SORT

Statistics and Operations Research Transactions

Volume
50
Number 1, January-June 2026

Generalitat de Catalunya
Institut d'Estadística de Catalunya

SORT

Statistics and Operations Research Transactions

Volume 50, Number 1, January-June 2026

eISSN: 2013-8830

www.idescat.cat/sort/

Aims

SORT (*Statistics and Operations Research Transactions*) —formerly Qüestiió— is an international journal launched in 2003 and distributed in printed form as well as in digital form online. From 2024 SORT is published in digital form only. It is published twice-yearly by the Institut d'Estadística de Catalunya (Idescat), co-edited by the Universitat Politècnica de Catalunya, Universitat de Barcelona, Universitat Autònoma de Barcelona, Universitat de Girona, Universitat Pompeu Fabra, Universitat de Lleida i Universitat Rovira i Virgili and the cooperation of the Spanish Section of the International Biometric Society, the Catalan Statistical Society and the Departament de Recerca i Universitats, of the Generalitat de Catalunya. SORT promotes the publication of original articles of a methodological or applied nature or motivated by an applied problem in statistics, operations research, official statistics or biometrics as well as book reviews. We encourage authors to include an example of a real data set in their manuscripts. SORT is an Open Access journal which does not charge publication fees.

SORT is indexed and abstracted in the *Science Citation Index Expanded* and in the *Journal Citation Reports* (Clarivate Analytics) from January 2008 and also in *Diamond Discovery Hub* (DDH) from March 2026. The journal is also described in the *Encyclopedia of Statistical Sciences* and indexed as well by: *Current Index to Statistics*, *Índice Español de Ciencia y Tecnología*, *MathSci*, *Current Mathematical Publications and Mathematical Reviews*, and *Scopus*.

SORT represents the third series of the *Quaderns d'Estadística i Investigació Operativa* (*Qüestiió*), published by Idescat since 1992 until 2002, which in turn followed from the first series *Quaderns d'Estadística, Sistemes, Informàtica i Investigació Operativa* (1977-1992). The three series of *Qüestiió* have their origin in the *Cuadernos de Estadística Aplicada e Investigación Operativa*, published by the UPC till 1977.

Secretary

Institut d'Estadística de Catalunya (Idescat)
Via Laietana, 58
08003 Barcelona (Spain)
Tel. +34 - 93 557.30.76 - 93 557.30.00
E-mail: sort@idescat.cat

Publisher: Institut d'Estadística de Catalunya (Idescat)

eISSN: 2013-8830
DL B-46.085-1977
Key title: SORT
Numbering: 1 (december 1977)

www.idescat.cat/sort/

2025 CALIDAD REVISTAS CIENTÍFICAS ESPAÑOLAS
FECYT-073/2025
Fecha de certificación: 20 de mayo de 2011 (2ª edición)
Válido hasta: 19 de diciembre de 2027

eISSN: 2013-8830
SORT 50 (1) January-June (2026)

SORT

Statistics and Operations Research Transactions

Coediting institutions

Universitat Politècnica de Catalunya
Universitat de Barcelona
Universitat de Girona
Universitat Autònoma de Barcelona
Universitat Pompeu Fabra
Universitat de Lleida
Universitat Rovira i Virgili
Institut d'Estadística de Catalunya

Supporting institutions

Spanish Region of the International Biometric Society
Societat Catalana d'Estadística
Departament de Recerca i Universitats

Generalitat de Catalunya
Institut d'Estadística de Catalunya

SORT

Volume 50
Number 1
January-June 2026
eISSN: 2013-8830

Invited article

Articles

SORT 50 (1) January-June 2026, 3-42 DOI: 10.57645/20.8080.02.30

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

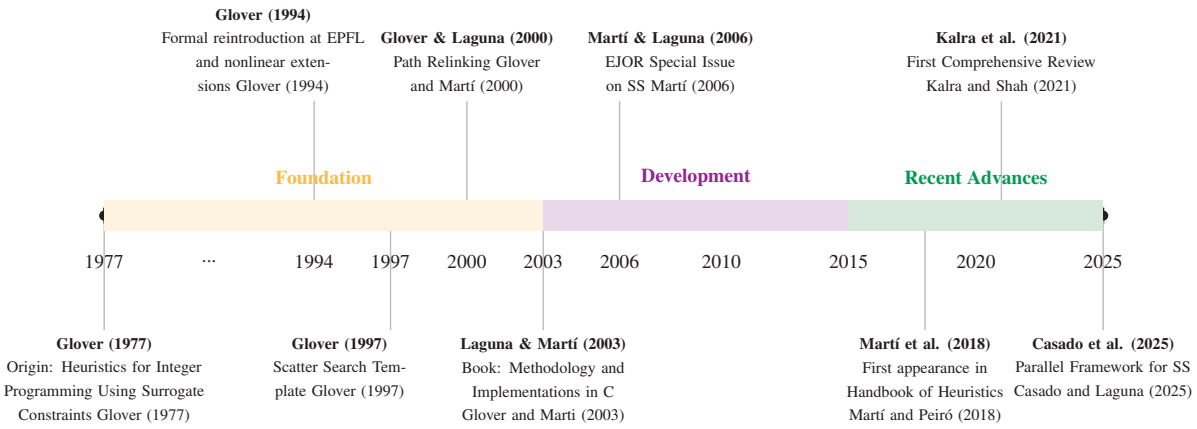

Figure 1. *SS timeline in three periods: Foundation, Development, and Recent Advances.*

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.

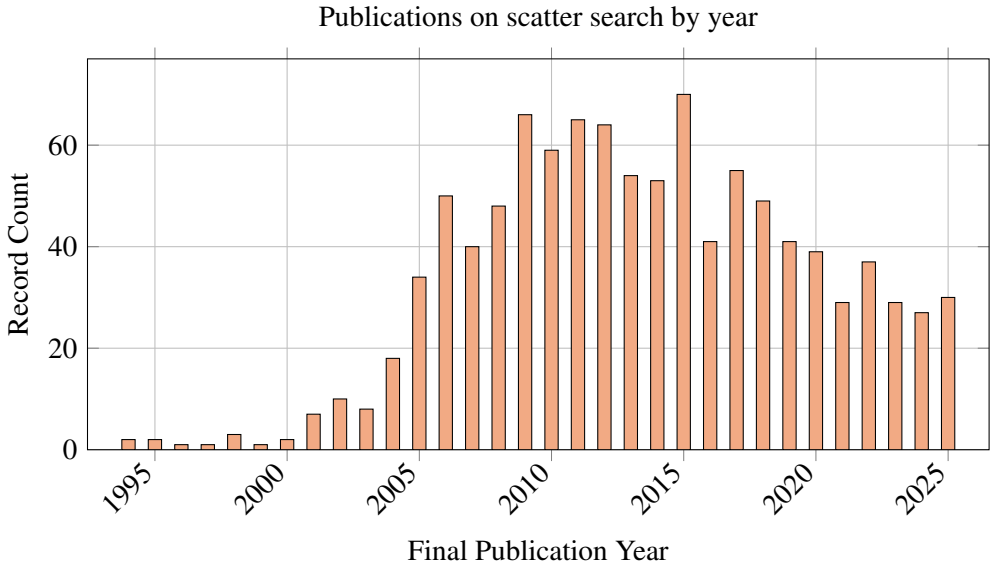

Figure 2. *Annual publication counts for scatter search–related works, based on Web of Science data.*

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.

