**29**	**29**	0.02	**29**	**29**	0.29
scp43	**29**	64.34	35	35.8	59.17	**29**	**29.3**	19.89	**29**	**29**	0.49
scp44	**30**	401.49	37	38.3	67.82	**30**	**30.8**	0.23	**30**	**30**	0.78
scp45	**30**	189.87	35	36.7	59.83	**30**	**30**	0.16	**30**	**30**	0.42
scp46	**29**	40.71	36	36.8	61.30	30	30	0.13	**29**	**29**	0.52
scp47	**30**	61.62	37	37.8	63.42	31	31	0.18	**30**	**30**	0.56
scp48	**30**	1226.66	36	36.6	58.83	**30**	**30**	3.77	**30**	**30**	0.40
scp49	**30**	68.14	36	37	50.99	**30**	**30.4**	7.83	**30**	**30**	0.46
scp410	**30**	61.28	36	37.5	60.64	**31**	**31**	0.31	**30**	**30**	0.43
scp51	28	7200.24	35	36.4	57.25	**27**	**27.9**	0.02	**27**	**27**	0.91
scp52	**27**	2255.4	34	35.3	61.93	**27**	**27**	1.81	**27**	**27**	0.68
scp53	**27**	299.95	34	36.5	68.56	**27**	**27**	0.71	**27**	**27**	0.70
scp54	**27**	101.44	35	36	72.10	**27**	**27.9**	5.95	**27**	**27**	0.68
scp55	**27**	950.39	34	35.5	67.67	**27**	**27**	2.74	**27**	**27**	0.92
scp56	**27**	85.65	34	35.9	66.83	**27**	**27**	16.98	**27**	**27**	0.90
scp57	**27**	1880.42	34	35.7	69.97	**27**	**27**	0.43	**27**	**27**	0.69
scp58	**27**	336.27	34	36.9	73.46	**27**	**27.7**	3.23	**27**	**27**	1.16

Table 2 continued

Instance	CPLEX		RWLS			AAS			PSSC		
	UB	Optimal time	Best	Avg	Time	Best	Avg	Time	Best	Avg	Time
scp59	**28**	3819.3	36	37	54.58	**28**	**28**	4.36	**28**	**28**	0.80
scp510	**27**	2074.97	34	35.7	71.50	**27**	**27**	19.79	**27**	**27**	1.04
scp61	16	7200.27	19	19.5	0.05	**15**	**15**	21.37	**15**	**15**	0.60
scp62	**15**	642.62	17	18.2	0.03	**15**	**15**	0.00	**15**	**15**	0.31
scp63	**16**	7200.33	17	18.7	0.07	**16**	**16**	0.00	**16**	**16**	0.37
scp64	**16**	7200.25	19	19.7	0.03	**16**	**16**	0.00	**16**	**16**	0.33
scp65	**16**	6724.65	19	19.5	0.05	**16**	**16**	0.00	**16**	**16**	0.44
scpa1	31	7200.42	38	41.3	75.15	**30**	**30**	14.78	**30**	**30**	1.37
scpa2	**30**	7200.46	36	40.1	0.14	**30**	**30**	5.63	**30**	**30**	1.18
scpa3	31	7200.48	39	40.7	0.16	**30**	**30**	8.05	**30**	**30**	1.47
spca4	30	7200.44	38	40.1	81.00	**29**	**29**	0.72	**29**	**29**	1.22
scpa5	30	7200.42	39	40.4	0.12	**29**	**29**	0.91	**29**	**29**	1.59
scpb1	**16**	7200.46	19	19.3	0.18	**16**	**16**	0.01	**16**	**16**	1.47
scpb2	**16**	7200.42	19	19.2	0.14	**16**	**16**	0.01	**16**	**16**	1.44
scpb3	17	7200.45	19	19.5	0.12	**16**	**16**	0.10	**16**	**16**	1.59
scpb4	**16**	7200.43	19	19.6	0.26	**16**	**16**	8.01	**16**	**16**	1.90
scpb5	17	7200.42	19	19.4	0.13	**16**	**16**	0.02	**16**	**16**	2.03
scpc1	33	7200.43	43	44.1	0.15	33	33	0.21	**32**	**32**	**2.84**
scpc2	33	7200.39	42	43.4	0.09	33	33	0.05	**32**	**32**	**1.90**
scpc3	34	7200.45	41	43.1	0.32	33	33	0.75	**32**	**32**	**4.42**
scpc4	33	7200.42	41	42.8	0.25	**32**	**32**	16.67	**32**	**32**	2.15
scpc5	34	7200.4	41	44.1	0.09	**33**	**33**	0.23	**33**	**33**	1.77
scpd1	18	7200.51	20	21.2	0.36	18	18	0.00	**17**	**17**	**3.48**

