SORT 50 (1) January-June 2026, 1-42 DOI:

Scatter Search: Foundations and Implementations

Manuel Laguna[1], Sergio Cavero[2] and Rafael Martí[3]

Abstract

Scatter search is a population-based metaheuristic designed to solve complex optimization problems through structured solution combination and adaptive memory. Unlike traditional evolutionary algorithms, scatter search emphasizes deterministic strategies to balance intensification and diversification. We present a comprehensive review of scatter search and its connection to path relinking, covering their historical development, core methodology, and applications. Key components of scatter search include diversification generation, improvement, reference set updating, subset generation, and solution combination. Advanced strategies such as dynamic reference set updating, tiered memory structures, constructive and destructive neighborhoods, and vocabulary building enhance its performance and scalability. Scatter search has been successfully applied in scheduling, routing, bioinformatics, and software engineering. Hybridizations with other metaheuristics and integration with machine learning further expand its applicability. The review concludes with a tutorial on a scatter search Python implementation for 0-1 knapsack problems that includes a Jupyter Notebook with code, execution traces, visualizations, and didactic analyses.

MSC: 97D50, 68T20, 90C27, 90C59, 90-08, 68W50.

Keywords: Scatter Search, Path Relinking, Metaheuristics, Optimization.

1. Introduction

Scatter search (SS) and path relinking (PR) are population-based metaheuristics that differ from traditional evolutionary algorithms by emphasizing strategic choices and the systematic use of adaptive memory. Unlike genetic algorithms, SS and PR give pre-

[1] Leeds School of Business, University of Colorado Boulder, USA. laguna@colorado.edu

[2] Departamento de Informática y Estadística, Universidad Rey Juan Carlos, Spain. sergio.cavero@urjc.es

[3] Departamento de Estadística e Investigación Operativa, Universitat de València, Spain. rafael.marti@uv.es
Received: January 2026

ference to strategy and context-information over randomization, offering a robust framework for tackling a wide range of optimization problems that emerge in practice.

The name scatter search may seem to indicate a lack of focus and absence of mechanisms to zeroing in on promising areas of the solution space. In contrast to genetic algorithms (GAs), whose original design did not include explicit forms of search intensification, the SS methodology recognizes the need for localized searches. Therefore, the word "scatter" in the name of the methodology refers to the goal of identifying structurally different points in the solution space from which to initiate a localized exploration, also known as exploitation. Exploitation is achieved within the SS framework via the so-called intensification method. This involves making small changes, based on local knowledge, to refine a solution. The goal of small improvements is to obtain the best outcome from a specific region of the solution space. True exploration, on the other hand, is the strategy of "discovering" new information and trying out new, potentially better, but unknown options to build a solution. Exploration means leaving the current area to search a new, possibly disconnected part of the solution space.

Striking a balance between exploration and exploitation is at the core of most metaheuristics for optimization. SS addresses this balance in a very direct way by including a diversification method for exploration and an intensification method for exploitation. The combination method in SS is another form of exploitation because it creates solutions based on elements present in the so-called reference solutions. Advanced versions of scatter search include path relinking as a combination method.

Path relinking generates new solutions by tracing paths between selected reference solutions. PR focuses on exploring the "trajectories" or paths that connect reference solutions in the neighborhood space. Essentially, path relinking extends th e solution-combining principles of scatter search by actively exploring the paths between them to find better solutions. This involves moving from an initial solution to a guiding solution by making a series of moves to find potentially better solutions within that path. Its core function is to intensify the search around promising areas. When combined with the exploration mechanisms of scatter search the result is a balance between PR's focus on exploitation with a broader, scattered exploration of the solution space.

2. Historical Background

Scatter search was first introduced by Glover in 1977 as a heuristic to solve integer programming problems, particularly through the use of surrogate constraint relaxation (Glover, 1977). The method was conceived as an extension of mathematical relaxation techniques, aiming to generate new information by combining existing constraints and solutions in a structured manner. This foundational idea, strategic combination of elements to uncover latent information, remains central to SS today.

Although SS was proposed in the 1970s, it did not gain significant attraction until the early 1990s, when it was revisited at the EPFL (*École Polytechnique Fédérale de Lausanne*) Seminar on Operations Research and Artificial Intelligence Search Methods.

The renewed interest led to a formal publication in 1994, expanding the scope of SS to include nonlinear, binary, and permutation-based optimization problems (Glover, 1994).

The algorithmic structure of SS was later formalized in a *scatter search template* (Glover, 1997), which provided a simplified, yet powerful, framework for its implementation. This template became the reference point for most subsequent SS applications and inspired the research community to solve a wide variety of challenging optimization problems.

In parallel, path relinking emerged as a generalization of a neighborhood search within the Tabu Search framework. Initially proposed by Glover and Laguna in the early 1990s Glover (1997), PR introduced the concept of generating trajectories in the neighborhood space between elite solutions (Glover and Martí, 2000). This approach enhances intensification and diversification strategies and was later integrated into SS as a combination method.

Two major milestones further consolidated the role of SS in the metaheuristic landscape. First, the publication of the book *Scatter Search: Methodology and Implementations in C* by Laguna and Martí Laguna and Martí (2003) provided practical tools and implementation details for researchers and practitioners. Second, a special issue of the *European Journal of Operational Research* in 2006 showed successful applications of SS across various domains (Martí, 2006).

In the 2010s, SS was recognized as a robust and flexible metaheuristic, particularly well-suited for solving NP-hard combinatorial optimization problems, both single-objective and multiobjective. Its relevance was acknowledged in chapters of well-known reference works such as the *Handbook of Heuristics* (Martí, Corberán and Peiró, 2018; Martí, Laguardia and León, 2025) and the *Handbook of Metaheuristics* (Glover and Marti, 2003; Resende and Martí, 2010).

Over the last decade, SS research has focused on methodological enhancements and the development of new variants. These efforts include specialized strategies aimed at improving individual phases of the method, as well as broader advances such as parallelization frameworks. Notable contributions include the introduction of a parallel SS framework optimized for modern computing architectures (Casado and Laguna, 2025) and a comprehensive survey dedicated exclusively to SS, which synthesizes its evolution, theoretical underpinnings, and diverse applications (Kalra and Shah, 2021).

Given its maturity and well-established performance, it is now possible to define a chronological axis of SS development, highlighting the most relevant milestones. This timeline, illustrated in Fig. 1, can be structured into three periods: *Foundation* (1977–2003), *Development* (2003–2015), and *Recent Advances* (2015–2025).

Before delving into specific methodological aspects, we first examine the overall publication trends on SS since its inception. To this end, we retrieved bibliographic data from the Web of Science Core Collection[1]. The search strategy was designed to identify contributions that explicitly mention “Scatter Search” or its variations (such as “Scatter

[1] https://www.webofscience.com/wos/woscc/summary/098cc43f-b76d-46be-9040-cac95ba5cd6b-01888557c0/relevance/1

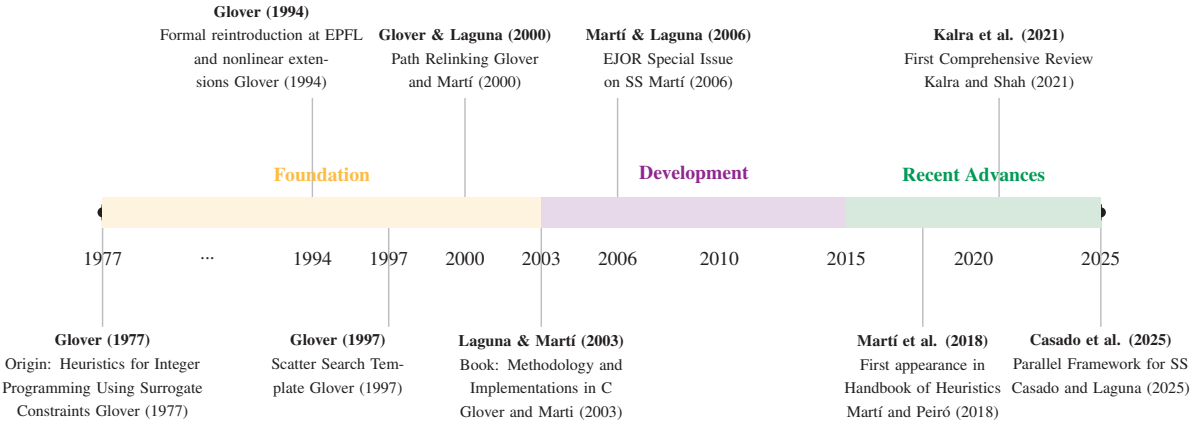

Figure 1. *SS timeline in three periods: Foundation, Development, and Recent Advances.*

Search Algorithm" or "Scatter Search Method") within the topic or all fields, ensuring the inclusion of works where this methodology plays a significant role.

The results reveal that the SS methodology has maintained a sustained presence in the optimization literature, not necessarily through direct application, but as a conceptual reference or methodological influence. Its appearance in related publications, whether cited for foundational principles, comparative analysis, or theoretical basis, indicates continued relevance and intellectual impact. This trend is illustrated in Figure 2, which shows a robust stream of articles that reference SS, underscoring its role as a recognized component in the broader metaheuristic landscape.

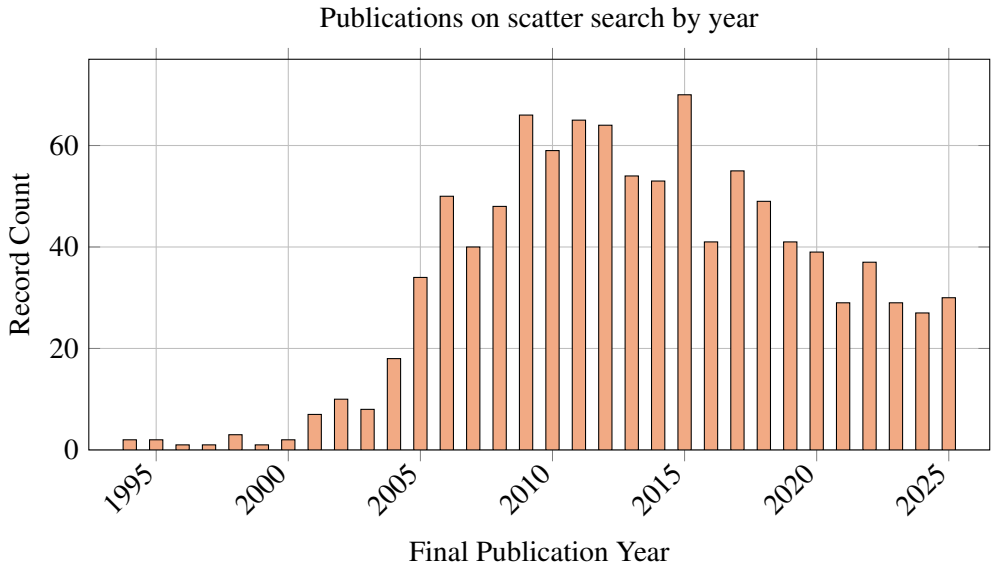

Figure 2. *Annual publication counts for scatter search–related works, based on Web of Science data.*

From a venue perspective, the distribution of publications across journals and conference proceedings is highly concentrated within high-impact, application-oriented outlets. Figure 3 shows that the European Journal of Operational Research and Computers & Operations Research lead the list, jointly accounting for over 10% of the total output. Notably, the prominence of venues such as Expert Systems with Applications, Applied Soft Computing, and various manufacturing and engineering journals underscores the heavily applied nature of scatter search. Rather than remaining a purely theoretical construct, SS is evidently favored as a practical solver for complex real-world problems in industrial engineering and management science.