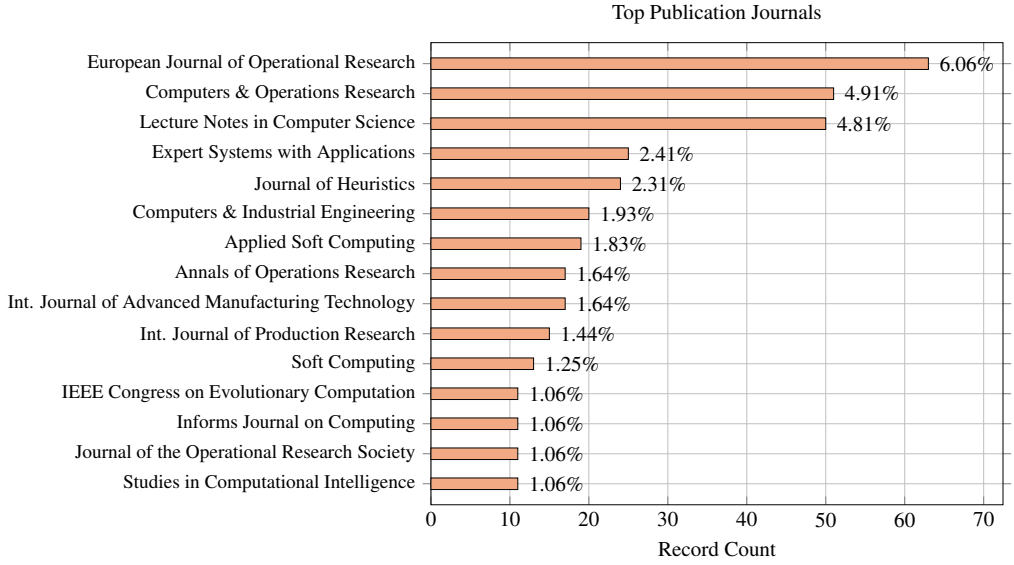

Figure 3. *Distribution of scatter search publications across the top 15 publication journals.*

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

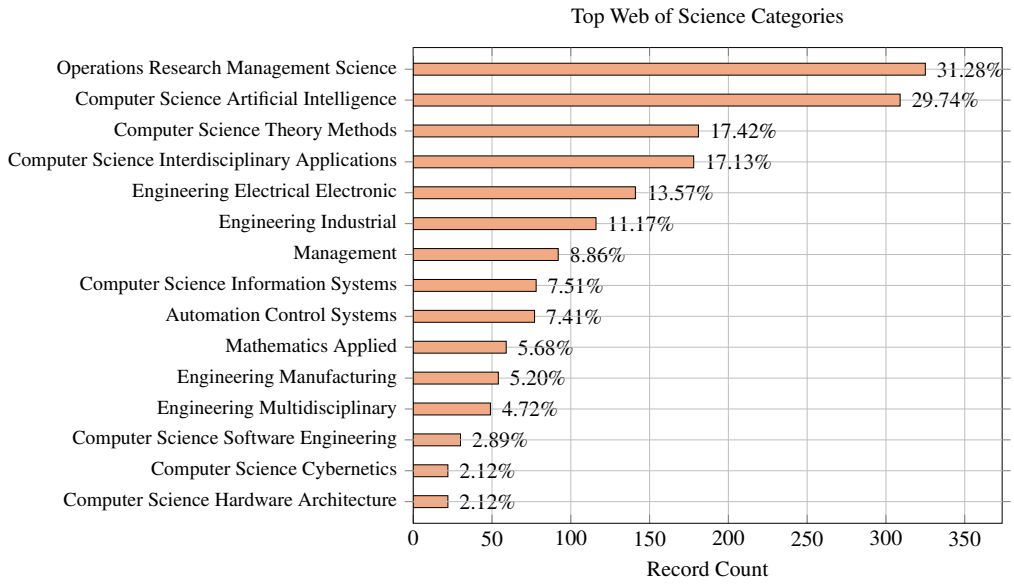

Figure 4. *Distribution of scatter search publications across the top 15 Web of Science Categories.*

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.

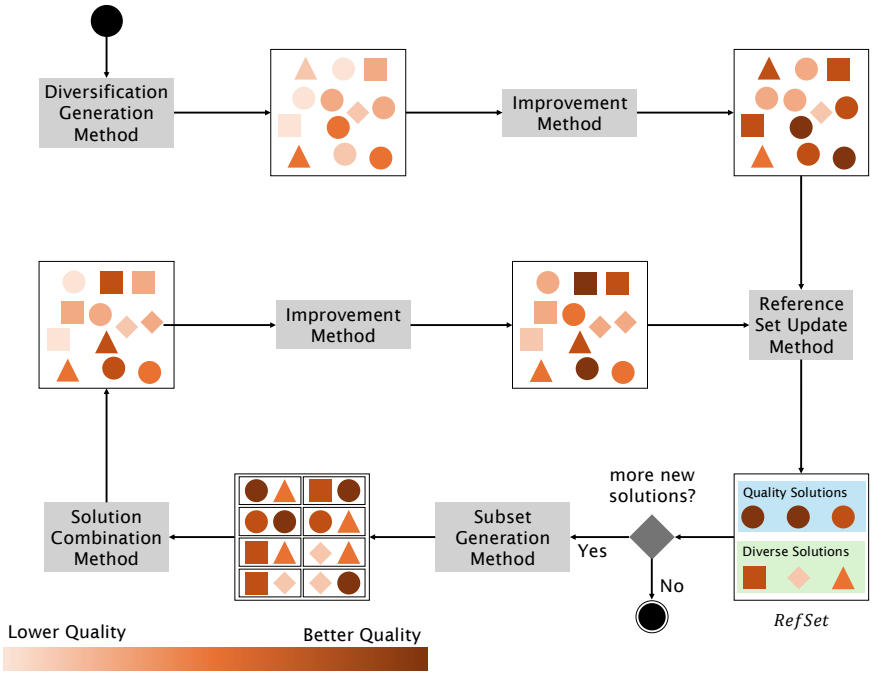

Figure 5. *SS illustration with the iterative RefSet update based on quality (color) and diversity (shape).*

Algorithm 1 Scatter search framework

Generate initial population P using the Diversification Generation Method
Apply the Improvement Method to each solution in P
Build initial *RefSet* from P (quality and diversity)
repeat
    Generate subsets from *RefSet* using the Subset Generation Method
    for all subsets S **do**
        Create new solutions by applying the Solution Combination Method
        Apply the Improvement Method to each new solution
        Evaluate new solutions and update *RefSet* if they qualify
    end for
until no new solutions are added to *RefSet*
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.

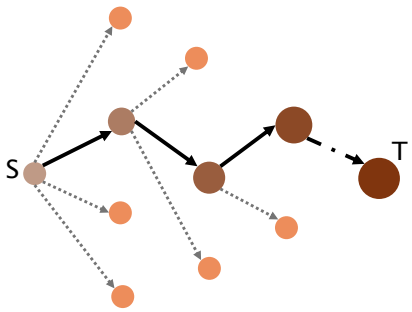

Figure 6. *Forward path relinking from an initial solution S to a guiding solution T.*

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.