Table 2 continued

Instance	CPLEX		RWLS			AAS			PSSC		
	UB	Optimal time	Best	Avg	Time	Best	Avg	Time	Best	Avg	Time
scpd2	**18**	7200.52	20	21.1	0.27	**18**	**18**	0.01	**18**	**18**	1.54
scpd3	**18**	7200.52	20	21.1	0.21	**18**	**18**	0.01	**18**	**18**	2.19
scpd4	**18**	7200.51	21	21.4	0.26	**18**	**18**	0.03	**18**	**18**	2.51
scpd5	**18**	7200.52	21	21.6	0.22	**18**	**18**	0.02	**18**	**18**	2.66
scpnre1	12	7200.96	12	12.4	0.76	**11**	**11**	0.52	**11**	**11**	8.93
scpnre2	12	7200.94	12	12.5	0.52	**11**	**11**	0.45	**11**	**11**	8.63
scpnre3	12	7200.93	12	12.4	0.59	**11**	**11**	0.54	**11**	**11**	8.58
scpnre4	12	7200.93	12	12.5	1.26	**11**	**11**	0.35	**11**	**11**	8.63
scpnre5	12	7200.95	12	12.7	0.49	**11**	**11**	1.14	**11**	**11**	9.14
scpnrf1	7	7201.21	7	7	0.07	**7**	**7**	0.00	**7**	**7**	12.57
scpnrf2	7	7201.45	7	7	0.11	**7**	**7**	0.00	**7**	**7**	12.56
scpnrf3	7	7201.44	7	7	0.09	**7**	**7**	0.00	**7**	**7**	15.11
scpnrf4	7	7201.47	7	7	0.07	**7**	**7**	0.00	**7**	**7**	13.59
scpnrf5	7	7201.51	7	7	0.08	**7**	**7**	0.00	**7**	**7**	13.95
scpnrg1	45	7201.53	53	54.4	1.14	44	44	0.45	**43**	**43**	**5.48**
scpnrg2	44	7201.6	53	54.5	1.22	44	44	0.36	**43**	**43**	**6.39**
scpnrg3	45	7201.57	54	54.5	0.97	44	44	0.30	**43**	**43**	**15.56**
scpnrg4	45	7201.55	54	54.8	0.94	44	44	0.48	**43**	**43**	**17.46**
scpnrg5	45	7201.62	53	54.6	0.96	44	44	0.58	**43**	**43**	**9.05**

Table 2 continued

Instance	CPLEX		RWLS			AAS			PSSC		
	UB	Optimal time	Best	Avg	Time	Best	Avg	Time	Best	Avg	Time
scpnrh1	23	7202.94	26	26.1	1.73	23	23	0.03	**22**	**22.2**	**41.64**
scpnrh2	24	7202.94	25	25.7	3.38	23	23	0.03	**22**	**22.4**	**27.43**
scpnrh3	23	7202.96	25	25.9	2.23	23	23	0.02	**22**	**22.4**	**31.25**
scpnrh4	23	7202.96	25	25.9	2.68	23	23	0.04	**22**	**22.4**	**27.00**
scpnrh5	23	7202.97	26	26	1.95	23	23	0.04	**22**	**22.3**	**30.50**
scpclr10	15	4.3	16	16.9	0.07	15	15	0.02	**14**	**14**	**0.05**
scpclr11	**15**	7200.26	16	16.0	0.12	**15**	**15**	0.00	**15**	**15**	0.27
scpclr12	**14**	7200.32	14	14.9	0.65	**14**	**14**	0.00	**14**	**14**	0.83
scpclr13	**12**	185.82	14	14.2	2.04	**12**	**12**	0.08	**12**	**12**	0.79
scpcyc06	50	7200.31	53	54.1	49.61	50	50	0.51	**48**	**48**	**0.32**
scpcyc07	121	7200.27	142	145.2	0.05	124	124.9	1.97	**118**	**118**	**83.22**
scpcyc08	298	7200.39	338	346.6	0.19	293	294.5	28.39	**281**	**281**	**19.18**
scpcyc09	713	7200.54	874	893.3	0.41	677	679.2	21.09	**652**	**652**	**86.00**
scpcyc10	1737	7201.19	1931	1962.3	1.85	1532	1536.9	33.64	**1505**	**1510.8**	**99.43**
scpcyc11	11264	7205.27	3433	3441.4	100.00	3412	3413.7	24.90	**3383**	**3383.4**	**99.96**
Mean	210.84	5282.57	116.12	118.05	21.23	102.67	102.87	3.77	101.08	101.19	10.26
#Best	33	–	0	0	–	49	49	–	75	75	–