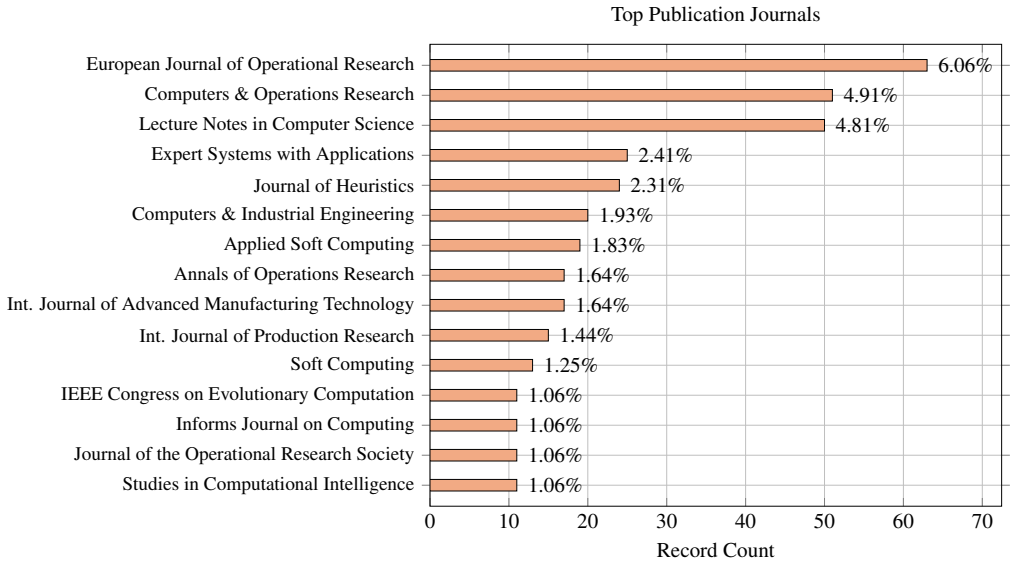

Figure 3. *Distribution of scatter search publications across the top 15 publication journals.*

Finally, regarding the subject categorization presented in Figure 4, the Web of Science classification reveals the methodology's strong dual nature. The field is dominated by two primary pillars: *Operations Research & Management Science* (31.28%) and *Computer Science: Artificial Intelligence* (29.74%), which appear in nearly equal measure. Beyond this core intersection, the significant presence of interdisciplinary applications and various engineering subfields (electrical, industrial, and manufacturing) reinforces the broad applicability of SS. The taxonomy illustrates SS as a transversal optimization tool, bridging the gap between theoretical computer science and practical engineering solutions.

3. Methodology

Scatter search and path relinking are population-based metaheuristics that differ fundamentally from other evolutionary methods driven primarily by random variation. Their

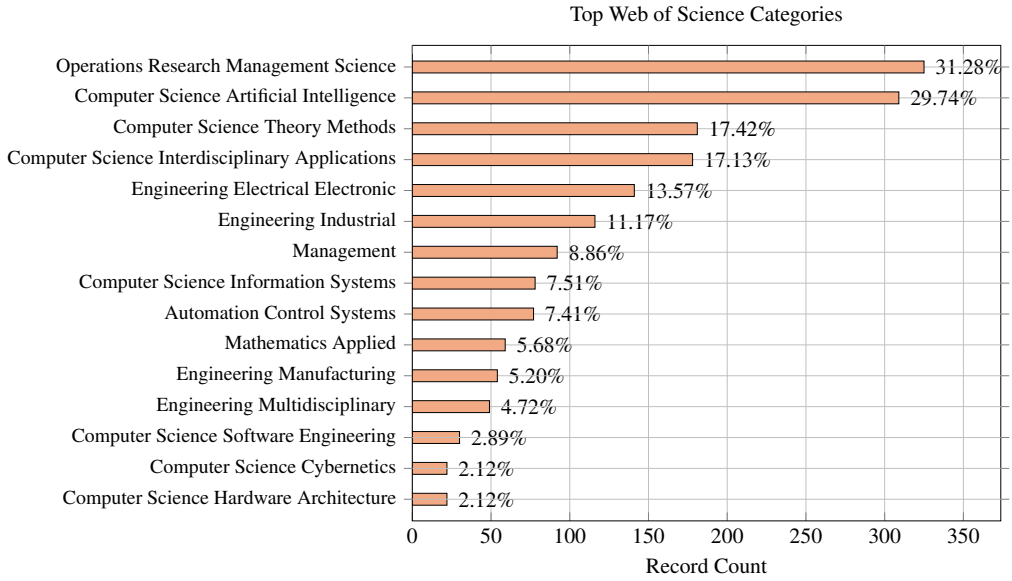

Figure 4. *Distribution of scatter search publications across the top 15 Web of Science Categories.*

design is rooted in the systematic combination and improvement of solutions, guided by strategic memory structures. Rather than relying on stochastic operators to maintain diversity, these approaches employ deterministic mechanisms and controlled probabilistic choices to generate and select candidate solutions, aiming to strike a deliberate balance between intensification (searching around high-quality solutions) and diversification (exploring new regions of the solution space).

Figure 5 illustrates a general framework of scatter search. The process starts with a Diversification Generation Method to create an initial population, P. An Improvement Method is then applied to enhance these initial solutions. The core of the algorithm is the Reference Set Update Method, which constructs and maintains the *RefSet* by selecting a small set of elite solutions. As depicted in the figure, this selection process is driven by two distinct criteria: solution quality, represented by color intensity (where darker shades indicate higher quality), and diversity, represented by different geometric shapes (squares, triangles, rhombuses, etc.). The algorithm then enters an iterative cycle where a Subset Generation Method selects solutions from the *RefSet* to be combined by a Solution Combination Method, creating new candidates. These new solutions are improved and may be used to update the *RefSet*, repeating the cycle until a stopping condition is met.

Algorithm 1 shows the process represented in Figure 5 in a sequential way, following the basic template of SS. The process begins in Step 1 with the generation of an initial population using the Diversification Generation Method, followed in Step 2 by the application of the Improvement Method to refine each solution. The initial Reference Set is constructed in Step 3 by selecting solutions that are both diverse and with high quality.

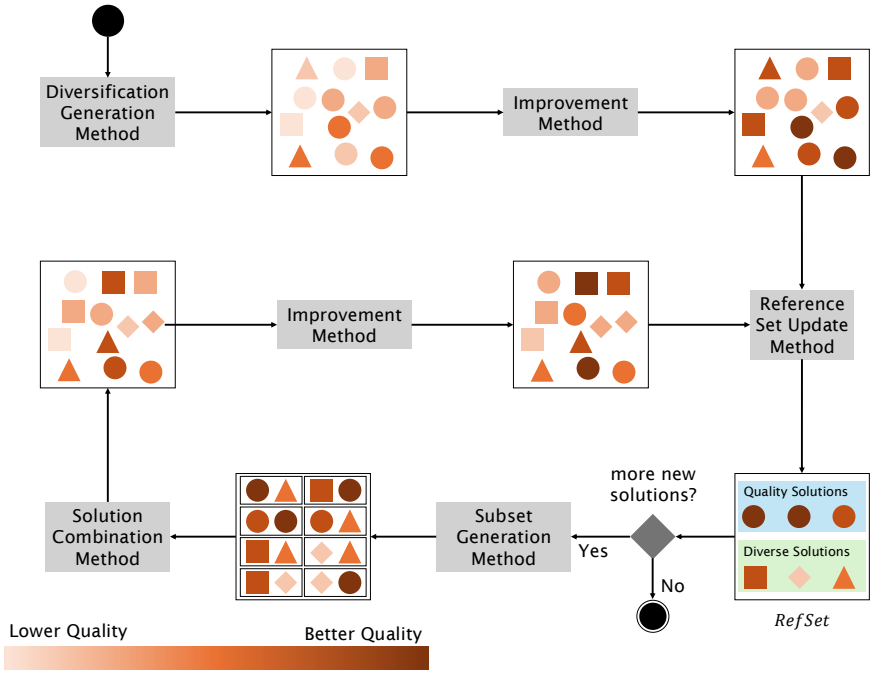

Figure 5. *SS illustration with the iterative RefSet update based on quality (color) and diversity (shape).*

Algorithm 1 Scatter search framework

1: Generate initial population P using the Diversification Generation Method
2: Apply the Improvement Method to each solution in P
3: Build initial *RefSet* from P (quality and diversity)
4: **repeat**
5: Generate subsets from *RefSet* using the Subset Generation Method
6: **for all** subsets S **do**
7: Create new solutions by applying the Solution Combination Method
8: Apply the Improvement Method to each new solution
9: Evaluate new solutions and update *RefSet* if they qualify
10: **end for**
11: **until** no new solutions are added to *RefSet*
12: **return** Best solution(s) from *RefSet*

The main iterative cycle, spanning Steps 5 to 9, combines solutions from the Reference Set, improves the resulting candidates, and updates the set when better or more diverse solutions are found. The search continues until the stopping condition in Step 11 is satisfied, after which the best solution(s) are returned in Step 12. This structure serves as the foundation for the detailed mechanisms described in the following subsections.

3.1. The Scatter Search Methods

SS is organized around five fundamental components that work together to build, improve, and combine high-quality solutions in search of an optimal solution. We now describe each method in detail.

Diversification Generation Method (DGM)

The Diversification Generation Method produces the initial set of solutions, denoted as P. Its goal is to generate many different solutions in an attempt to achieve a certain degree of diversity that results in an extensive exploration of the solution space, thus avoiding a premature convergence of the algorithm. The size of P (*PSize*) is typically at least 10 times the size of *RefSet*. DGM often applies deterministic or semi-random constructive heuristics, instead of the pure random generators applied by many evolutionary methods. A typical design implemented in many recent SS algorithms is based on greedy randomized constructions popularized by GRASP, striking a balance between diversity and initial solution quality. In short, the DGM is the mechanism to create the initial set P that is referred to as a first generation in evolutionary terminology.

Improvement Method (IM)

The IM transforms a given solution of the problem, s, into an enhanced solution s_0. Usually, both solutions are expected to be feasible, but this is not a requirement of the methodology. The IM generally relies on local search (LS) procedures (also known as iterative improvement procedures). As is well known, the local search tries to improve solution s by making 'small changes' in its structure. This may involve operations such as swaps, insertions, or reassignments, applied iteratively until no further improvement is possible or a stopping condition is reached. If an improvement is achieved in this way, then a new solution s_0 is obtained. If no improvement of s was found, then s would be the output of the method.

Reference Set Update Method (RSUM)

The Reference Set, denoted as *RefSet*, contains a small number of high-quality and diverse solutions. One of the main differences between scatter search and other evolutionary methods is that SS operates on this small set of solutions, instead of over the entire population P, as the other methods typically do. The first time that RSUM is run,

it acts as a *RefSet* building procedure. RSUM selects a solution from P to enter into the *RefSet* according to its quality, diversity, or both. Many SS implementations indicate that a good trade-off between quality and diversity is achieved by considering building the *RefSet* with a 50% based on a quality criterion, and the remaining 50% with a diversity criterion; however, but these proportions can be modified depending on the problem being solved.