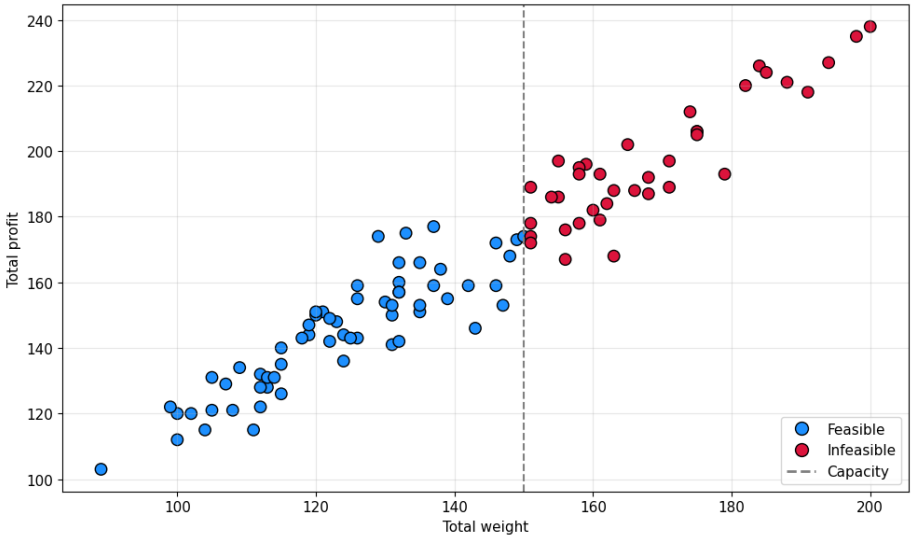

Figure 7. *Scatter plot of 100 diversified solutions. Dashed vertical line indicates knapsack capacity ($W = 150$).*

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).

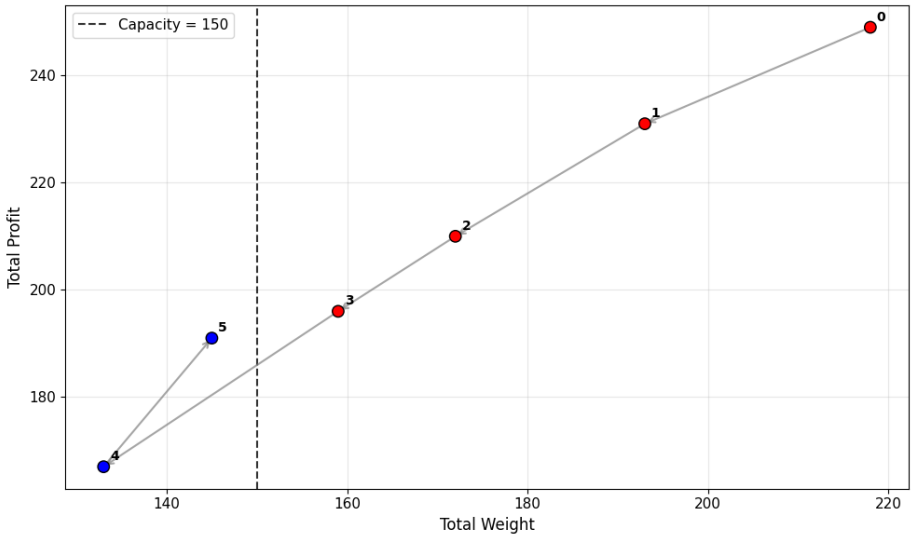

Figure 8. *Improvement trajectory in weight-value space.*

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.

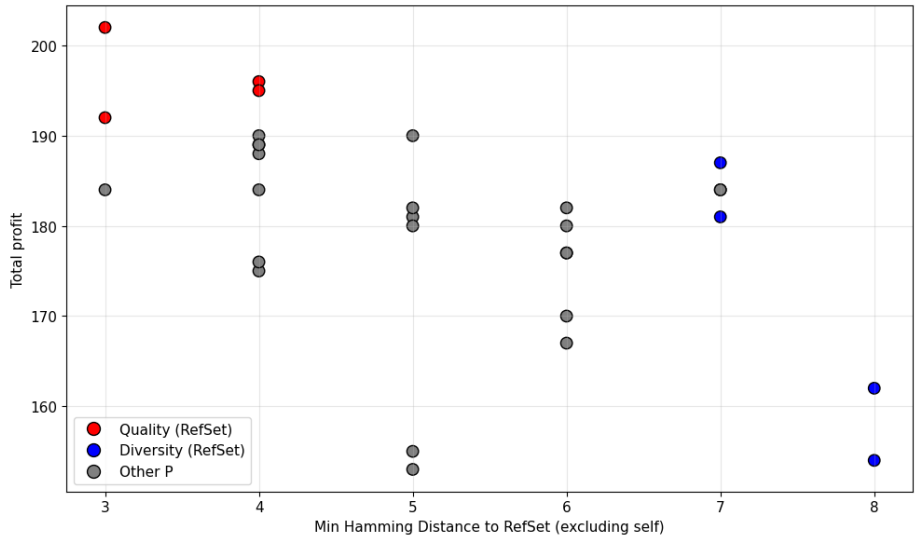

Figure 9. *Reference Set selection from pool P. Red: high-quality (b_1), Blue: diverse (b_2), Gray: non-selected. X-axis: minimum Hamming distance to RefSet, Y-axis: objective value.*