Table 3 Comparative results on benchmark n95

Instance	CPLEX		RWLS			AAS			PSSC		
	UB	Optimal time	Best	Avg	Time	Best	Avg	Time	Best	Avg	Time
scp41	**33**	1004.29	42	44.2	70.62	37	37.7	0.48	**33**	**33**	0.11
scp42	**32**	74.86	41	43	76.30	35	35.5	0.42	**32**	**32**	0.02
scp43	**33**	219.15	42	43	72.15	36	36.8	3.87	**33**	**33**	0.02
scp44	**34**	7028.48	44	45.7	72.68	38	38.8	0.32	**34**	**34**	0.05
scp45	**33**	676.74	41	44.4	66.25	36	37.2	3.56	**33**	**33**	0.03
scp46	**33**	1754.16	42	43.2	80.63	36	36.9	1.68	**33**	**33**	0.03
scp47	**34**	2046.62	42	44.1	70.57	36	38.6	3.40	**34**	**34**	0.04
scp48	**33**	4210.62	42	43.3	65.55	36	36.7	1.37	**33**	**33**	0.03
scp49	**33**	1333.41	42	43.4	81.49	38	38.8	5.05	**33**	**33**	0.05
scp410	**34**	2350.17	43	44.8	73.57	36	37.4	5.81	**34**	**34**	0.03
scp51	**30**	7200.37	41	42.4	68.05	33	33.7	12.51	**30**	**30**	0.26
scp52	31	7200.34	42	43.1	65.55	32	33.4	2.38	**30**	**30**	0.07
scp53	**30**	7200.29	41	43.5	70.97	33	33.7	5.36	**30**	**30**	0.07
scp54	**30**	2402.31	42	43.3	62.65	33	33.6	12.38	**30**	**30**	0.05
scp55	**30**	7200.30	39	42.9	80.79	32	32.8	3.01	**30**	**30**	0.05

Table 3 continued

Instance	CPLEX		RWLS			AAS			PSSC		
	UB	Optimal time	Best	Avg	Time	Best	Avg	Time	Best	Avg	Time
scp56	**30**	7200.38	42	43.5	72.16	33	33.5	7.12	**30**	**30**	0.04
scp57	**30**	7200.32	41	42	74.80	32	32.8	12.77	**30**	**30**	0.08
scp58	31	7200.30	42	43.8	74.23	34	34.3	6.71	**30**	**30**	0.12
scp59	32	7200.29	42	44	71.84	32	33.5	3.64	**31**	**31**	0.04
scp510	**30**	7200.30	41	42.9	69.93	34	34.4	6.96	**30**	**30**	0.10
scp61	19	7200.28	22	24.5	42.54	18	18.8	7.01	**18**	**18**	0.02
scp62	18	7200.38	23	24.3	43.33	18	18.1	1.13	**17**	**17**	0.02
scp63	18	7200.36	23	24.7	49.69	18	18.6	2.91	**17**	**17**	0.04
scp64	19	7200.36	24	24.8	29.55	19	19.3	1.64	**18**	**18**	0.02
scp65	**18**	7200.38	24	24.9	33.85	19	19.3	9.03	**18**	**18**	0.03
scpa1	34	7200.43	47	49.1	83.56	36	37.6	2.09	**33**	**33**	0.30
scpa2	34	7200.42	46	49	73.50	37	37.2	6.63	**33**	**33**	0.45
scpa3	34	7200.48	48	49.8	80.48	37	37.6	13.72	**33**	**33**	0.14
spca4	33	7200.43	45	48.7	87.81	35	35.7	7.89	**32**	**32**	2.17
scpa5	34	7200.46	47	49.2	79.70	36	36.9	2.48	**33**	**33**	0.05
scpb1	**18**	7200.38	27	27.7	63.49	19	19.1	24.02	**18**	**18**	0.06