In a standard initial run of RSUM, the construction starts with the selection of the best $b/2$ solutions from P. For each solution in $P \setminus RefSet$, the minimum distance to the solutions in *RefSet* is computed. Then, the solution that maximizes the minimal distances is selected. This solution is added to *RefSet* and deleted from P, and the minimal distances are updated. This process is repeated $b/2$ times. The resulting reference set has $b/2$ high-quality solutions and $b/2$ diverse solutions.

RSUM evolves the *RefSet* during the search by replacing inferior solutions in terms of quality with newly generated ones. The update operation consists of maintaining a record of the b best solutions found, where the value of b is treated as a constant search parameter

Subset Generation Method (SGM)

The Subset Generation Method creates subsets of solutions from *RefSet* for combinations. These subsets are generally small, containing between two and four elements, and are generated in a systematic way to avoid redundant evaluations. Strategies such as clustering or anti-clustering can be applied to promote intensification or diversification, respectively, depending on the current search needs.

Note that the general SS framework considers the generation of subsets with two, three, and four solutions, but only generates a given subset if its solutions are used to create this subset for the first time. This situation differs from those considered in the context of genetic algorithms, where the combinations are typically determined by the spin of a roulette wheel and are usually restricted to the combination of only two solutions.

Solution Combination Method (SCM)

The Solution Combination Method produces new solutions by combining elements of the subsets generated by the SGM. Instead of generic crossover operators, SS uses problem-specific combination methods designed to preserve and exploit useful features of the parent solutions. These may include convex or non-convex combinations, path relinking, or rule-based synthesis. The resulting solutions are improved, evaluated, and considered for inclusion in *RefSet*, completing the iterative cycle of the method.

It should be mentioned that the convergence of the method is based on the ability to obtain better solutions than those generated in the original *RefSet*. To this end, a customized combination method based on the problem characteristics is more likely to generate quality solutions than a random-based mechanism that only provides diversification

and would require extra effort from the local search (with the associated computing time) to improve the combined solutions. Scatter search usually exhibits shorter running times than other evolutionary methods due to its effective design, which is highly oriented to achieve a good balance between search intensification and diversification.

4. Advanced Search Strategies

Beyond its basic design, scatter search can incorporate a variety of advanced strategies to improve performance and adaptability. In this section, we describe the most effective ones.

Dynamic Reference Set Updating

Instead of updating *RefSet* in fixed intervals or using static replacement rules, dynamic strategies evaluate replacement opportunities continuously. Traditional SS implementations often regenerate a full population of new solutions before considering any updates to the *RefSet*. In contrast, a dynamic approach assesses each new solution generated by the Combination Method and improved by the Improvement Method for its potential inclusion in the *RefSet* immediately. This allows the algorithm to adapt more quickly to promising search trajectories and prevents stagnation (Laguna and Martí, 2003; Martí and León, 2025). The decision to include a new solution often triggers a comparison against the worst solution in the set but can also involve more complex rules, such as replacing the most similar solution to encourage diversity. This "steady-state" update mechanism ensures that high-quality information is incorporated into the reference set without delay, making the search process more responsive and often more efficient (Campos, Laguna and Martí, 1999).

RefSet Rebuild Mechanism

When the *RefSet* becomes stagnant, evidenced by multiple iterations without improvement or the addition of new solutions, a rebuild mechanism can reactivate the search process. This strategy addresses the common problem where the *RefSet* converges to a local region, making it difficult for new solutions to enter due to high-quality barriers. The rebuild process generates a completely new diverse population while preserving the current best solution, then reconstructs the *RefSet* using the standard quality-diversity criteria. This mechanism balances intensification around known good solutions with systematic diversification to explore unexplored regions. The rebuild threshold is typically set as a function of total iterations (e.g., 5-10% of maximum iterations) or based on convergence indicators. This approach ensures continued exploration when traditional combination methods fail to produce improvements, effectively implementing an adaptive restart strategy within the SS framework.

Tiered Reference Sets

A tiered *RefSet* structure divides the set into two or more levels (e.g., 2-tier or 3-tier) according to solution quality or diversity. This advanced mechanism, often called a 2-tier SS, partitions the reference set into a high-quality tier, R_1, and a high-diversity tier, R_2. The upper tier, R_1, contains the best solutions found so far and is used primarily for intensification, generating new solutions by combining elite parents. The lower tier, R_2, maintains a pool of diverse solutions that may not be of the highest quality but are structurally different from those in R_1. Movement between tiers is governed by performance rules: a high-quality solution generated from combinations in R_1 or R_2 can be promoted to R_1, while a solution in R_1 might be relegated to R_2 if it becomes non-competitive or too similar to other elite solutions. This structure creates a formal balance between intensification in the upper tier and diversification in the lower tier (Martí, Laguna and Glover, 2006).

Memory Structures

SS naturally lends itself to the use of memory structures, which can be *explicit* (i.e., storing complete solutions) or *attributive* (i.e., tracking the frequency of individual components or features). The *RefSet* is itself a form of explicit long-term memory. However, more sophisticated strategies incorporate attributive memory, a concept borrowed from Tabu Search (Glover, 1994). Attributive memory tracks characteristics of elite solutions, such as the number of times a specific variable has been assigned a certain value or an edge has been included in a tour. This information can be used to guide the search process. For example, the Diversification Generation Method can be biased to generate solutions containing attributes that have been infrequent in the *RefSet*, thus exploring novel regions of the search space. Conversely, the Combination Method can be guided to prioritize combining attributes that have historically appeared in high-quality solutions. This memory-driven approach allows the search to learn from its history and make more strategic decisions.

Constructive and Destructive Neighborhoods

While the Improvement Method in SS typically employs a local search based on simple neighborhood operators (e.g., swaps), more advanced implementations utilize constructive and destructive approaches, such as those in the Iterated Greedy methodology (Stützle and Ruiz, 2018). A constructive method begins with an incomplete solution, often a high-quality partial configuration derived from elite parents, and intelligently adds elements until a complete, feasible solution is formed. Conversely, a destructive method starts with a complete solution, removes a subset of its components, and then reconstructs it. By alternating between these approaches, the algorithm can explore the search space more thoroughly. Destructive methods, in particular, allow the search space to move away from local optima, while constructive methods can effectively build high-quality solutions based on promising greedy criteria (Martí and Peiró, 2018).

Vocabulary Building

Vocabulary building is a sophisticated strategy that refers to the systematic identification, storage, and reuse of high-quality partial solution components, often called "building blocks" (Glover, 1997). Instead of just combining complete solutions, this approach analyzes the members of the *RefSet* to extract critical sub-structures that are associated with high solution quality. For instance, in a scheduling problem, a building block could be a specific sequence of three jobs; in a graph problem, it could be a well-formed subgraph. These building blocks are stored in a special memory structure (the "vocabulary"). New solutions can then be constructed by assembling these proven components in novel ways, much like forming new sentences from a dictionary of "powerful" words. This method improves the search with a deeper level of learning, allowing it to exploit the underlying problem structure far more effectively than by operating on complete solutions alone.

Parallelization

The modular nature of SS makes it well-suited for parallel computing environments. For example, subsets can be generated and combined independently, and improvement procedures can be applied concurrently to different candidate solutions. This parallelism can significantly reduce computation times for large-scale problems.

The parallelization of scatter search has evolved significantly over the past three decades, reflecting both theoretical advancements and practical demands for scalable metaheuristics. Early foundational works, such as those by Fleurent et al. Fleurent and Valli (1996) and Glover Glover (1997) laid the groundwork for parallel implementations by proposing modular and adaptable frameworks, even though parallelism was not their primary focus. The inherent structure of SS, based on combining subsets of elite solutions, naturally lends itself to parallel execution, particularly in the evaluation and improvement phases.

From the early 2000s onward, researchers began to explore explicit parallelization strategies. García-López et al. García-López and Moreno-Vega (2003) applied parallel SS to the p-median problem, demonstrating its effectiveness in large-scale combinatorial optimization. Adenso-Díaz, García-Carbajal and Lozano (2006) conducted an empirical investigation into parallelization strategies, highlighting trade-offs between synchronous and asynchronous models. Further contributions by García López et al. García López and Moreno-Vega (2006) extended these ideas to feature subset selection, showcasing the adaptability of parallel SS to machine learning contexts.

In computational biology, Penas et al. Penas and Doallo (2015) introduced an asynchronous cooperative enhanced SS tailored for systems biology applications. This work emphasized the benefits of parallel metaheuristics in domains requiring extensive simulation and data integration.

More recently, the field has seen renewed interest in parallel frameworks. Casado et al. Casado and Laguna (2025) proposed a novel parallel architecture and studied it on

problems such as MaxCut and capacitated dispersion. Additionally, Zhao et al. Zhao and Jonrinaldi (2023) and Zuo et al. Zuo and Zhang (2025) developed knowledge-based cooperative SS algorithms for distributed flow shop scheduling, integrating reinforcement learning to guide parallel search processes.

5. Hybridization with Other Methodologies

Scatter search, despite its origins in the 1970s, remains a relevant and powerful metaheuristic. Its maturity has enabled researchers to explore numerous hybridizations and combinations with other optimization strategies. As discussed in previous sections, Path Relinking is one of the most natural extensions of SS, enhancing intensification through trajectory-based exploration between elite solutions (Glover and Martí, 2000). Additionally, advanced strategies have incorporated memory structures from Tabu Search to guide diversification and avoid cycling (Glover, 1994).

5.1. Path Relinking

Path Relinking is a trajectory-based intensification strategy, originally introduced within the scatter search and tabu search methodologies (Glover, 1994, 1997). Its primary purpose is to explore intermediate solutions that lie along a path between two or more high-quality (elite) solutions. Given an *initiating solution* and a *guiding solution*, PR progressively incorporates attributes from the guiding solution into the initiating one, generating a sequence of intermediate solutions. Each of these intermediate solutions is evaluated, and the best one encountered may be considered for inclusion in the high-quality (elite) solutions (that is, in SS, the Reference Set).

In tabu search, any two solutions are linked by a series of moves executed during the search. The path between solutions is determined by the "normal" operational rules of the search procedure. The moves chosen during the relinking process are different from the moves during the "normal" search because the relinking moves do not use the change of the objective function as the guiding principle; they are chosen to get "closer" to the guiding solution.

PR has been extensively applied as the Combination Method in scatter search, usually replacing traditional combination rules. It provides a structured and systematic approach to investigating solution space regions that are often neglected by purely combinatorial combination methods. Instead of directly producing a new solution when combining two or more original solutions, PR generates paths between and beyond the selected solutions in the neighborhood space. This hybridization may be viewed as a highly focused strategy to incorporate attributes of good solutions by creating inducements to favor these attributes in the selected moves, thus creating a path of solutions instead of a single combined solution, as combination methods typically do. This integration often leads to a better balance between intensification and diversification.

We refer the reader to the recent review by Laguna et al. Laguna and Resende (2025) on the hybridization of Path Relinking with GRASP. Some of the most important variants proposed over the last 20 years are:

- *Forward PR* begins with the initiating solution and moves toward the guiding solution.
- *Backward PR*: reverses the process, starting from the guiding solution and moving toward the initiating one.
- *Mixed PR*: explores both forward and backward directions, potentially identifying better intermediate points.
- *Extrapolated PR*: extends the path beyond the guiding solution in search of novel high-quality solutions.
- *Multiparent PR*: generalizes the concept to more than two elite solutions, enabling richer diversification and exploration of multidirectional paths.