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)


```

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.

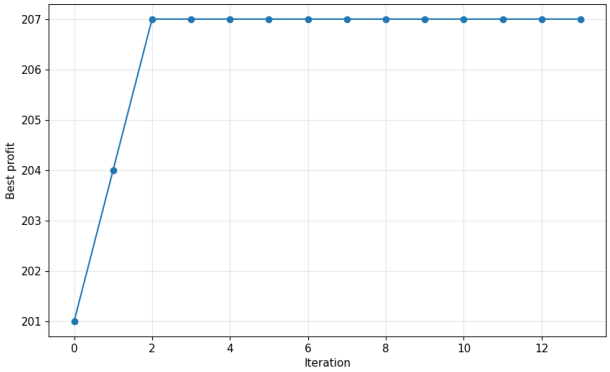

Figure 10. *Best solution value over iterations.*

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). Multi-dimensional 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.

SORT 50 (1) January-June 2026, 43-66 DOI: 10.57645/20.8080.02.31

Non-crossing neural network quantile regression estimation for driving data with telematics

Xenxo Vidal-Llana[1,*], Carlos Salort Sánchez[1], Vincenzo Coia[2] and Montserrat Guillén[3]

Abstract

State-of-the-art methods for estimating extreme quantiles (value at risk, VaR) and their tail expectations (conditional tail expectation, CTE) under covariate control primarily rely on quantile regression but lack explicit constraints for non-crossing conditions. To address this, we introduce the non-crossing dual neural network, a deep learning model that simultaneously estimates VaRs and CTEs across multiple quantile levels, incorporates covariate dependence, and enables the reconstruction of individual conditional distributions and risk profiles while ensuring the natural order of quantile levels. Using a 2015 telematics dataset, the proposed methodology outperforms benchmark models while enforcing previously unaddressed conditions. The model can be used to identify risk within an insurance portfolio and to analyze extreme right-tail behaviour at the individual level.

MSC: 68T07, 62G32, 62P05, 91G70.

Keywords: Risk evaluation, telematics, quantile regression, motor insurance, value at risk, conditional tail expectation.

1. Introduction

Quantile regression aims at modeling conditional quantiles of a response variable given some covariates and a fixed quantile level (Koenker and Bassett, 1978). This framework naturally allows the reconstruction of conditional distributions of the response by considering multiple quantile levels. In insurance, quantile regression is part of a general

* Corresponding author: juan.j.vidal@uv.es

[1] Universitat de València. Avinguda dels Tarongers. 46022 València, Spain.

[2] BGC Engineering Inc. Suite 500, 980 Howe St. Vancouver, BC Canada.

[3] Universitat de Barcelona. Gran Via de les Corts Catalanes 585. 08007 Barcelona, Spain.

Received: February 2025
Accepted: February 2026

framework known as risk regression, where the aim is to model a conditional risk measure of the response. Algorithms to pursue risk evaluation conditional on covariates are difficult to implement because of their inherent technical difficulties. The availability of large data sets allows predicting high quantiles under this regression framework and creates an environment where big losses can be evaluated given the fixed conditions surrounding the response (Guillen, Bermúdez, and Pitarque, 2021a). A natural extension of quantile regression is to model tail expectations. This enables a more granular characterization of risk at the individual level.

For financial studies, value at risk (VaR) appears as a common risk measure used by the analyst to evaluate extreme quantiles from the left part of the distribution. But, in insurance, it is common to evaluate quantiles from the right part of the distribution, as this literature focuses on predicting losses and its potential impact to the insurance company. The usage of quantile regression models has a long history (Koenker and Bassett, 1978), as it provides information on extreme quantiles where mean-based models are inadequate. Quantile regression for modeling VaR conditional on covariates appears in the financial literature (Bodnar, Hayt and Marston, 1998; Vidal-Llana and Guillén, 2022), but other fields use it as well, like weather forecasting (Cannon, 2018), flood prediction (Zhang, 2016) and car telematics data for telematics-based ratemaking (Guillen et al., 2020; Baione and Biancalana, 2021). In the latter field, the VaR has been especially used due to telematic information containing heavy tails (Pérez-Marín et al., 2019). When predicting losses in the insurance sector, estimators like the VaR help insurance firms manage exposure to extreme events, like earthquakes or massive crashes. Mean-based models assume an underlying normal distribution; therefore, they are not recommended for predicting asymmetric response outcomes, thus the importance of risk measurements such as VaR.

In recent years, some attention has been placed on the conditional tail expectation (CTE), since the Basel III committee (Basel Committee on Banking Supervision, 2016) promoted its use to evaluate risk on portfolios and reserves. The main motivation for CTE is the inclusion of more information about the tail than the VaR, as the VaR limits the estimation of the conditional quantile, while the CTE it estimates the expected beyond-quantile loss. But this change carries a complication, the elicitability or existence of a consistent scoring function that allows estimating the CTE (Gneiting, 2011). While the VaR is elicitable (Koenker and Bassett, 1978), CTE cannot be estimated directly, as it needs the VaR to be estimated beforehand. Fissler and Ziegel (2016) propose a consistent scoring function for the pair (VaR, CTE) which is a generalization of a particular case proposed before by Acerbi and Szekely (2014). Nolde and Ziegel (2017) compare traditional backtesting for elicitable estimators against a specification of the Fissler and Ziegel (2016) scoring function, concluding a better performance and flexibility from the latter one. Afterwards, some other specifications of this loss function were studied and performance across quantile levels and datasets was evaluated (cf. Taylor, 2019; Patton, Ziegel and Chen, 2019; Ziegel et al., 2020). A review of specifications of the general loss function and comparison with state-of-the-art methods for

time series datasets can be found in Taylor (2020). Despite this elicitability issue, there is plenty of fields that use CTE for evaluating risks on heavy tailed distributions. In finance, CTE is used for optimizing portfolios (Ciliberti, Kondor and Mézard, 2007), historical simulation of crises (Kourouma et al., 2010) or for regulation (Chen, 2018). In the energy field, Thaler, Grabec and Poredoš (2005) show the importance of forecasting energy consumption for predicting risk of energy demand and González-Pedraz, Moreno and Peña (2014) study the use of extreme estimators as the tail risk measures for energy portfolios that assume normality and underestimate the actual tail risk. Finally, classical insurance use cases for VaR and CTE are risk capital allocation (Asimit et al., 2011) and applications in predicting extreme losses in motor insurance portfolios (Frees, Shi and Valdez, 2009; Guillen et al., 2021a).