Table 3 continued

Instance	CPLEX		RWLS			AAS			PSSC		
	UB	Optimal time	Best	Avg	Time	Best	Avg	Time	Best	Avg	Time
scpb2	19	7200.43	27	27.8	42.62	19	19.8	0.72	**18**	**18**	0.09
scpb3	19	7200.43	26	27.7	38.82	19	19.9	1.27	**18**	**18**	0.66
scpb4	19	7200.43	27	28	50.60	19	19.9	0.62	**18**	**18**	0.28
scpb5	19	7200.43	27	27.8	50.10	19	19.5	11.87	**18**	**18**	0.16
scpc1	38	7200.42	53	57.4	85.59	40	40.9	20.94	**36**	**36**	2.69
scpc2	38	7200.44	52	56.9	87.66	40	41	15.64	**37**	**37**	0.08
scpc3	38	7200.43	54	57.6	81.48	40	40.7	11.51	**36**	**36**	0.49
scpc4	38	7200.37	55	57.4	89.53	40	40.5	4.11	**36**	**36**	0.40
scpc5	38	7200.36	56	57.6	79.91	40	41.44	2.73	**37**	**37**	0.12
scpd1	21	7200.46	29	30.5	43.27	21	21.8	1.76	**20**	**20**	0.18
scpd2	21	7200.56	30	30.7	57.37	22	22	0.84	**20**	**20**	0.13
scpd3	21	7200.47	28	30.1	67.47	22	22.1	0.21	**20**	**20**	0.13
scpd4	21	7200.51	30	30.5	47.34	22	22	2.87	**20**	**20**	0.13
scpd5	21	7200.50	29	30.5	45.35	22	22.1	8.32	**20**	**20**	0.56
scpnre1	14	7200.97	19	19	39.97	14	14	0.10	**13**	**13**	0.25
scpnre2	14	7200.95	18	18.8	43.45	14	14	0.29	**13**	**13**	0.28
scpnre3	14	7200.94	19	19.1	50.56	14	14	0.12	**13**	**13**	0.67
scpnre4	14	7200.98	19	19	39.52	14	14	0.11	**13**	**13**	0.34
scpnre5	14	7200.92	19	19.2	46.44	14	14	0.12	**13**	**13**	0.45

Table 3 continued

Instance	CPLEX		RWLS			AAS			PSSC		
	UB	Optimal time	Best	Avg	Time	Best	Avg	Time	Best	Avg	Time
scpnrf1	10	7201.42	10	10.1	37.56	**8**	**8**	0.16	**8**	**8**	0.29
scpnrf2	10	7201.44	10	10.1	43.43	**8**	**8**	0.11	**8**	**8**	0.27
scpnrf3	**8**	7201.45	10	10.6	12.66	**8**	**8**	0.13	**8**	**8**	0.12
scpnrf4	9	7201.50	10	10.7	16.02	**8**	**8**	0.13	**8**	**8**	0.17
scpnrf5	**8**	7201.43	10	10.1	42.46	**8**	**8**	0.14	**8**	**8**	0.13
scpnrg1	51	7201.61	83	86.7	90.10	55	55.4	11.59	**49**	**49**	8.42
scpnrg2	52	7201.59	83	86.8	85.52	55	55.5	9.58	**49**	**49**	49.23
scpnrg3	52	7201.66	83	86.2	92.25	55	55.7	9.01	**49**	**49**	22.40
scpnrg4	52	7201.65	84	86	86.46	54	55.6	14.11	**50**	**50**	4.85
scpnrg5	52	7201.63	84	87.2	86.84	55	56	12.91	**50**	**50**	0.86
scpnrh1	28	7202.89	38	39	71.06	28	28.1	13.77	**26**	**26**	1.07
scpnrh2	27	7203.00	38	38.9	63.60	28	28.3	14.86	**26**	**26**	3.87
scpnrh3	27	7202.96	38	38.7	70.27	28	28.2	16.37	**26**	**26**	1.04
scpnrh4	27	7203.06	37	38.7	57.67	28	28	18.64	**26**	**26**	9.34
scpnrh5	27	7202.95	38	38.8	64.58	28	28.4	19.03	**26**	**26**	1.26