In Figure 6, we show an illustration of the *Forward path relinking*, with the initiating (or initial) solution S and the guiding solution T. More generally, we can say that given two solutions S and T to relink, forward path relinking uses the better solution, say for example T, as the guiding solution and the other (S) as the initiating solution. A typical implementation is referred to as *greedy* since upon incorporation in S of the attributes of T not present in S, the updated initiating solution is selected as the one resulting from the introduction of the attribute leading to the solution with best cost. This figure illustrates the path from S to T with its *intermediate solutions*, depicted in black, and several solutions in their neighborhoods, depicted in gray.

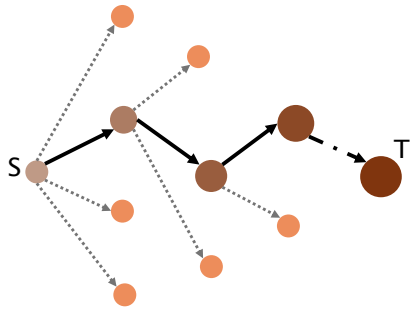

Figure 6. *Forward path relinking from an initial solution S to a guiding solution T.*

The PR method can be easily incorporated into SS by replacing the standard combination step in the Solution Combination Method with the procedure illustrated in Figure 6, or by hybridizing them to alternate between purely combinatorial combinations

and PR-driven trajectories. In practice, PR often leads to significant performance gains, particularly in structured combinatorial problems such as scheduling, routing, or assignment (Glover and Marti, 2003).

5.2. Evolutionary Methods

SS is a member of the broader family of population-based metaheuristics, which also includes evolutionary algorithms. We now examine the conceptual and methodological connections between SS and Genetic Algorithms (GAs), the latter being perhaps the most widely recognized approach within the domain of evolutionary algorithms.

The application of the biological principle of natural evolution to artificial systems, first introduced over three decades ago, has experienced remarkable development in recent years. Collectively referred to as evolutionary algorithms or evolutionary computation, this field encompasses several related paradigms, including genetic algorithms, evolution strategies, evolutionary programming, and genetic programming. Evolutionary algorithms have demonstrated considerable success across a wide range of domains, such as optimization, automated programming, machine learning, economics, ecology, population genetics, evolutionary studies, and social system modeling.

Both scatter search and Genetic Algorithms were introduced in the 1970s. Holland Holland (1975) proposed Genetic Algorithms in 1975, inspired by natural evolution and the principle of "survival of the fittest," while Glover Glover (1977) introduced SS in 1977 as a heuristic for integer programming that extended the concept of surrogate constraints. Although both methodologies maintain and evolve a population (or set) of solutions throughout the search process, they differ in several fundamental ways. Notably, Genetic Algorithms were originally conceived as a framework for hyperplane sampling rather than for optimization. Over time, however, GAs evolved into a methodology primarily focused on solving optimization problems.

Although GAs and SS have contrasting views about searching a solution space, it is possible to create a hybrid approach without entirely compromising the SS framework. Specifically, if we view the crossover and mutation operators as instances of a Combination Method, it is straightforward to design a SS procedure that employs genetic operators for combination purposes. GA operators have been used to replace the combination and improvement phases of SS, yielding robust performance in knapsack and scheduling problems (El-Sayed, El-Wahed and Ismail, 2008). Similarly, Differential Evolution has been embedded to enhance mutation diversity and avoid premature convergence (Li and Tian, 2015).

Martí, Laguna and Campos (2002) compare the performance of SS and GA, employing four classes of problems whose solutions can be represented as permutations. The SS and GA implementations are based on a model that treats the objective function evaluation as a black box, making the search procedures context-independent. This means that neither implementation takes advantage of the structural details of the tests problems. The comparisons are based on experimental testing with four well-known

problems: the linear ordering, the traveling salesperson, matrix bandwidth reduction, and a job-sequencing problem.

The authors compare the SS design with two GAs: one without local search and another with the same local search as SS. The experiments show that the SS procedure is able to obtain high quality solutions from the very beginning of the search. Specifically, after 100,000 objective function evaluations, the percent deviation from the best-known solutions for the SS method is 0.7, while after 1 million evaluations, the GA and GALS methods are still at 4.8 and 0.8, respectively. The strategic choices of SS in this case make a performance difference when compared to the two GA variants. As a conclusion, the authors mentioned that mixing combination strategies with random elements and those based on systematic mechanisms seems to benefit both procedure-based GAs and SS methodologies. Thus, it is confirmed what is well-known nowadays: that hybrid designs may obtain superior outcomes across different types of problems.

5.3. Trajectory-based Methods

Tabu Search remains a natural partner for SS due to their shared emphasis on memory structures and strategic exploration. TS has been used to enhance the improvement phase or to guide the selection of promising regions in multiobjective optimization (Beausoleil, 2005). Similarly, Variable Neighborhood Search (VNS) has also been incorporated as an advanced improvement method to dynamically adjust neighborhood structures during the search (Fahim and Hedar, 2014).

Swarm-based algorithms, such as Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO), have also been combined with SS. PSO has been used to guide the search in flow shop scheduling, while ACO has been integrated to improve solution quality in multicast routing problems (Chunxin and Wangpeng, 2018). These hybrids leverage the collective intelligence of swarms with the structured memory and combination mechanisms of SS.

Recent innovations include the use of chaotic maps to generate pseudorandom numbers for SS initialization, combination, and improving exploration in complex landscapes (Davendra and Pluhacek, 2014). Binary adaptations of SS have also been proposed for discrete problems, with tailored diversification and combination operators (Gortázar and Martí, 2010).

Multiobjective variants of SS, such as AbYSS and MOSS, incorporate Pareto dominance, external archives, and density estimation to handle trade-offs between conflicting objectives (Beausoleil, 2005; Nebro and Beham, 2008). These frameworks have demonstrated competitive performance in engineering design, scheduling, and network optimization.

More recent developments include the integration of reinforcement learning and knowledge-based mechanisms into SS. These hybrids adaptively guide the search based on learned patterns, as seen in distributed flow shop scheduling problems (Zhao and Jonrinaldi, 2023; Zuo and Zhang, 2025).

6. Applications

Scatter search has proven to be a highly adaptable and effective metaheuristic, successfully addressing a wide range of optimization problems, from classical combinatorial formulations to complex, large-scale real-world challenges involving uncertainty, nonlinearity, and simulation. Its structured approach to combining elite solutions, leveraging adaptive memory, and promoting controlled diversification has made it a valuable tool in diverse domains such as logistics, manufacturing, finance, healthcare, and bioinformatics.

Table 1 presents an overview of representative SS applications between 2000 and 2025, organized by thematic area.

Application Area	**References**
Scheduling & Timetabling	(Manikas, and Chang, 2009; Mansour, Isahakian and Ghalayini, 2011; Pendharkar, 2013; Ranjbar, De Reyck and Kianfar, 2009; Vandenheede, Vanhoucke and Maenhout, 2016; Vanhoucke, 2010; Yamashita, Armentano and Laguna, 2006; Zhao et al., 2023; Zuo, Zhao and Zhang, 2025)
Transportation & Routing	(Belfiore and Yoshizaki, 2009; Chu, Labadi and Prins, 2006; Keskin, and Üster, 2007; Piñol, and Beasley, 2006; Russell, and Chiang, 2006; Tang, Zhang and Pan, 2010; Zhang, Chaovalitwongse and Zhang, 2012)
Data Mining & Feature Selection	(Duman and Ozcelik, 2011; Gharehchopogh, and Amjad, 2019; García López, et al., 2006; Pacheco, 2005; Wang, et al., 2012; Wang, et al., 2014)
Healthcare	(Burke, et al., 2010; Maenhout and Vanhoucke, 2006)
Medical Imaging & Forensics	(Cordón, Damas and Santamaría, 2006; Cordón, et al., 2008; Ibáñez, et al., 2012; Santamaría, et al., 2007; Valsecchi, et al., 2014)
Bioinformatics & Computational Biology	(Boumedine, and Bouroubi, 2021; Egea, et al., 2014; Mansour, Kehyayan, and Khachfe, 2009; Remil, et al., 2019)
Electrical & Electronic Engineering	(Hung, et al., 2002; Yang, et al., 2025)
Facility Location, Layout & Network Design	(Crainic, and Gendreau, 2007; Hakli, and Ortacay, 2019; Keskin, and Üster, 2007; Khooban, et al., 2015; Kothari, and Ghosh, 2014)
Manufacturing & Disassembly/Assembly Planning	(González, and Adenso-Díaz, 2006; Jabal-Ameli, and Moshref-Javadi, 2014; Prabhaharan, Ramesh and Asokan, 2007; Rahimi-Vahed, et al., 2007)
Civil/Water Resources	(Baños, et al., 2009; Liberatore, and Sechi, 2009)
Software Engineering & Testing	(Blanco, Tuya and Adenso-Díaz, 2009; Liu et al., 2021; Ren and Zhu, 2023; Sagarna and Lozano, 2006)

Table 1. *Representative applications of scatter search (2000–2025) across multiple domains.*

One of the most impactful industrial implementations of SS is the *OptQuest* optimization engine (Laguna and Martí, 2003b). OptQuest integrates scatter search with simulation models to tackle problems too complex for conventional mathematical programming. It functions as a *black-box optimizer*, meaning it can guide the search process even when the objective function is implicit, noisy, or computationally expensive, relying solely on input–output evaluations. This capability allows it to optimize production scheduling, inventory planning, network design, and workforce allocation under uncertainty.

Beyond OptQuest, several works have proposed specialized *black-box* implementations of SS tailored to specific problem classes (Gortázar and Martí, 2010; Laguna and Martí, 2014; Laguna and Hernández-Díaz, 2010), expanding its applicability to domains where direct analytical formulations are not feasible.

7. Tutorial: 0-1 Knapsack Problems

We now show a practical implementation of scatter search. In particular, we present a comprehensive tutorial to solve the well-known 0-1 knapsack problem. This classic combinatorial optimization problem serves as an excellent testbed for demonstrating the key components of the SS methodology, while its binary nature makes it particularly suitable for educational purposes.

In addition to the explanations provided in this section, we have developed a comprehensive Jupyter Notebook that includes detailed information, execution traces, visualizations, and additional analysis. The notebook, which we recommend for a deeper understanding, can be accessed via the following link: https://github.com/scaverod/Scatter-Search-Tutorial-0-1-Knapsack-Problems. We have also created a concise and self-contained Python implementation. The code presented below corresponds to this simplified version. For clarity and brevity, we have omitted error checking, input validation, default parameter values, and exception handling from the code listings.

7.1. Problem Formulation

The 0-1 Knapsack Problem is formally defined as:

$$\text{maximize} \quad Z = \sum_{i=1}^{n} v_i x_i \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \leq W \tag{2}$$

$$x_i \in \{0,1\}, \quad i = 1,\ldots,n \tag{3}$$

where v_i denotes the profit of item i, w_i its weight, W the knapsack capacity, and x_i a binary decision variable. The objective is to maximize total profit without exceeding the available capacity.