Despite VaR and CTE being popular risk measures in areas other than the insurance and financial industry, there exists an understudied issue: the crossing between the estimates. He (1997) and Yu, Lu and Stander (2003) established the issue of crossing quantiles when estimating several quantiles jointly. Namely, for each outcome variable y_i (i refers to the individual) and each pair of quantile levels $q_1,\ q_2,\ 0 < q_1 < q_2 < 1$, this literature seeks models to ensure that the order is preserved, namely that the quantile of y_i at level q_1 is less than or equal to the quantile of y_i at level q_2, $\text{VaR}_{q_1}(y_i) \leq \text{VaR}_{q_2}(y_i)$. For an authoritative treatment of this subject, see Takeuchi and Furuhashi (2004); Dette and Volgushev (2008); Bondell, Reich and Wang (2010); Chernozhukov, Fernández-Val and Galichon (2010); Liu and Wu (2011). More recent advances propose non-crossing quantile estimation using neural networks, which improve computational time and performance in comparison with other classical methods (cf. Cannon, 2018; Moon et al., 2021). The non-crossing condition that affects the VaR and its corresponding CTE, both assessed at the same quantile level, and which represents one of the ordering properties considered in this framework, was addressed by Acerbi and Szekely (2014), who added a constant inside the loss function, but in practice, finding that constant is non-trivial. Guillen et al. (2021a) solve this issue with a reparametrization of the CTE as a positive excess of the VaR that ensures the desired conditions but only for one quantile at a time. Recently, Fissler, Merz and Wüthrich (2023) proposed the deep composite regression architecture for the joint estimation of value at risk and conditional tail expectation above and below at a given quantile level, ensuring internal coherence between the three risk measures. These approaches focus on estimating VaR and CTE above and below at a single quantile level at a time and do not explicitly address ordering constraints when several quantile levels are considered simultaneously. In contrast, the method proposed in this paper aims to jointly estimate VaR and CTE across a set of quantile levels within a single neural-network framework, while enforcing all natural non-crossing conditions-across quantile levels and across risk measures-by construction. We propose a method that estimates both risk measures VaR and CTE for a set of quantile levels while asserting non-crossing conditions.

One of the areas where data are abundant and there is a need for conditional risk assessment is motor insurance. Driving data provide information to the insurer over time,

such as total distance driven or excess speed, and that information can be combined with static characteristics such as age, type of vehicle or main driving zone. This setting requires multivariate methods that incorporate covariates and are computationally feasible. When choosing quantile regression as the basic approach and evaluating the models separately at different quantile levels, we expect that quantiles estimated for a response are naturally ordered, so that, conditional on the same covariate values, a lower quantile level estimate is lower than a higher quantile level estimate. This is known as the *non-crossing condition* (He, 1997), but it does not necessarily hold in practice. Motor insurance telematics therefore provides a natural and practically relevant setting to illustrate how conditional risk measures can be used to reconstruct individual risk profiles in environments characterized by heterogeneity and extreme outcomes.

We present a neural network that estimates the conditional quantile, or VaR, and the expected loss above its quantile, the CTE, simultaneously and at different quantile levels while requesting that non-crossing conditions are satisfied, and covariates may be introduced. We refer to this proposed approach as the non-crossing dual neural network (NCDNN). By jointly estimating VaRs and CTEs across multiple quantile levels, the proposed framework allows the reconstruction of individual conditional distributions and associated risk profiles. The method shows competitive precision and reasonable computational time compared to other machine learning methods presented in the literature. We compare the results of the non-crossing model to a quantile regression model quantile by quantile (as in Guillen et al. (2021a)) and to the monotone composite quantile regression neural network (MCQRNN) (Cannon, 2018), using a telematics dataset from a Spanish insurance company with 9,614 drivers from 2015, where we evaluate quantile levels 0.5, 0.6, 0.7, 0.8, 0.9, 0.925, 0.95, 0.975 and 0.99. We fit the VaRs and the corresponding CTEs.

This paper is organized as follows. First, Section 2 presents the definitions for value at risk, conditional tail expectation, quantile regression and the non-crossing dual neural network. Section 3 provides a description of the dataset and explains the distribution of the response and covariates used. Section 4 shows the main results, with comparisons between the estimation methods in terms of model performance, and a method to estimate individual conditional distributions of a driver's behaviour. Section 6 concludes our work.

2. Methodology

In this section we explicitly write the notation first and then we present the new proposed model. In insurance we define losses as positive real numbers because we refer to losses as amounts that will be compensated to policyholders. So, the risk is present in the right tail, whereas in finance losses are usually expressed as negative returns and the risk is located on the left tail. This sign convention differs across fields, so we state it explicitly for clarity.

We will assume that losses are positive in the rest of this work and so, that our response variable is positive, as discussed in previous related works (Nolde and Ziegel, 2017).

2.1. Value at risk and quantile regression

The VaR at level q for a random variable Y, $\text{VaR}_q(Y)$, is defined as the magnitude of exceedances of the underlying distribution with probability q, $0 < q < 1$. Mathematically,

$$\text{VaR}_q(Y) = \inf\{y \in \mathbb{R} | F_Y(y) > q\} = F_Y^{-1}(1-q)\,, \tag{1}$$

where F_Y is the distribution function of Y.

For predicting VaR we take use of quantile regression, an extension of linear regression whose objective is to fit the quantile of the response variable for a specified level using a set of k covariates (Koenker and Bassett, 1978; Koenker, 2017). Denoting X as the covariate vector, $T(X) = \{X_1, X_2, \ldots, X_k\}$, where the $T(X)$ denotes the transposed vector, the conditional quantile function can be written as a linear combination of the covariates as

$$Q_{Y|X}(q) = \text{VaR}_{(1-q)}(Y|X) = \beta_{(q)0} + \beta_{(q)1}X_1 + \beta_{(q)2}X_2 + \cdots + \beta_{(q)k}X_k = T(X)\beta_{(q)}\,, \tag{2}$$

with parameter vector $\beta_{(q)}$ corresponding to $\arg\min_{\beta} \mathbb{E}[\rho_q(T(X)\beta, Y)]$, and ρ being the scoring function proposed by Koenker and Bassett (1978):

$$\rho_q(r,y) = (q - \mathbb{1}_{\{y-r<0\}})(y-r)\,, \tag{3}$$

where r is the quantile prediction, y the observed random variable Y, and $\mathbb{1}_{\{.\}}$ is the indicator function, with a value equal to 1 when the subscript is true and 0 otherwise.

In settings characterized by asymmetric response distributions with heavy right tails, classical generalized linear models, which focus on modeling the conditional mean, may provide an incomplete representation of risk and can lead to underpricing of extreme outcomes in insurance applications. By contrast, quantile regression allows for the direct modeling of different parts of the conditional distribution, making it better suited to capture tail behaviour and heterogeneity across risk profiles

2.2. Conditional tail expectation and scoring functions

The CTE for a level q, $0 < q < 1$, is defined as:

$$\text{CTE}_q(Y) = \mathbb{E}[Y | Y \geq \text{VaR}_q(Y)]\,. \tag{4}$$