Table 3 continued

Instance	CPLEX		RWLS			AAS			PSSC		
	UB	Optimal time	Best	Avg	Time	Best	Avg	Time	Best	Avg	Time
scpclr10	**19**	2124.16	22	22.3	30.65	**19**	**19.9**	6.74	**19**	**19**	0.01
scpclr11	**18**	7200.41	21	21.6	32.51	**18**	**18.8**	4.88	**18**	**18**	0.02
scpclr12	**17**	7200.44	21	21.3	54.84	**17**	**17.3**	6.19	**17**	**17**	0.04
scpclr13	**15**	2876.04	20	25.7	78.04	**15**	**15**	0.01	**15**	**15**	0.03
scpcyc06	**55**	7200.31	60	61.1	39.64	58	58.8	14.81	**55**	**55**	0.01
scpcyc07	134	7200.39	137	137	0.29	135	135.8	0.09	**129**	**129**	6.69
scpcyc08	315	7200.38	334	334.9	3.34	330	332.8	0.79	**312**	**312**	13.21
scpcyc09	778	7200.59	754	761.2	38.99	739	739	0.17	**711**	**711**	70.24
scpcyc10	1878	7201.09	1719	1725	100.00	1768	1776.5	83.46	**1628**	**1628.8**	95.48
scpcyc11	11264	7205.13	3895	3904.8	100.00	3833	3834.4	11.02	**3818**	**3819.6**	99.94
Mean	217.19	6327.44	126.21	128.08	61.25	117.67	118.39	7.07	112.81	112.85	5.36
#Best	26	–	0	0	–	9	9	–	75	75	–

Table 4 Wilcoxon signed ranks test results of PSSC and three compared methods on 75 n90 instances, with a significance level of 0.05

Comparison	R^{+}_{best}	R^{-}_{best}	p-value	R^{+}_{avg}	R^{-}_{avg}	p-value
vs CPLEX	36	0	4.03e−17	36	0	4.03e−17
vs RWLS	69	0	1.16e−11	70	0	8.63e−11
vs AAS	24	0	9.6e−19	30	0	9.65e−19

Table 5 Wilcoxon signed ranks test results of PSSC and three compared methods on 75 n95 instances, with a significance level of 0.05

Comparison	R^{+}_{best}	R^{-}_{best}	p-value	R^{+}_{avg}	R^{-}_{avg}	p-value
vs CPLEX	49	0	1.18e−17	49	0	1.18e−17
vs RWLS	75	0	2.41e−13	75	0	8.80e−14
vs AAS	65	0	1.70e−16	69	0	1.70e−16

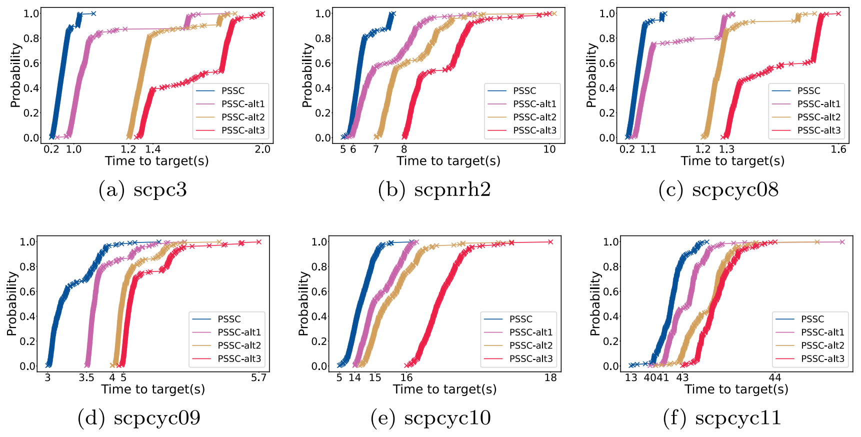


Fig. 3 Cumulative probability distribution plot of different alternative versions of PSSC on different instances

4.3 Effects of core algorithmic strategies

In PSSC, there are four core components: modified tabu search, multiple path-relinking, fix-and-optimize strategy, and dynamic resource allocation. Since the search is driven by MTS, we retain it and assess the impact of remaining core algorithmic technique through an ablation study. In this study, we created three alternative versions of PSSC by omitting each strategy individually. These alternative versions are referred to as *PSSC-alt1*, *PSSC-alt2*, and *PSSC-alt3*, respectively.