In the examples and experiments presented throughout this section, we use a specific instance of the 0-1 Knapsack Problem to illustrate the performance of the scatter search algorithm. The problem can be formulated as:

$$\begin{aligned}
\text{maximize} \quad & Z = 24x_1 + 18x_2 + 15x_3 + 40x_4 + 22x_5 + 33x_6 + 17x_7 + 28x_8 \\
& \quad + 19x_9 + 31x_{10} + 25x_{11} + 14x_{12} + 37x_{13} + 21x_{14} \\
& \quad + 16x_{15} + 29x_{16} + 23x_{17} + 35x_{18} + 20x_{19} + 27x_{20} \qquad (4) \\
\text{subject to} \quad & 12x_1 + 25x_2 + 18x_3 + 32x_4 + 20x_5 + 27x_6 + 15x_7 + 22x_8 \\
& \quad + 19x_9 + 24x_{10} + 17x_{11} + 13x_{12} + 30x_{13} + 21x_{14} \\
& \quad + 16x_{15} + 26x_{16} + 14x_{17} + 28x_{18} + 23x_{19} + 20x_{20} \leq 150 \qquad (5) \\
& x_i \in \{0,1\}, \quad i = 1,\ldots,20 \qquad (6)
\end{aligned}$$

7.2. Data Structures and Problem Representation

The implementation begins with a clear data structure for representing knapsack instances. Specifically, it is necessary to store the profit and weight associated with each item, as these are the fundamental parameters that define the problem. Additional properties and methods can be defined to facilitate efficient algorithmic operations, such as computing the number of items or the efficiency ratio of each item.

Code 1 shows a `frozen` dataclass in Python that ensures immutability and provides computed properties for algorithmic efficiency.

Code 1: Problem representation with efficiency ratios

```python
@dataclass(frozen=True)
class KnapsackProblem:
    profits: List[int]
    weights: List[int]
    capacity: int

    @property
    def n(self): # number of items
        return len(self.profits)

    @property
    def efficiency_ratios(self): # v_i/w_i
        return np.array(self.profits) / np.array(self.weights)

    @property
    def total_weight(self):
        return sum(self.weights)
```

7.3. Core Problem Operations

Next, we define in Code 2 three fundamental operations that will be used throughout the optimization process and are generic to any knapsack instance: objective evaluation, weight calculation, and feasibility checking. Additionally, we include the function `ratio_order`, which returns the indices of items sorted by their efficiency ratio (profit-to-weight). These operations are not specific of SS and can be reused in other metaheuristic approaches.

Code 2: Essential knapsack operations

```
def objective(pb, x):
    return int(sum(p * s for p, s in zip(pb.profits, x)))

def total_weight(pb, x):
    return int(sum(w * s for w, s in zip(pb.weights, x)))

def is_feasible(pb, x):
    return total_weight(pb, x) <= pb.capacity

def ratio_order(pb, ascending=False):
    indices = list(range(pb.n))
    ratios = pb.efficiency_ratios
    indices.sort(key=lambda i: ratios[i], reverse=not ascending)
    return indices
```

7.4. Systematic Diversification

The Diversification Generation Method constructs an initial pool of structurally diverse binary solutions using two complementary strategies: systematic toggling patterns based on (h,q) parameters, and capacity-aware random selection. This dual approach ensures broad coverage of the solution space while adapting to the specific constraints of the knapsack problem.

The first strategy relies on deterministic toggling patterns. For each combination of step size h and phase offset q, the algorithm toggles the positions $q-1, q-1+h, q-1+2h, \ldots$ in a binary seed vector. This produces a new solution and its binary complement. The resulting structural diversity contributes to a systematic exploration of the search space. This mechanism aligns with the SS principle of generating solutions that are scattered across the solution space (Glover, 1997; Martí and Glover, 2006).

Although this method can generate solutions that span the entire space, many of them may be infeasible, particularly when the number of active items (ones) is high. To mitigate this, the second strategy estimates the number of items that can reasonably fit

into the knapsack by dividing the total capacity by the average item weight. A small random deviation is then applied to this estimate, and the algorithm randomly selects the corresponding number of positions to activate. More formally,

$$target\_items = \max\left(1, \min\left(n, \left\lfloor \frac{W}{\overline{w}} \right\rfloor + \alpha\right)\right) \tag{7}$$

where W is the knapsack capacity, $\overline{w}$ is the average item weight, n is the number of items, and α is a small random integer deviation. This formulation guides the generation of solutions that are close to the feasible region, increasing the likelihood of producing valid candidates.

Unlike population-based metaheuristics such as genetic algorithms, which depend heavily on stochastic operators like crossover and mutation, SS may exploit problem-specific knowledge to guide the generation of new solutions. By embedding structural information—such as efficiency ratios in knapsack problems or precedence relations in scheduling tasks—the method achieves a more directed exploration of the search space and reduces reliance on random variation, thus promoting faster convergence toward high-quality solutions.

Now, we illustrate the construction process. Consider an initial seed vector of length $n = 20$ corresponding to the instance described in Section 7. For step size $h = 2$ and phase $q = 1$, the algorithm toggles the positions $[1,3,5,7,9,11,13,15,17,19]$, resulting in the solution $[1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0]$. Its binary complement is $[0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1]$.

Next, the capacity-aware randomization step is applied to both solutions. Based on the equation above, the estimated number of items that can fit is $target\_items = 5$. Five positions are randomly selected to remain active, while the rest are set to zero. For instance, the randomized version of the original solution might be $[0,0,1,0,0,0,1,0,1,0,0,$ $0,1,0,0,0,1,0,1,0]$, while the randomized complement could be $[0,1,0,0,0,1,0,1,0,0,$ $0,0,0,1,0,1,0,0,0,1]$. These two solutions are then used as seeds for subsequent (h,q) combinations, and the process continues iteratively.

Code 3 shows the implementation of the complete diversification generation method. Functions `create_systematic_solution(seed, h, q)` and `fit_to_target_items (solutions, target_items)` have been omitted for the sake of brevity. The first one, `create_systematic_solution(seed, h, q)`, generates a solution by toggling the elements of the `seed` vector at positions determined by the step size `h` and the phase offset `q`. The second one, `fit_to_target_items(solutions, target_items)`, applies the capacity-aware randomization step. This function modifies the input solutions by randomly selecting `target_items` positions to remain active (set to 1) and deactivating all others (set to 0), thus ensuring the solution's density is close to the estimated feasible capacity.

Code 3: Complete diversification generation method

```python
def generate_diverse_solutions(pb, target):
    n = pb.n
    hmax = min(max(2, n - 1), 10)
    max_items_estimate = estimate_capacity_items(pb)
    max_deviation = max(1, max_items_estimate // 3)
    pool = []
    seed = [0] * n

    while len(pool) >= target:
        for h in range(2, min(hmax + 1, 5)):
            for q in range(1, h + 1):
                # We generate a solution and its complementary
                systematic_solution = create_systematic_solution(seed, h, q)
                complementary_solution = [1 - x for x in systematic_solution]
                seed = systematic_solution # Update the seed for next iteration

                # Fit to target items
                alpha = random.randint(-max_deviation, max_deviation)
                target_items = max(1, min(n, max_items_estimate + alpha))
                fit_to_target_items(solutions, target_items) # Capacity-aware randomization step

                for solution in [systematic_solution, complementary_solution]:
                    if tuple(solution) not in pool:
                        pool.append(solution)

    return pool[:target] # return the first "target" solutions of the pool
```

To assess the effectiveness of the diversification method, we generated 100 solutions for the knapsack instance. Figure 7 shows a scatter plot in which each point (i.e., a solution) is plotted according to total weight (x-axis) and total value (y-axis). The gray dashed vertical line marks the knapsack capacity ($W = 150$); solutions to the right, colored red, are infeasible and require repair.

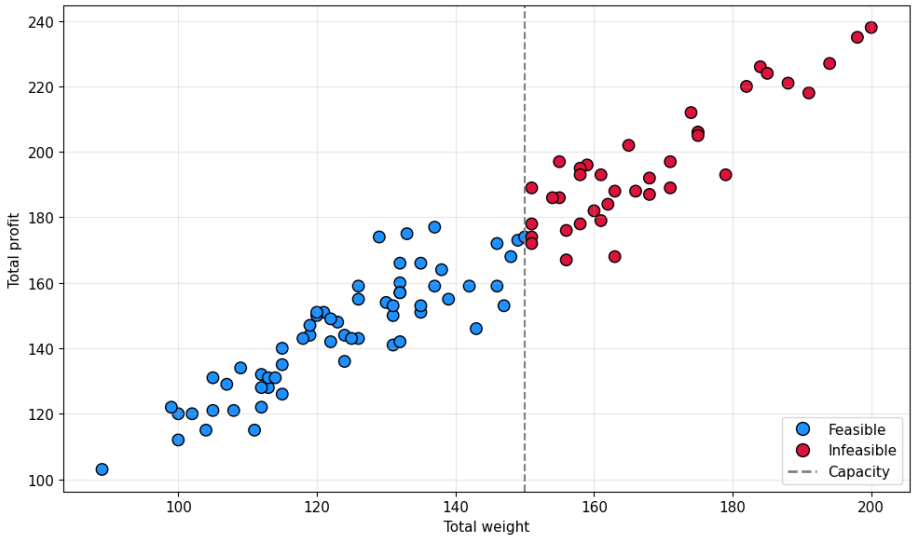

Figure 7. *Scatter plot of 100 diversified solutions. Dashed vertical line indicates knapsack capacity ($W = 150$).*

Among the 100 generated solutions, 62% are feasible. The value ranges from 103 to 238 (mean: 164.5), and the weights range from 89 to 200 (mean: 140.6). All solutions are unique, although profit or weight may be identical.

7.5. Two-Phase Improvement Method

The improvement method ensures that all solutions are both feasible and locally optimal through a systematic two-phase approach: the repair phase and the quality enhancement phase.

The repair phase receives a potentially infeasible solution and systematically removes items to achieve feasibility. Code 4 shows how items are removed in order of worst efficiency until the solution becomes feasible.

The enhancement phase receives a feasible solution and attempts to improve its quality by adding items. Code 5 shows how items are added in order of best efficiency while maintaining feasibility.

Code 4: Feasibility repair using efficiency ordering

```
def remove_worst_items(pb, x):
    repaired_solution = x[:] # clone the given solution
    current_value = objective(pb, repaired_solution)
    efficiency_order_asc = ratio_order(pb, ascending=True)

    for item_idx in efficiency_order_asc:
        if is_feasible(pb, repaired_solution):
            break
        if repaired_solution[item_idx] == 1: # if item is
    selected
            repaired_solution[item_idx] = 0 # remove the item
            current_value -= pb.profits[item_idx] # remove its
    profit

    return repaired_solution, current_value
```

Code 5: Greedy enhancement using efficiency ordering

```
def add_best_items(pb, x, current_value):
    improved_solution = x[:]
    efficiency_order_desc = ratio_order(pb, ascending=False)

    for item_idx in efficiency_order_desc:
        if improved_solution[item_idx] == 0: # it item is not
    selected
            new_weight = total_weight(pb, improved_solution) + pb
    .weights[item_idx]
            if new_weight <= pb.capacity: # check if it would fit
                improved_solution[item_idx] = 1 # if so, we add
    it
                current_value += pb.profits[item_idx] # and
    update the total profit

    return improved_solution, current_value
```

Finally, the complete improvement method coordinates both phases. Figure 8 shows the trajectory of a solution through both phases. Red points indicate infeasible states, blue points are feasible, and arrows trace the improvement path. In particular, point 0 is the initial infeasible solution with weight 218 and value 249. During the repair phase,

items are removed sequentially (solutions 1, 2, 3) based on ascending efficiency until feasibility is reached at solution 4 (weight 133, value 167). In the enhancement phase, the most efficient item is added without violating the capacity constraint, resulting in solution 5 (weight 145, value 191).