The CTE, also known as the expected shortfall (ES), tail conditional expectation (TCE), or tail value at risk (TVaR), is a risk measure that evaluates the expected loss conditioned on exceeding its corresponding VaR.

As mentioned above, it is common to assume only positive (or negative) values for the random variable Y, for example, Fissler and Ziegel (2016) define the scoring function for the pair (VaR, CTE) for the negative part of the tail, aiming to low quantile levels, but as for this work, we are interested in the positive part. So, we will use the specification as in Nolde and Ziegel (2017):

$$\begin{aligned} S_q(r_1, r_2, y) = \mathbb{1}_{\{y>r_1\}}\big(-G_1(r_1) + G_1(y) - G_2(r_2)(r_1 - y)\big) + \\ (1-q)\big(G_1(r_1) - G_2(r_2)(r_2 - r_1) + \mathcal{G}_2(r_2)\big)\,, \end{aligned} \tag{5}$$

where r_1 represents the VaR_q, r_2 represents the CTE_q, and $G_1,\ G_2,\ \mathcal{G}_2 : \mathbb{R} \to \mathbb{R}$ are real functions. To ensure consistency as a scoring function for the pair $(\text{VaR}_q,\ \text{CTE}_q)$, Fissler and Ziegel (2016) prove that G_1 must be an increasing function, $\mathcal{G}_2$ must be increasing and concave, and $\mathcal{G}_2' = G_2$. It is important to note that this definition holds for absolutely continuous distributions.

There are several options proposed in the literature for choosing G_1 and G_2. Dimitriadis and Bayer (2019) propose two options for G_1 and five options for G_2 and compare their performance, showing minimal differences between some of them. Taylor (2020) summarizes previously proposed combinations in more depth. We will use $G_1(x) = 0$ and $G_2(x) = 1/x$ for simplicity, which renders the following scoring function:

$$S_q^{AL}(r_1, r_2, y) = \mathbb{1}_{\{y>r_1\}}\frac{y - r_1}{r_2} + (1-q)\left(\frac{r_1 - r_2}{r_2} + \log(r_2)\right)\,. \tag{6}$$

The scoring function represents the negative log-likelihood of an Asymmetric Laplace, hence its superscript AL (Taylor, 2019).

Although there is evidence to suggest that some scoring functions are more appropriate than others (Dimitriadis and Bayer, 2019) and that alternative specifications of the functions G_1 and G_2 may yield slightly improved predictive performance, such improvements often come at the cost of a substantial increase in computational time (Vidal-Llana, Coia and Guillen, 2022). In light of these trade-offs, we will retain the study to this scoring functions specification for simplicity.

2.3. The non-crossing condition and the non-crossing dual neural network

Empirical applications have shown that crossings between VaRs are possible. He (1997) and Yu et al. (2003) studied that issue. The fact that a VaR could possibly exceed its corresponding CTE was analyzed by (Acerbi and Szekely, 2014; Guillen et al., 2021a). While certain non-crossing conditions related to VaR or the relationship between VaR and CTE have been previously discussed in the literature, to the best of our knowledge a joint formulation imposing coherent non-crossing constraints simultaneously on VaRs and CTEs across multiple quantile levels has not been explicitly stated. We define the non-crossing conditions for VaR and CTE across several quantile levels with the following property.

Property 1. *For each observation y_i, $i \in \{1,\dots,N\}$, and a pair of consecutive quantile levels from an increasing succession $\{q_j\}_{j=0}^{J}$, $J > 0$, $0 < q_j < 1$ $\forall j$, the (VaR, CTE) predictions ensure non-crossing properties if, $\forall i$:*

1. $VaR_{q_j}(y_i) \leq VaR_{q_{j+1}}(y_i)$,
2. $VaR_{q_j}(y_i) \leq CTE_{q_j}(y_i)$,
3. $CTE_{q_j}(y_i) \leq CTE_{q_{j+1}}(y_i)$.

so, a model that jointly estimates VaR and CTE with a consistent scoring function of the form of (5), while asserting Property 1 produces predictions satisfying the non-crossing property. The model that we present in this study that assures the non-crossing property is the non-crossing dual neural network (NCDNN), a deep learning model that creates an environment in which VaR and CTE predictions maintain the natural ordering of their corresponding quantile levels.

Deep learning is a family of machine learning models based on neural networks (LeCun, Bengio and Hinton, 2015). These models have been used successfully in a variety of fields, including computer vision (Voulodimos et al., 2018) and natural language processing (Vaswani et al., 2017). The architecture of a basic neural network, also called fully connected, consists of L "fully connected" layers. Each layer $l \in \{1,\dots,L\}$, with dimension $h_l > 0$, is defined as:

$$x_{l+1} = Y_l(x_l) = \sigma(W_l \cdot x_l + b_l) \tag{7}$$

where $x_l \in \mathbb{R}^{h_l}$ is the $l-1$-layer output, x_1 is the input vector of covariates ($h_1 = k$), and x_L denotes the output of the final layer of the network, representing the J predicted quantile levels. $W_l \in \mathbb{R}^{h_{l+1} \times h_l}$ denotes the weights matrix, $b_l \in \mathbb{R}^{h_l}$ corresponds to the bias vector and $\sigma(\cdot)$ is a non-linear function.

The model learning process can be roughly explained in two steps. 1) Forward step: The input data passes through the layers where, at each layer, Equation (7) is evaluated, i.e. the output of the previous layer is the input for the current one. 2) Backward step: The scoring function is evaluated on the neural network's output and weights are updated by computing a conjugate gradient variation (backpropagation). Neural networks are particularly well suited for risk estimation problems due to their ability to model complex nonlinear relationships between covariates and outputs, while allowing structural constraints to be incorporated directly into the architecture. In this work, this flexibility is exploited to jointly estimate multiple risk measures across several quantile levels and to enforce non-crossing conditions by construction through the network design.

Our proposed model, the NCDNN, consists of a deep learning model that jointly predicts VaR and CTE for several quantile levels while asserting the non-crossing conditions defined in Property 1. These conditions are met with a model architecture that consists of two fully connected neural networks, one for calculating VaRs and another for calculating CTEs, that estimate consecutive VaR and CTE as exceedances of previous predictions, thus ensuring the natural ordering of the resulting estimates. Many

authors propose the calculation of each VaR as an excess shift from the previous one (cf. Granger, 2010; Schmidt and Zhu, 2016; Catania and Luati, 2022) or estimate VaR and its corresponding CTE above and below (Fissler et al., 2023). Another way to proceed is to use neural networks to predict different quantile levels without crossing (cf. Cannon, 2018; Moon et al., 2021) and this architecture allows incorporating CTE estimation across several quantile levels while enforcing ordering constraints.

For illustrating the definition of the NCDNN, Figure 1 illustrates the distribution of an observed variable together with predictions of VaR and CTE for two quantile levels. We present two possible scenarios. We estimate directly using (3) the VaR for the first quantile level (VaR_{q_0}), and we calculate the second VaR (VaR_{q_1}) and the first CTE (CTE_{q_0}) separately as exceedances of this first VaR. Before computing the next CTE (CTE_{q_1}), two possibilities arise: $\text{VaR}_{q_1} < \text{CTE}_{q_0}$ or the opposite. Both possibilities are represented in the left and right panels of Figure 1, respectively. To ensure non-crossing conditions, we calculate the second CTE (CTE_{q_1}) as an excess of the maximum between its corresponding VaR (VaR_{q_1}) and the previous CTE (CTE_{q_0}). Following this scheme, the non-crossing property holds over a grid of quantile levels. We will show that, under the NCDNN architecture, Property 1 holds.