In this ablation study, we not only consider the algorithm's convergence degree but also focus on its efficiency. Thus, we employ Time-to-target (TTT) plots as a means to visualize the algorithm's running time. TTT plots depict the algorithm's probability on the ordinate axis and the solutions that meet or exceed a given target value within a specified running time on the abscissa axis Aiex et al. (2007). For each instance, the running time is sorted in ascending order. We assign the i-th sorted running time t_i to the probability $p_i = (i - 1/2)/n$ and plot the points $z_i = [t_i, p_i]$ for $i = 1, 2, ..., n$. Figure 3 illustrates the cumulative probability distribution of varying strategies on diverse instances. To provide a comprehensive illustration of the experiment results, we present TTT plots for six instances, namely scpc3, scpnrh2, scpcyc08, scpcyc09, scpcyc10, scpcyc11. Each instance was executed 100 times to ensure robustness and accuracy in our analysis. It is evident that PSSC consistently attains the best solution in the briefest duration across all instances. Although other strategies occasionally achieve best solutions, they generally necessitate more time compared to PSSC. This further underscores the efficacy of the core algorithmic techniques employed in PSSC.

5 Conclusion and future work

In this paper, we have presented a highly efficient algorithm, known as PSSC, designed specifically for addressing the PSCP. We have introduced a novel framework for local improvement, aimed at iteratively identifying a thinning k for MSKCP capable of covering a minimum of n elements. Further, we have proposed intra-population strategies to effectively handle exploitation, encompassing the modified tabu search and fix-and-optimize techniques. In terms of exploration, we have introduced an inter-population approach called the multiple path-relinking strategy. Furthermore, in order to enhance computational efficiency, we have integrated a dynamic resource allocation strategy. By combining these powerful strategies, PSSC has emerged as a successful solver for PSCP.

The effectiveness and efficiency of PSSC have been extensively evaluated on a diverse set of existing and newly generated instances. Our experimental findings demonstrate that PSSC exhibits exceptional performance, surpassing the state-of-the-art algorithms in terms of the quality of solutions obtained. Notably, PSSC has improved upon 67 best-known solutions. It is worth highlighting that PSSC demonstrates a notable advantage when applied to large-scale instances, making it well-suited for real-world scenarios.

As for future work, it would be beneficial to reduce the number of parameters or explore adaptive approaches for parameter value selection. Additionally, it would be interesting to adapt some of the innovative ideas employed in PSSC to tackle other challenging combinatorial optimization problems.

Funding This work was supported in part by the Natural Science Research Foundation of Jilin Province of China under Grant No. YDZJ202201ZYTS415, NSFC under Grant No.(62206045, 62106040, 62306149), Jilin Province Philosophy and social science planning project/major project under Grant No.2023ZD15, the Fundamental Research Funds for the Central Universities under Grant No.(2412022QD041, 2412022ZD016, 2412022QD040), Jilin Education Department Project under Grant No.(JJKH20231319KJ, JJKH202313 20KJ, JJKH20241430KJ), Jilin Science and Technology Association under Grant No. QT202320, National Social Science Foundation in 2024 under Grant No.24BMZ101.

Data Availability Data is available at https://github.com/LmR308/PSSC

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This material has not been published in whole or in part elsewhere. The manuscript is not currently being considered for publication in another journal. All authors have been personally and actively involved in substantive work leading to the manuscript, and will hold themselves jointly and individually responsible for its content.

Informed consent Informed consent was obtained from all individual participants included in the study.