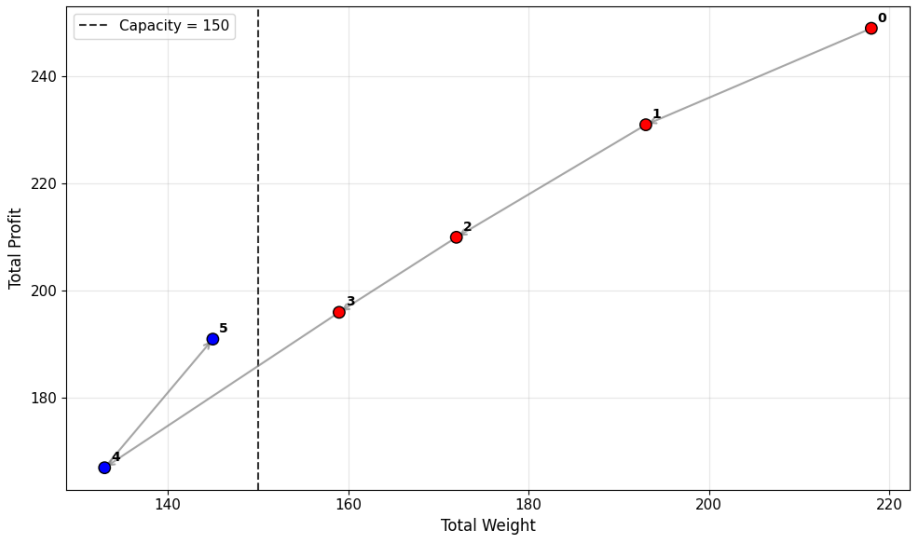

Figure 8. *Improvement trajectory in weight-value space.*

7.6. Reference Set Management

The Reference Set (*RefSet*) is constructed using a two-phase strategy that balances solution quality and structural diversity. In the first phase, the top b_1 solutions are selected based on objective value, ensuring intensification around promising regions. In the second phase, diversity is introduced by selecting b_2 solutions that maximize the minimum Hamming distance to the current *RefSet*. The Hamming distance between two binary vectors x and y is defined as:

$$d_H(x,y) = \sum_{i=1}^{n} |x_i - y_i| \tag{8}$$

Code 6 shows the implementation of this process. The function `hamming_distance` computes the number of differing bits between two solutions, and `create_refset` builds the *RefSet* by first selecting the best solutions, then adding the most structurally diverse ones.

Code 6: Reference Set creation balancing quality and diversity

```python
def hamming_distance(x, y):
    return sum(abs(xi - yi) for xi, yi in zip(x, y))

def min_hamming_dist_to_refset(s, rs_solutions):
    return min(hamming_distance(s, r) for r in rs_solutions)

def create_refset(solutions_with_values, b, b1):
    P_sorted = sorted(solutions_with_values, key=lambda t: t[1],
    reverse=True)
    rs_solutions = [P_sorted[i][0][:] for i in range(min(b1, len(
    P_sorted)))]
    rs_values = [P_sorted[i][1] for i in range(min(b1, len(
    P_sorted)))]

    candidates = [t for t in P_sorted if tuple(t[0]) not in {
    tuple(r) for r in rs_solutions}]
    while len(rs_solutions) < b and candidates:
        best = max(candidates, key=lambda t:
    min_hamming_dist_to_refset(t[0], rs_solutions))
        rs_solutions.append(best[0][:])
        rs_values.append(best[1])
        candidates = [t for t in candidates if tuple(t[0]) !=
    tuple(best[0])]

    return rs_solutions, rs_values
```

For this tutorial, we consider a pool of 30 solutions, a *RefSet* of size $b = 8$, composed of $b_1 = 4$ high-quality solutions and $b_2 = 4$ diverse solutions. Figure 9 visualizes the selection process. Red points represent the top 4 solutions selected for quality (b_1), located highest on the vertical axis (value). Blue points correspond to the 4 most diverse solutions (b_2), positioned farthest to the right on the horizontal axis (minimum Hamming distance to *RefSet*). Gray points are non-selected candidates.

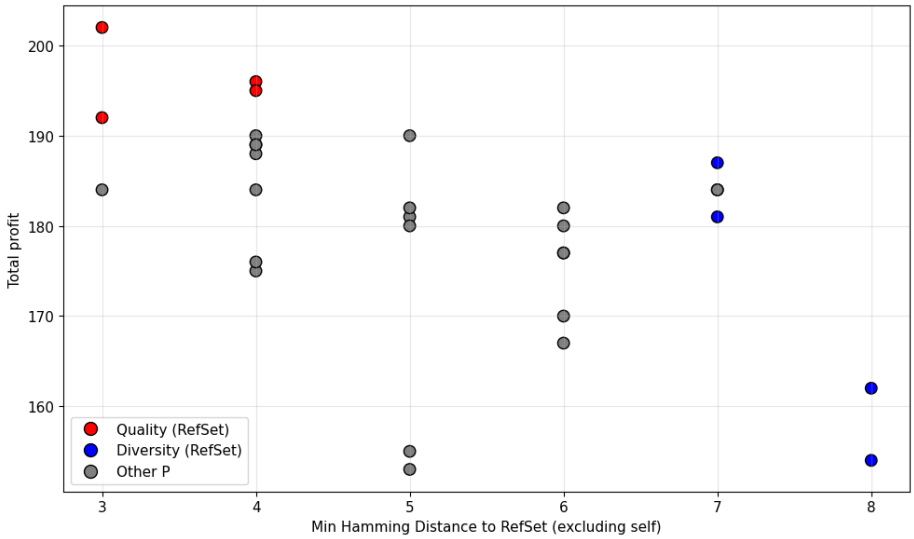

Figure 9. *Reference Set selection from pool P. Red: high-quality (b_1), Blue: diverse (b_2), Gray: non-selected. X-axis: minimum Hamming distance to RefSet, Y-axis: objective value.*

7.7. Solution Combination Method

The combination method generates new trial solutions by combining multiple *RefSet* members using a quality-weighted scoring approach. Unlike traditional pairwise crossover, this method supports variable-sized subsets and produces multiple candidate solutions per combination.

For each variable position (item) i, a score is computed based on the weighted average of the values across the selected parent solutions:

$$\text{score}(i) = \frac{\sum_{j \in \text{subset}} x_i^{(j)} \cdot \text{ObjVal}(j)}{\sum_{j \in \text{subset}} \text{ObjVal}(j)} \tag{9}$$

Trial solutions are generated by applying different thresholds to these scores. If $\text{score}(i) \geq r$, then $x_i = 1$; otherwise, $x_i = 0$. Multiple thresholds are used to produce diverse candidates.

Code 7 shows the implementation of this method. The function computes weighted scores and generates several trial solutions using fixed and random thresholds. Edge cases are handled to avoid empty solutions.

Code 7: Multi-solution combination using weighted scoring

```
def combine_multiple(pb, solutions, values):
    total_value = sum(max(0, v) for v in values)
    weights = [1.0 / len(values)] * len(values) if total_value ==
     0
                                          else [max(0, v) /
    total_value for v in values]

    scores = []
    for i in range(pb.n):
        weighted_sum = sum(solutions[j][i] * weights[j] for j in
    range(len(solutions)))
        score = max(0.0, min(1.0, weighted_sum))
        scores.append(score)

    trials = []
    thresholds = [0.2, 0.4, 0.5, 0.6, 0.8]
    for threshold in thresholds:
        trial = [1 if scores[i] >= threshold else 0 for i in
    range(pb.n)]
        if sum(trial) == 0:
            high_indices = [i for i, s in enumerate(scores) if s
    > 0.1]
            if high_indices:
                trial[random.choice(high_indices)] = 1
        trials.append(trial)

    return trials
```

To illustrate the process, consider the following subset of three parent solutions selected from the RefSet:

- **Parent 1:** $[1,0,0,1,1,1,0,0,0,0,1,0,0,0,0,0,1,1,0,0]$, with an objective value of 202.
- **Parent 2:** $[1,0,1,0,0,1,1,0,0,1,1,0,0,0,0,0,1,0,0,1]$, with an objective value of 195.
- **Parent 3:** $[1,0,0,0,0,0,1,0,1,1,1,1,0,0,1,0,0,1,0,0]$, with an objective value of 181.

For each variable position, we compute a quality-weighted score. First, the total objective value is calculated: $202+195+181=578$. This sum serves as the denominator for the weighted average.

The score for each position i is then computed using the formula. For example, the scores for the first and third positions are: $\text{Score}(1) = \frac{(1\cdot202)+(1\cdot195)+(1\cdot181)}{578} = 1.000$, and $\text{Score}(3) = \frac{(0\cdot202)+(1\cdot195)+(0\cdot181)}{578} \approx 0.337$.

The Table 2 details the scores for all variable positions and the resulting binary values for two trial solutions generated with thresholds $r = 0.3$ and $r = 0.6$. A variable is set to 1 if its score is greater than or equal to the threshold.

Table 2. *Calculation of scores and generation of trial solutions*

Var. Index (i)	**P1**	**P2**	**P3**	**Score**(i)	**Trial** ($r \geq 0.3$)	**Trial** ($r \geq 0.6$)
1	1	1	1	1.000	1	1
2	0	0	0	0.000	0	0
3	0	1	0	0.337	1	0
4	1	0	0	0.349	1	0
5	1	0	0	0.349	1	0
6	1	1	0	0.687	1	1
7	0	1	1	0.651	1	1
8	0	0	0	0.000	0	0
9	0	0	1	0.313	1	0
10	0	1	1	0.651	1	1
11	1	1	1	1.000	1	1
12	0	0	1	0.313	1	0
13	0	0	0	0.000	0	0
14	0	0	0	0.000	0	0
15	0	0	1	0.313	1	0
16	0	0	0	0.000	0	0
17	1	1	0	0.687	1	1
18	1	0	1	0.663	1	1
19	0	0	0	0.000	0	0
20	0	1	0	0.337	1	0

7.8. Algorithm Orchestration

The scatter search procedure is structured around a modular and iterative process that coordinates all components: diversification, improvement, reference set management, subset generation, solution combination, and convergence control. Code 8 presents the complete orchestration.

The algorithm begins by generating a diverse pool of candidate solutions using the Diversification Generation Method. These solutions are then improved through a two-phase procedure that ensures feasibility and local optimality. The improved solutions are used to construct the initial Reference Set (*RefSet*), which balances quality and diversity.