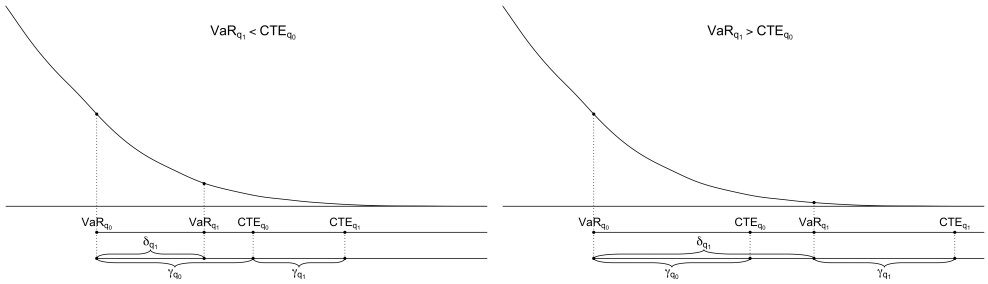

Figure 1. *Representation of NCDNN model for motivation purposes for quantile levels q_0, q_1, $q_0 < q_1$.*

Lemma 1. *Given $\{q_j\}_{j=0}^{J}$, an ordered set of quantile levels, $J > 0$, $0 < q_j < 1\ \forall j$, and let δ_{q_j}, $\gamma_{q_j} > 0$, be the outputs of a neural network. We define:*

$$\begin{aligned} VaR_{q_{j+1}} &:= VaR_{q_j} + \delta_{q_{j+1}}\ , \\ CTE_{q_{j+1}} &:= VaR_{q_{j+1}} + max(0, CTE_{q_j} - VaR_{q_{j+1}}) + \gamma_{q_{j+1}}\ . \end{aligned}$$

Then the estimates for VaR and CTE using a strictly consistent scoring function for this pair are non-crossing, i.e., they meet Property 1.

Proof.

1.) $\text{VaR}_{q_j}(y_i) \leq \text{VaR}_{q_{j+1}}(y_i)$

$$\text{VaR}_{q_j+1} = \text{VaR}_{q_j} + \delta_{q_{j+1}} \geq \text{VaR}_{q_j} + 0$$

2.) $\text{VaR}_{q_j}(y_i) \leq \text{CTE}_{q_j}(y_i)$

$$\text{CTE}_{q_j} = \text{VaR}_{q_j} + max(0, \text{CTE}_{q_{j-1}} - \text{VaR}_{q_j}) + \gamma_{q_j} \geq \text{VaR}_{q_j} + 0 + 0$$

3.) $\text{CTE}_{q_j}(y_i) \leq \text{CTE}_{q_{j+1}}(y_i)$

Case 1: $\text{VaR}_{q_{j+1}} \geq \text{CTE}_{q_j}$

$$\text{CTE}_{q_{j+1}} = \text{VaR}_{q_{j+1}} + 0 + \gamma_{q_{j+1}} \overbrace{\geq}^{Case\ 1\ condition} \text{CTE}_{q_j} + \gamma_{q_{j+1}} \geq \text{CTE}_{q_j} + 0$$

Case 2: $\text{VaR}_{q_{j+1}} < \text{CTE}_{q_j}$

$$\text{CTE}_{q_{j+1}} = \cancel{\text{VaR}_{q_{j+1}}} + \text{CTE}_{q_j} - \cancel{\text{VaR}_{q_{j+1}}} + \gamma_{q_{j+1}} = \text{CTE}_{q_j} + \gamma_{q_{j+1}} \geq \text{CTE}_{q_j} + 0$$

■

We omit the random variable in the notation, but every calculation is done elementwise, so Property 1 is ensured to all observations. We name δ_{q_j} and γ_{q_j} the outputs of the neural networks, which represent the exceedances from previous values after having applied a sigmoid activation function to assure positive values. The architectural representation of the model is found in Figure 2.

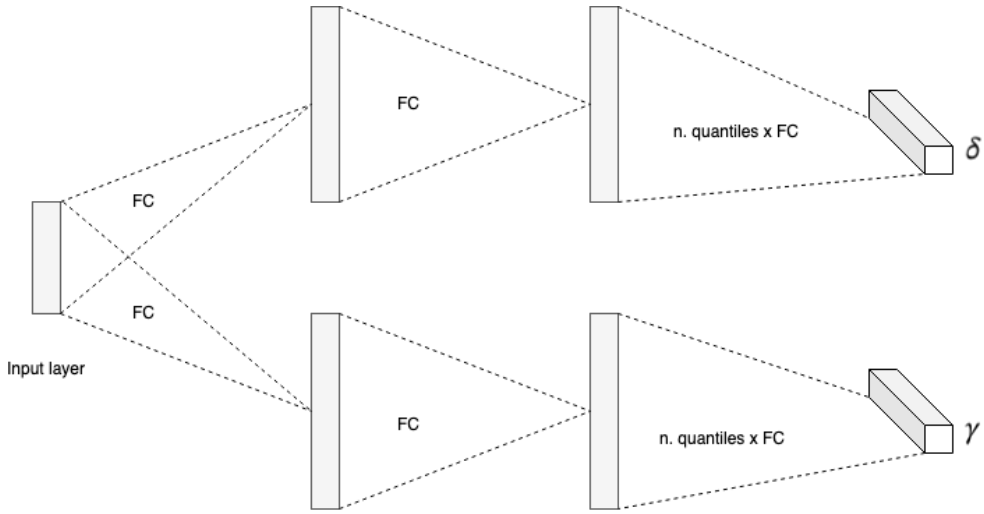

Figure 2. *Architectural diagrams of NCDNN. FC corresponds to a fully connected layer. The last layer is composed of as many FC layers as quantiles are calculated simultaneously.*

For comparison purposes, we are going to show results of the NCDNN compared to a linear quantile regression for the VaR following the same structure as (2) for each quantile level and we optimize a linear model for predicting each corresponding CTE

as an excess. This model structure has been successfully implemented in Acerbi and Szekely (2014); Guillen et al. (2021a). The CTE is defined as:

$$CTE_{Y|X}(q) = VaR_q(Y) + \gamma_{(q)0} + \gamma_{(q)1}X_1 + \gamma_{(q)2}X_2 + \cdots + \gamma_{(q)k}X_k = VaR_q(Y) + T(X)\gamma_{(q)}\,, \tag{8}$$

by using the scoring function (6) for optimizing $\gamma_{(q)}$ and using the previously estimated VaR. This process is repeated for each quantile level, so that predictions meet condition 2 of the non-crossing Property 1 ($VaR_{q_i} < CTE_{q_i}\ \forall i$) as we remove negative values during the estimation process, but the other two non-crossing conditions ($VaR_{q_i} < VaR_{q_{i+1}}\ \forall i$ and $CTE_{q_i} < CTE_{q_{i+1}}\ \forall i$) are not necessarily going to hold and cannot be corrected unless a comprehensive approach is introduced. The code for the NCDNN is publicly available to enable reproducibility of the results and application to other datasets at the following GitHub repository (https://github.com/JuanJoseVidal/ncdnn).

While monotonicity mechanisms based on cumulative positive increments have been previously used in quantile regression and neural network settings to prevent crossings across quantile levels, the contribution of this work does not lie in the monotonicity mechanism itself. Rather, the novelty of the proposed approach is the joint estimation of VaR and CTE over multiple quantile levels within a single neural network architecture, where all natural ordering relationships-across quantile levels and across risk measures-are simultaneously enforced by construction. To the best of our knowledge, existing approaches either address quantile crossing alone or treat VaR and CTE separately, without guaranteeing all non-crossing conditions in a unified framework.

We present the results obtained from our proposed model, the NCDNN, and compare them to two models: 1) a quantile regression estimation results for VaR and the subsequent optimization that leads to the estimation of the CTE one quantile level at a time. This two-step approach is called in this work QR+CTE, and it is implemented in the same way as Guillen et al. (2021a), which assures non-crossing between each VaR and its corresponding CTE (only point 2 of Property 1), but not between different VaRs nor CTEs of different levels. And 2), to the monotone composite quantile regression neural network (Cannon, 2018) for estimating the different VaR and estimating the CTE as averages of all posterior VaRs. We center our study on quantile levels 0.5, 0.6, 0.7, 0.8, 0.9, 0.925, 0.95, 0.975 and 0.99. The particular neural network used in this work consists of 5 neurons in the input layer, corresponding to the number of covariates, and 64 neurons in each fully connected hidden layer, following the architecture presented in Figure 2. Finally, the output layer has a number of neurons equal to the number of quantiles, where the network output represents the increments between the estimated VaRs and CTEs. The stopping criteria used is training was terminated early using a patience-based criterion: if the CTE loss component did not exhibit a strictly lower value than its previously observed minimum for 100 consecutive epochs, optimization was halted and the model parameters from the last epoch were returned. For the three models, we trained 50 different seeds for each model and chose the one that best performs on average for all quantile levels. The best performing selection criteria is based on the average of the normalized errors across quantiles, meaning that the errors for each quantile across

seeds are divided by the maximum, and then for each seed the average is performed. We select the seed with the lower one. Mean and standard deviation of the loss across quantile levels are reported in the Appendix in Table 3, indicating stable performance of the proposed NCDNN under stochastic optimization. All models use scoring functions (3) and (6) for estimating VaR and CTE, respectively, and were trained on a MacBook Pro (13-inch, M1, 2020, 8GB RAM); average training times per random seed were 0.71 minutes for the QR+CTE model, 35.76 minutes for MCQRNN, and 4.10 minutes for the proposed NCDNN.

3. Telematics data from a spanish insurance company

This illustration takes a sample of 9,614 drivers' information from a spanish insurance company consisting of four aggregated factors about their driving behaviour during the year 2015 and two personal factors. Although the data presents annual observations, the same exercise can be applied to monthly, weekly, or even daily observations for creating a more dynamic approach, but putting in risk the consistency of the study, as more granular data implies more volatile information.