References

Aiex, R.M., Resende, M.G., Ribeiro, C.C.: Ttt plots: a perl program to create time-to-target plots. Optim. Lett. **1**, 355–366 (2007)

Alfieri, A., Kroon, L., Van de Velde, S.: Personnel scheduling in a complex logistic system: a railway application case. J. Intell. Manuf. **18**, 223–232 (2007)

Babalola, A.E., Ojokoh, B.A., Odili, J.B.: A review of population-based optimization algorithms. In: 2020 International Conference in Mathematics, pp. 1–7. IEEE, Computer Engineering and Computer Science (ICMCECS) (2020)

Balas, E., Carrera, M.C.: A dynamic subgradient-based branch-and-bound procedure for set covering. Oper. Res. **44**(6), 875–890 (1996)

Bansal, N., Caprara, A., Sviridenko, M.: A new approximation method for set covering problems, with applications to multidimensional bin packing. SIAM J. Comput. **39**(4), 1256–1278 (2010)

Beasley, J.E.: Or-library: distributing test problems by electronic mail. J. Operat. Res. Soc. **41**(11), 1069–1072 (1990)

Beasley, J.E., Chu, P.C.: A genetic algorithm for the set covering problem. Eur. J. Oper. Res. **94**(2), 392–404 (1996)

Beasley, J.E., Jörnsten, K.: Enhancing an algorithm for set covering problems. Eur. J. Oper. Res. **58**(2), 293–300 (1992)

Benhaya, K., Hocine, R., Bendib, S.S.: Ga-based approaches for optimization energy and coverage in wireless sensor network: State of the art. In: International Conference on Artificial Intelligence and its Applications, Springer, pp 346–355 (2021)

Bilal, N., Galinier, P., Guibault, F.: An iterated-tabu-search heuristic for a variant of the partial set covering problem. J. Heuristics **20**(2), 143–164 (2014)

Boschetti, M., Maniezzo, V.: A set covering based matheuristic for a real-world city logistics problem. Int. Trans. Oper. Res. **22**(1), 169–195 (2015)

Brusco, M.J., Jacobs, L.W., Thompson, G.M.: A morphing procedure to supplement a simulated annealing heuristic for cost-andcoverage-correlated set-covering problems. Ann. Oper. Res. **86**, 611–627 (1999)

Caprara, A., Toth, P., Fischetti, M.: Algorithms for the set covering problem. Ann. Oper. Res. **98**(1), 353–371 (2000)

Chaurasia, S.N., Kim, J.H.: An evolutionary algorithm based hyper-heuristic framework for the set packing problem. Inf. Sci. **505**, 1–31 (2019)

Chekuri, C., Quanrud, K., Zhang, Z.: On approximating partial set cover and generalizations. arXiv preprint arXiv:1907.04413 (2019)

Crawford, B., Soto, R., Cuesta, R., Paredes, F.: Application of the artificial bee colony algorithm for solving the set covering problem. Sci. World J. **1**, 189164 (2014)

Daskin, M.S., Owen, S.H.: Two new location covering problems: The partial p-center problem and the partial set covering problem. Geogr. Anal. **31**(3), 217–235 (1999)

Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. IEEE Trans. Syst. Man Cybernetics part b cybernetics **26**(1), 29–41 (1996)

Dorneles, Á.P., De Araújo, O.C., Buriol, L.S.: A fix-and-optimize heuristic for the high school timetabling problem. Comput. Oper. Res. **52**, 29–38 (2014)

Emerick, B., Song, M.S., Lu, Y., Vasko, F.: An application of machine learning tools to predict the number of solutions for a minimum cardinality set covering problem. In: International Conference on Optimization and Learning, Springer, pp 175–185 (2023)

Festa, P., Pardalos, P.M., Pitsoulis, L.S., Resende, M.G.: Grasp with path-relinking for the weighted maximum satisfiability problem. In: Experimental and Efficient Algorithms: 4th International Workshop, WEA 2005, Santorini Island, Greece, May 10-13, 2005. Proceedings 4, Springer, pp 367–379 (2005)

Gao, C., Weise, T., Li, J.: A weighting-based local search heuristic algorithm for the set covering problem. In: 2014 IEEE Congress on Evolutionary Computation (CEC), IEEE, pp 826–831 (2014)

Gao, C., Yao, X., Weise, T., Li, J.: An efficient local search heuristic with row weighting for the unicost set covering problem. Eur. J. Oper. Res. **246**(3), 750–761 (2015)

Glover, F.: A template for scatter search and path relinking. In: European Conference on Artificial Evolution, Springer, pp 1–51 (1997)

Hartmanis, J.: Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson). Siam Review 24(1):90 (1982)

Hochbaum, D.S.: Approximation algorithms for the set covering and vertex cover problems. SIAM J. Comput. **11**(3), 555–556 (1982)