Code 8: Complete scatter search algorithm

```
\lstset{ %
showspaces=false,           % show spaces adding particular
    underscores
showstringspaces=false,     % underline spaces within strings
showtabs=false,         % show tabs within strings adding
    particular underscores
frame=single,           % adds a frame around the code
tabsize=2,              % sets default tabsize to 2 spaces
captionpos=b,           % sets the caption-position to bottom
breaklines=true,            % sets automatic line breaking
breakatwhitespace=false,        % sets if automatic breaks should
     only happen at whitespace
escapeinside={\%*}{*)}      % if you want to add a comment within
     your code
}
def run_scatter_search(pb, max_iter, refset_size,
    max_iter_no_impr):
    # 1. Diversification Generation
    initial_pool = generate_diverse_solutions(pb, target=30)

    # 2. Improvement
    improved_pool = []
    for sol in initial_pool:
        improved_sol, value = improve_solution(pb, sol)
        improved_pool.append((improved_sol, value))

    # 3. Create initial RefSet
    rs_solutions, rs_values = create_refset(improved_pool, b=
    refset_size, b1=refset_size//2)

    best_value = max(rs_values)
    best_solution = rs_solutions[rs_values.index(best_value)]
    no_improvement = 0

    # 4. Main iterative loop
    for iteration in range(max_iter):
        added_new = False

        # 5. Generate subsets and combine solutions
        for i in range(len(rs_solutions)):
            for j in range(i+1, len(rs_solutions)):
                subset_sols = [rs_solutions[i], rs_solutions[j]]
```

```
                subset_vals = [rs_values[i], rs_values[j]]

                # 6. Combination and improvement
                trials = combine_multiple(pb, subset_sols,
    subset_vals)

                for trial in trials:
                    improved_trial, trial_value =
    improve_solution(pb, trial)

                    # 7. RefSet update
                    trial_tuple = tuple(improved_trial)
                    already_exists = any(tuple(sol) ==
    trial_tuple for sol in rs_solutions)

                    if not already_exists and trial_value > min(
    rs_values):
                        worst_idx = rs_values.index(min(rs_values
    ))
                        rs_solutions[worst_idx] = improved_trial
                        rs_values[worst_idx] = trial_value
                        added_new = True

        # Update best solution
        current_best = max(rs_values)
        if current_best > best_value:
            best_value = current_best
            best_solution = rs_solutions[rs_values.index(
    best_value)]
            no_improvement = 0
        else:
            no_improvement += 1

        # Patience-based termination
        if no_improvement >= max_iter_no_impr:
            break

    return best_solution, best_value
```

The main loop iteratively generates subsets from the *RefSet*, combines their members using the weighted scoring method, and applies the improvement procedure to the resulting trials. If a trial solution is both new and better than the worst in the *RefSet*, it replaces it. This dynamic update mechanism ensures that the *RefSet* evolves with the search, maintaining a balance between intensification and diversification.

The algorithm tracks the best solution found so far and uses a patience-based stopping criterion: if no improvement is observed over a fixed number of iterations, the search terminates. This is a practical alternative to the classical convergence condition used in scatter search, which stops when no new solutions are added to the *RefSet*. Both approaches aim to prevent unnecessary computation once the search stagnates.

Variants of scatter search may include elite solution preservation, tiered reference sets, or adaptive memory structures. These extensions enhance robustness and adaptability, especially in dynamic or multiobjective contexts.

To finalize the tutorial, we execute the complete scatter search algorithm on the test instance. Figure 10 shows the evolution of the best solution value across iterations. The initial population yields a best value of 201. After two *RefSet* update cycles, the value improves to 204 and then to 207, demonstrating effective intensification. The curve reflects a typical behavior: rapid early improvement followed by stabilization, indicating convergence. Note that, the performance profile shown here is reasonable for this instance, but it may vary significantly across different problems or datasets.

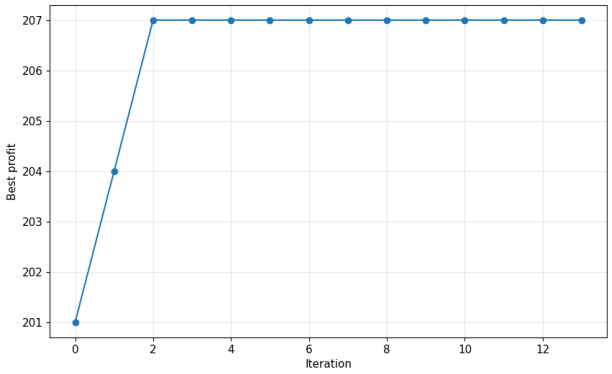

Figure 10. *Best solution value over iterations.*

Finally, it is worth mentioning that the orchestration of SS involves several parameters whose configuration critically affects algorithmic performance. Key parameters include the size of the initial pool, the Reference Set size and its quality/diversity ratio, the number of subsets generated per iteration, the thresholds used in the combination method and stopping criteria such as maximum iterations. While manual tuning based on domain expertise remains common, automated configuration tools such as irace (López-Ibáñez and Stützle, 2016) have proven effective for systematically exploring parameter spaces and identifying robust settings. These tools employ racing strategies to compare

candidate configurations under limited computational budgets, thereby reducing the risk of overfitting and improving reproducibility in experimental studies.

8. Conclusions

Scatter search and path relinking have evolved into robust and versatile metaheuristics, distinguished by their strategic design and memory-based learning mechanisms. Unlike traditional evolutionary algorithms that rely heavily on stochastic variation, structured combination, and adaptive memory, SS emphasizes a deliberate balance between intensification and diversification.

This review has traced the historical development of SS, from its foundational principles to its integration with PR and its expansion into advanced strategies such as dynamic reference set updating, tiered memory structures, and vocabulary building. These enhancements have significantly improved the algorithm's responsiveness, scalability, and ability to escape local optima.

The adaptability of SS has been demonstrated across a wide range of applications, including scheduling, routing, bioinformatics, medical imaging, and software engineering. Its modular architecture has facilitated parallelization and hybridization with other metaheuristics, such as Genetic Algorithms, Differential Evolution, Particle Swarm Optimization, and Tabu Search. These combinations have yielded high-performance algorithms capable of tackling large-scale and multiobjective problems.

Future research may focus on automated configuration of SS components, deeper integration with simulation-based optimization, and the incorporation of learning mechanisms such as reinforcement learning. The continued exploration of SS in emerging domains, including dynamic and uncertain environments, will further solidify its role as a foundational tool in the metaheuristic landscape.

Acknowledgements

This research has been partially supported with grants PID2021-125709OB-C21 and PID2024-160226OB-C21 funded by the Spanish Government (MCIN/AEI/10.13039/501 100011033) and by ERDF-A way of making Europe. It has also been supported by the Generalitat Valenciana (CIAICO/2021/224) and the Universidad Rey Juan Carlos (2024/SOLCON-82652 and 2025/SOLCON-95639).

References

Adenso-Díaz, B., García-Carbajal, S., and Lozano, S. (2006). An empirical investigation on parallelization strategies for scatter search. *European Journal of Operational Research*, 169(2):490–507.

Baños, R., Gil, C., Reca, J., and Martínez, J. (2009). Implementation of scatter search for multi-objective optimization: a comparative study. *Computational Optimization and Applications*, 42(3):421–441.

Beausoleil, R. P. (2005). MOSS multiobjective scatter search applied to non-linear multiple criteria optimization. *European Journal of Operational Research*, 169(2):426–449.

Belfiore, P. C. and Yoshizaki, H. T. Y. (2009). Scatter search for a real-life heterogeneous fleet vehicle routing problem with time windows and split deliveries in Brazil. *European Journal of Operational Research*, 199(3):750–758.

Blanco, R., Tuya, J., and Adenso-Díaz, B. (2009). Automated test data generation using a scatter search approach. *Information and Software Technology*, 51(4):708–720.

Boumedine, N. and Bouroubi, S. (2021). Protein structure prediction in the HP model using scatter search algorithm. *Bulletin du Laboratoire*, 4:73–85.

Burke, E. K., Curtois, T., Qu, R., and Vanden Berghe, G. (2010). A scatter search methodology for the nurse rostering problem. *Journal of the Operational Research Society*, 61(11):1667–1679.

Campos, V., Laguna, M., and Martí, R. (1999). Scatter search for the linear ordering problem. *New Ideas in Optimization*, 331:339.

Casado, A., Pérez-Peló, S., Sánchez-Oro, J., Duarte, A., and Laguna, M. (2025). A novel parallel framework for scatter search. *Knowledge-Based Systems*, 314:113248.

Chu, F., Labadi, N., and Prins, C. (2006). A scatter search for the periodic capacitated arc routing problem. *European Journal of Operational Research*, 169(2):586–605.

Chunxin, S., Xiaoxia, Z., Hongyang, C., Jiao, Y., and Wangpeng, W. (2018). A hybrid scatter search algorithm for QoS multicast routing problem. In *Chinese Control and Decision Conference (CCDC)*, pages 4875–4878.

Cordón, O., Damas, S., and Santamaría, J. (2006). A fast and accurate approach for 3D image registration using the scatter search evolutionary algorithm. *Pattern Recognition Letters*, 27(11):1191–1200.

Cordón, O., Damas, S., Santamaría, J., and Martí, R. (2008). Scatter search for the point-matching problem in 3D image registration. *INFORMS Journal on Computing*, 20(1):55–68.

Cordón, O., Damas, S., and Santamaría, J. (2004). A scatter search algorithm for the 3D image registration problem. In *Parallel Problem Solving from Nature – PPSN VIII*, pages 471–480. Springer.

Crainic, T. G. and Gendreau, M. (2007). A scatter search heuristic for the fixed-charge capacitated network design problem. In Doerner, K. F. et al. (Eds.), *Metaheuristics: Progress in Complex Systems Optimization*, pages 25–40. Springer US, Boston, MA.

Davendra, D., Senkerik, R., Zelinka, I., and Pluhacek, M. (2014). Scatter search algorithm with chaos based stochasticity. In *IEEE Congress on Evolutionary Computation*.

Duman, E. and Ozcelik, M. H. (2011). Detecting credit card fraud by genetic algorithm and scatter search. *Expert Systems with Applications*, 38(10):13057–13063.

Egea, J. A., Henriques, D., Cokelaer, T., et al. (2014). MEIGO: an open-source software suite based on metaheuristics for global optimization in systems biology and bioinformatics. *BMC Bioinformatics*, 15(1):136.

El-Sayed, S. M., El-Wahed, W. F. A., and Ismail, N. A. (2008). A hybrid genetic scatter search algorithm for solving optimization problems. In *Proceedings of the 6th International Conference on Informatics and Systems*, pages 12–17.

Fahim, A. and Hedar, A. (2014). Hybrid scatter search for integer programming problems. In *9th International Conference on Informatics and Systems*, pages 61–69.

Fleurent, C., Glover, F., Michelon, P., and Valli, Z. (1996). A scatter search approach for unconstrained continuous optimization. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 643–648.

García-López, F., Melián-Batista, B., Moreno-Pérez, J. A., and Moreno-Vega, J. M. (2003). Parallelization of the scatter search for the *p*-median problem. *Parallel Computing*, 29(5):575–589.

García López, F., García Torres, M., Melián Batista, B., Moreno Pérez, J. A., and Moreno-Vega, J. M. (2006). Solving feature subset selection problem by a parallel scatter search. *European Journal of Operational Research*, 169(2):477–489.

Gharehchopogh, F. S. and Amjad, S. (2019). A novel hybrid approach for email spam detection based on scatter search algorithm and *k*-nearest neighbors. *Journal of Advances in Computer Engineering and Technology*, 5(3):169–178.

Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166.

Glover, F. (1994). Tabu search for nonlinear and parametric optimization (with links to genetic algorithms). *Discrete Applied Mathematics*, 49(1–3):231–255.

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

Glover, F., Laguna, M., and Martí, R. (2000). Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 29(3):653–684.

Glover, F., Laguna, M., and Marti, R. (2003). Scatter search and path relinking: Advances and applications. *Handbook of Metaheuristics*, pages 1–35.

González, B. and Adenso-Díaz, B. (2006). A scatter search approach to the optimum disassembly sequence problem. *Computers & Operations Research*, 33:1776–1793.

Gortázar, F., Duarte, A., Laguna, M., and Martí, R. (2010). Black box scatter search for general classes of binary optimization problems. *Computers & Operations Research*, 37(11):1977–1986.

Hakli, H. and Ortacay, Z. (2019). An improved scatter search algorithm for the uncapacitated facility location problem. *Computers & Industrial Engineering*, 135:855–867.

Herrera, F., Lozano, M., and Molina, D. (2006). Continuous scatter search: An analysis of the integration of some combination methods and improvement strategies. *European Journal of Operational Research*, 169(2):450–476.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.

Hung, W. N. N., Song, X., Aboulhamid, E. M., and Driscoll, M. A. (2002). BDD minimization by scatter search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(8):974–979.

Hvattum, L. M., Duarte, A., Glover, F., and Martí, R. (2013). Designing effective improvement methods for scatter search: an experimental study on global optimization. *Soft Computing*, 17(1):49–62.

Ibáñez, O., Cordón, O., Damas, S., and Santamaría, J. (2012). An advanced scatter search design for skull-face overlay in craniofacial superimposition. *Expert Systems with Applications*, 39(1):1459–1473.

Jabal-Ameli, M. S. and Moshref-Javadi, M. (2014). Concurrent cell formation and layout design using scatter search. *The International Journal of Advanced Manufacturing Technology*, 71(1):1–22.

Kalra, M., Tyagi, S., Kumar, V., Kaur, M., Mashwani, W. K., Shah, H., and Shah, K. (2021). A comprehensive review on scatter search: Techniques, applications, and challenges. *Mathematical Problems in Engineering*, 2021:5588486.

Keskin, B. B. and Üster, H. (2007). A scatter search-based heuristic to locate capacitated transshipment points. *Computers & Operations Research*, 34(10):3112–3125.

Khooban, Z., Farahani, R. Z., Miandoabchi, E., and Szeto, W. Y. (2015). Mixed network design using hybrid scatter search. *European Journal of Operational Research*, 247(3):699–710.

Kothari, R. and Ghosh, D. (2014). A scatter search algorithm for the single row facility layout problem. *Journal of Heuristics*, 20(2):125–142.

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

Laguna, M. and Martí, R. (2003). *Scatter Search: Methodology and Implementations in C*, volume 24 of *Operations Research/Computer Science Interfaces Series*. Springer US, New York, NY.

Laguna, M. and Martí, R. (2003b). The OptQuest callable library. In *Optimization Software Class Libraries*, pages 193–218. Springer.

Laguna, M. and Martí, R. (2005). Experimental testing of advanced scatter search designs for global optimization of multimodal functions. *Journal of Global Optimization*, 33(2):235–255.

Laguna, M., Molina, J., Perez, F., Caballero, R., and Hernández-Díaz, A. G. (2010). The challenge of optimizing expensive black boxes: a scatter search/rough set theory approach. *Journal of the Operational Research Society*, 61(1):53–67.

Laguna, M., Gortázar, F., Gallego, M., Duarte, A., and Martí, R. (2014). A black-box scatter search for optimization problems with integer variables. *Journal of Global Optimization*, 58(3):497–516.

Laguna, M., Martí, R., Martínez-Gavara, A., Pérez-Peló, S., and Resende, M. G. C. (2025). Greedy randomized adaptive search procedures with path relinking: An analytical review of designs and implementations. *European Journal of Operational Research*, 327:717–734.

Li, K. and Tian, H. (2015). A DE-based scatter search for global optimization problems. *Discrete Dynamics in Nature and Society*, 2015:303125.

Liberatore, S. and Sechi, G. M. (2009). Location and calibration of valves in water distribution networks using a scatter-search meta-heuristic approach. *Water Resources Management*, 23(8):1479–1495.

Liu, F., Huang, H., Yang, Z., Hao, Z., and Wang, J. (2021). Search-based algorithm with scatter search strategy for automated test case generation of NLP toolkit. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(3):491–503.

Maenhout, B. and Vanhoucke, M. (2006). New computational results for the nurse scheduling problem: A scatter search algorithm. In Gottlieb, J. and Raidl, G. R. (Eds.), *Evolutionary Computation in Combinatorial Optimization*, pages 159–170. Springer, Berlin, Heidelberg.

Manikas, A. and Chang, Y.-L. (2009). A scatter search approach to sequence-dependent setup times job shop scheduling. *International Journal of Production Research*, 47(18):5217–5236.

Mansour, N., Isahakian, V., and Ghalayini, I. (2011). Scatter search technique for exam timetabling. *Applied Intelligence*, 34(2):299–310.

Mansour, N., Kehyayan, C., and Khachfe, H. (2009). Scatter search algorithm for protein structure prediction. *International Journal of Bioinformatics Research and Applications*, 5(5):501–515.

Martí, R., Laguna, M., and Glover, F. (2006). Principles of scatter search. *European Journal of Operational Research*, 169(2):359–372.

Martí, R. (2006). Scatter search—wellsprings and challenges. *European Journal of Operational Research*, 169(2):351–358.

Martí, R., Corberán, A., and Peiró, J. (2018). Scatter search. In *Handbook of Heuristics*, pages 717–740. Springer, Cham.

Martí, R., Laguardia, J., and León, M. T. (2025). Fundamentals of scatter search. In *Handbook of Heuristics (2nd edition)*, pages 599–626. Springer, Cham.

Martí, R., Laguna, M., and Campos, V. (2002). Scatter search vs. genetic algorithms: An experimental evaluation with permutation problems. In Rego, C. and Alidaee, B. (Eds.), *Adaptive Memory and Evolution: Tabu Search and Scatter Search*. Kluwer Academic Publishers.

Nebro, A. J., Luna, F., Alba, E., Dorronsoro, B., Durillo, J. J., and Beham, A. (2008). AbYSS: adapting scatter search to multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 12(4):439–457.

Pacheco, J. A. (2005). A scatter search approach for the minimum sum-of-squares clustering problem. *Computers & Operations Research*, 32(5):1325–1335.

Pantrigo, J. J., Sánchez, A., Montemayor, A. S., and Duarte, A. (2008). Multidimensional visual tracking using scatter search particle filter. *Pattern Recognition Letters*, 29(8):1160–1174.

Penas, D. R., González, P., Egea, J. A., Banga, J. R., and Doallo, R. (2015). Parallel metaheuristics in computational biology: An asynchronous cooperative enhanced scatter search method. *Procedia Computer Science*, 51:630–639.

Pendharkar, P. C. (2013). Scatter search based interactive multi-criteria optimization of fuzzy objectives for coal production planning. *Engineering Applications of Artificial Intelligence*, 26(5):1503–1511.

Piñol, H. and Beasley, J. E. (2006). Scatter search and bionomic algorithms for the aircraft landing problem. *European Journal of Operational Research*, 171(2):439–462.

Prabhaharan, G., Ramesh, R., and Asokan, P. (2007). Concurrent optimization of assembly tolerances for quality with position control using scatter search approach. *International Journal of Production Research*, 45(21):4959–4988.

Rahimi-Vahed, A. R., Rabbani, M., Tavakkoli-Moghaddam, R., Torabi, S. A., and Jolai, F. (2007). A multi-objective scatter search for a mixed-model assembly line sequencing problem. *Advanced Engineering Informatics*, 21(1):85–99.

Ranjbar, M., De Reyck, B., and Kianfar, F. (2009). A hybrid scatter search for the discrete time/resource trade-off problem in project scheduling. *European Journal of Operational Research*, 193(1):35–48.

Remil, M. A., Mohamad, M. S., Deris, S., Sinnott, R., and Napis, S. (2019). An improved scatter search algorithm for parameter estimation in large-scale kinetic models of biochemical systems. *Current Proteomics*, 16(5):427–438.

Ren, J. and Zhu, W. (2023). Backtracking search optimization algorithm with dual scatter search strategy for automated test case generation. *Journal of King Saud University – Computer and Information Sciences*, 35(7):101600.

Resende, M. G. C., Ribeiro, C. C., Glover, F., and Martí, R. (2010). Scatter search and path-relinking: Fundamentals, advances, and applications. *Handbook of Metaheuristics*, pages 87–107.

Russell, R. A. and Chiang, W.-C. (2006). Scatter search for the vehicle routing problem with time windows. *European Journal of Operational Research*, 169(2):606–622.

Sagarna, R. and Lozano, J. A. (2006). Scatter search in software testing, comparison and collaboration with estimation of distribution algorithms. *European Journal of Operational Research*, 169(2):392–412.

Santamaría, J., Cordón, O., Damas, S., Alemán, I., and Botella, M. (2007). A scatter search-based technique for pair-wise 3D range image registration in forensic anthropology. *Soft Computing*, 11(9):819–828.

Stützle, T. and Ruiz, R. (2018). Iterated greedy. *Handbook of Heuristics*, pages 547–577.

Tang, J., Zhang, J., and Pan, Z. (2010). A scatter search algorithm for solving vehicle routing problem with loading cost. *Expert Systems with Applications*, 37(6):4073–4083.

Ugray, Z., Lasdon, L., Plummer, J., Glover, F., Kelly, J., and Martí, R. (2007). Scatter search and local NLP solvers: A multistart framework for global optimization. *INFORMS Journal on Computing*, 19(3):328–340.

Valsecchi, A., Damas, S., Santamaría, J., and Marrakchi-Kacem, L. (2014). Intensity-based image registration using scatter search. *Artificial Intelligence in Medicine*, 60(3):151–163.

Vandenheede, L., Vanhoucke, M., and Maenhout, B. (2016). A scatter search for the extended resource renting problem. *International Journal of Production Research*, 54(16):4723–4743.

Vanhoucke, M. (2010). A scatter search heuristic for maximising the net present value of a resource-constrained project with fixed activity cash flows. *International Journal of Production Research*, 48(7):1983–2001.

Wang, J., Hedar, A.-R., Wang, S., and Ma, J. (2012). Rough set and scatter search metaheuristic based feature selection for credit scoring. *Expert Systems with Applications*, 39(6):6123–6128.

Wang, J., Zhang, Q., Abdel-Rahman, H., and Abdel-Monem, M. I. (2014). A rough set approach to feature selection based on scatter search metaheuristic. *Journal of Systems Science and Complexity*, 27(1):157–168.

Yamashita, D. S., Armentano, V. A., and Laguna, M. (2006). Scatter search for project scheduling with resource availability cost. *European Journal of Operational Research*, 169(2):623–637.

Yang, K., Zheng, F., Ji, Q., Lin, J., Zhong, Y., and Lin, Y. (2025). Heuristic-guided scatter search for X-architecture Steiner minimum tree problems in VLSI design. *Swarm and Evolutionary Computation*, 98:102088.

Zhang, T., Chaovalitwongse, W. A., and Zhang, Y. (2012). Scatter search for the stochastic travel-time VRP with simultaneous pick-ups and deliveries. *Computers & Operations Research*, 39(10):2277–2290.

Zhao, F., Zhou, G., Xu, T., Zhu, N., and Jonrinaldi (2023). A knowledge-driven cooperative scatter search algorithm with reinforcement learning for the distributed blocking flow shop scheduling problem. *Expert Systems with Applications*, 230:120571.

Zuo, Y., Zhao, F., and Zhang, J. (2025). A knowledge-driven scatter search algorithm for the distributed hybrid flow shop scheduling problem. *Engineering Applications of Artificial Intelligence*, 142:109915.