Hutter, F., Hoos, H.H., Leyton-Brown, K.: Automated configuration of mixed integer programming solvers. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 7th International Conference, CPAIOR 2010, Bologna, Italy, June 14-18, 2010. Proceedings 7, Springer, pp 186–202 (2010)

Inamdar, T., Varadarajan, K.: On partial covering for geometric set systems. In: 34th International Symposium on Computational Geometry (SoCG 2018), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2018)

Joncour, C., Kritter, J., Michel, S., Schepler, X.: Generalized relax-and-fix heuristic. Comput. Oper. Res. **149**, 106038 (2023)

Kempe, D., Kleinberg, J., Tardos, É.: Maximizing the spread of influence through a social network. In: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, pp 137–146 (2003)

Kritter, J., Brévilliers, M., Lepagnot, J., Idoumghar, L.: On the optimal placement of cameras for surveillance and the underlying set cover problem. Appl. Soft Comput. **74**, 133–153 (2019)

Lai, X., Hao, J.K., Lü, Z., Glover, F.: A learning-based path relinking algorithm for the bandwidth coloring problem. Eng. Appl. Artif. Intell. **52**, 81–91 (2016)

Leutwiler, F., Corman, F.: Set covering heuristics in a benders decomposition for railway timetabling. Comput. Oper. Res. **159**, 106339 (2023)

Liao, C.C., Ting, C.K.: A novel integer-coded memetic algorithm for the set k-cover problem in wireless sensor networks. IEEE Trans. Cybernetics **48**(8), 2245–2258 (2017)

Liu, X., Zhou, Y., Yin, M., Lv, S.: An argentine ant system algorithm for partial set covering problem. Data Technol. Appl. **56**(5), 762–781 (2022)

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Oper. Res. Perspect. **3**, 43–58 (2016)

Luo, C., Xing, W., Cai, S., Hu, C.: Nusc: an effective local search algorithm for solving the set covering problem. IEEE Trans. Cybernetics **54**(3), 1403–1416 (2022)

Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**(11), 1097–1100 (1997)

Muritiba, A.E.F., Gomes, M.J.N., de Souza, M.F., Oria, H.L.G.: Path-relinking with tabu search for the capacitated centered clustering problem. Expert Syst. Appl. **198**, 116766 (2022)

Peng, B., Lü, Z., Cheng, T.C.E.: A tabu search/path relinking algorithm to solve the job shop scheduling problem. Comput. Oper. Res. **53**, 154–164 (2015)

Prügel-Bennett, A.: Benefits of a population: Five mechanisms that advantage population-based algorithms. IEEE Trans. Evol. Comput. **14**(4), 500–517 (2010)

Ran, Y., Zhang, Y., Zhang, Z.: Parallel approximation for partial set cover. Appl. Math. Comput. **408**, 126358 (2021)

Toledo, C.F.M., da Silva, Arantes M., Hossomi, M.Y.B., França, P.M., Akartunalı, K.: A relax-and-fix with fix-and-optimize heuristic applied to multi-level lot-sizing problems. J. Heuristics **21**, 687–717 (2015)

Usberti, F.L., França, P.M., França, A.L.M.: Grasp with evolutionary path-relinking for the capacitated arc routing problem. Comput. Oper. Res. **40**(12), 3206–3217 (2013)

Wang, L., Singh, C.: Population-based intelligent search in reliability evaluation of generation systems with wind power penetration. IEEE Trans. Power Syst. **23**(3), 1336–1345 (2008)

Wang, Y., Pan, S., Al-Shihabi, S., Zhou, J., Yang, N., Yin, M.: An improved configuration checking-based algorithm for the unicost set covering problem. Eur. J. Oper. Res. **294**(2), 476–491 (2021)

Yaghini, M., Karimi, M., Rahbar, M.: A set covering approach for multi-depot train driver scheduling. J. Comb. Optim. **29**, 636–654 (2015)

Zhou, Y., Liu, X., Hu, S., Wang, Y., Yin, M.: Combining max-min ant system with effective local search for solving the maximum set k-covering problem. Knowl.-Based Syst. **239**, 108000 (2022)

Zhou, Y., Fan, M., Liu, X., Xu, X., Wang, Y., Yin, M.: A master-apprentice evolutionary algorithm for maximum weighted set k-covering problem. Appl. Intell. **53**(2), 1912–1944 (2023)