The variables used in the study are the percentage of kilometers driven above speed limit (*% Above speed limit*) as the response variable, the percentage of night driving (*% Night*), the logarithm of kilometers driven (*log(Km)*) and the percentage of urban driving (*% Urban*) as telematic factors. Other factors used but only related to personal driver's information are the age (*Age*) and gender (*Gender*). The usage of percentage of kilometers above the speed limit is motivated by the relation of this factor with claim frequency (Pérez-Marín et al., 2019; Guillen et al., 2021a, Guillen, Pérez-Marín and Alcañiz, 2021b).

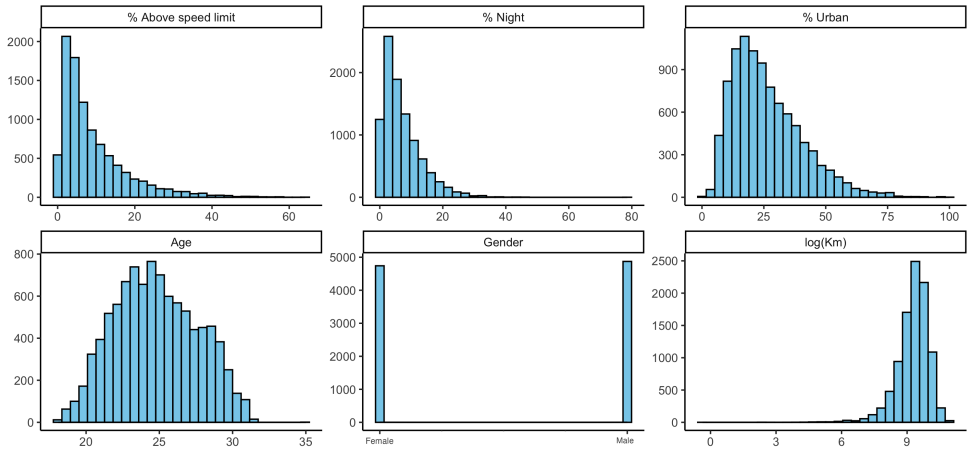

Figure 3. *Distributions of the response variable (%* Above speed limit*) and covariates of the 2015 recollection of telematic information from a spanish insurance company. The total number of drivers is 9,614 drivers in this study.*

Figure 3 shows the distribution of the variables used in the empirical analysis. We observe that the response variable (*% above speed limit*) presents a heavy right tail. Speed violations are directly related to claim frequency, therefore the extreme observations located on the right part of the tail represent a large loss to an insurance company, which motivates our study of extreme quantiles. Other factors also present heavy tails, both percentage or urban driving and percentage of night driving show right tails, and the logarithm of kilometers to the left (which would be right tailed without the logarithm transformation). The average driver on this dataset, by looking at the median values, would be a 25-year-old male, that drives around 11,700 kilometers per year, which are 23.4% on urban areas, 5.3% during the night and 6% above the speed limit.

4. Results

We observe the number of crossings for each pair of consecutive quantile levels and by type of estimator (between consecutive VaRs and consecutive CTEs) in Table 1. This table corresponds to the predictions using the quantile regression for VaR and CTE optimization, one quantile level at a time (QR+CTE). We did not include the table generated by the other two models (MCQRNN and NCDNN) as it is a table where all entries are equal to zero.

4.1. Crossings

Table 1. *Number of crossings for the quantile regression and CTE estimation one quantile level at a time (QR+CTE) (in parenthesis the percentage of observations against the whole sample, 9,614 observations).*

q_i - q_{i+1}	0.5 - 0.6	0.6 - 0.7	0.7 - 0.8	0.8 - 0.9	0.9 - 0.925	0.925 - 0.95	0.95 - 0-975	0.975 - 0.99
$VaR_{q_i} > VaR_{q_{i+1}}$	47 (0.5%)	29 (0.3%)	13 (0.1%)	6 (0.1%)	3 (0.0%)	1 (0.0%)	2 (0.0%)	2 (0.0%)
$CTE_{q_i} > CTE_{q_{i+1}}$	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	265 (2.8%)	33 (0.3%)	14 (0.1%)	764 (7.9%)

Table 1 shows the number of crossings obtained in the illustration data set by a model that considers one quantile level at a time. The first row shows a small number of crossings compared to the second one, 47 cases at most, 0.5% of the dataset. Nonetheless, we must remind that, for example, a value equal to 3 when comparing $VaR_{0.9}$ with $VaR_{0.925}$ means that three observations have been estimated with a $VaR_{0.9}$ greater than a $VaR_{0.925}$, which would be considered unfeasible estimations as they would not follow a natural ordering. We note that those are the type of estimation approaches based on quantile regression that are regularly used nowadays for estimating VaRs for financial firms and insurance companies. If we look at the second row of Table 1, the crossing issue is even more obvious, as CTE crossings reaches 16% for quantile levels 0.95 and 0.975, and although the scoring functions used for estimating CTE are strictly consistent, the

crossing issue must be considered during the model training process by using models like the NCDNN.

As an illustrative example, we consider an observation in which the QR+CTE model has a crossing between $CTE_{0.95}$ and $CTE_{0.99}$, which are two common quantile levels studied in financial and insurance reserving literature. We have an observation that corresponds to a 20-year-old male, that drove 6,800 kilometers, a 75% on urban areas, 3.7% night driving and only 1% above speed limits. For this particular observation, we plot in Figure 4 the predictions for the VaR and CTE for quantile levels 0.95 and 0.99 for both models, the QR+CTE and the NCDNN.

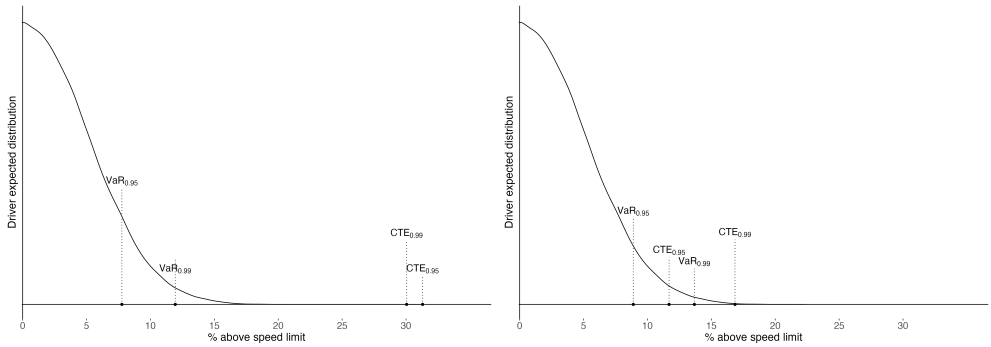

Figure 4. *VaR and CTE predictions for observation 26 of the dataset used in this study, for quantile levels 0.95 and 0.99 and for models QR+CTE (left) and NCDNN (right).*

We observe in Figure 4 how the QR+CTE model predicts a $CTE_{0.95}$ greater than the $CTE_{0.99}$ (left panel), which makes this prediction unfeasible for a direct application into pricing. Also, QR+CTE predictions are more spread than the ones provided by the NCDNN, and apparently the difference between estimates when different quantile levels are considered at the same time, show some difference between the two models. Quantile regression results are accumulated around value 10 and the CTE estimates are located around value 30, while the NCDNN is capable of finding more patterns of these extreme quantile levels and escape from inherited linear predictors that seem to be usual for the traditional QR+CTE models. We argue that this may be due to the non-linear structure that the neural networks are capable of adopting. We also calculated the scoring function (6) for CTE predictions for both quantile levels and for both models to show which is the improvement of changing from a model that admits crossing (QR+CTE) to a model that does not allow it (NCDNN). For quantile level 0.95 the scoring function changes from a 0.13 to a 0.11 (-17% loss), and for quantile level 0.99 the scores change from a 0.028 to a 0.026 (-6% loss). More information about the scoring results across the whole dataset is provided in Section 4.2.

4.2. Model performance

We quantify the improvement on performance of the NCDNN against the QR+CTE and the MCQRNN using scoring functions ρ_q and S_q^{AL} defined in (3) and (6). Results for VaR and CTE scorings are presented in Table 2.

Table 2. *Sum of scoring values for estimated VaR and CTE over the whole dataset for the three models: quantile regression for VaR and CTE optimized one quantile level at a time (QR+CTE), monotone composite quantile regression neural network (MCQRNN), and non-crossing dual neural network (NCDNN), across quantile levels 0.5 to 0.99. Best-performing models for each measure and quantile level are shown in bold.*

	Model	0.5	0.6	0.7	0.8	0.9	0.925	0.95	0.975	0.99
VaR	QR+CTE	27154.01	28709.86	28277.52	25136.43	17671.66	14704.90	11128.86	6631.66	3189.81
	MCQRNN	**26802.60**	28261.37	27719.30	24477.65	16942.51	14083.83	10598.19	6283.45	3010.47
	NCDNN	26802.62	**28257.24**	**27674.48**	**24408.12**	**16911.10**	**14063.51**	**10589.10**	**6269.75**	**3009.66**
CTE	QR+CTE	12734.08	10642.63	8371.30	5893.78	3158.79	2422.24	1658.31	862.27	357.78
	MCQRNN	12777.47	10632.43	8323.67	5832.35	3111.43	2385.44	1632.56	848.20	354.45
	NCDNN	**12604.96**	**10526.57**	**8269.67**	**5815.07**	**3109.02**	**2383.84**	**1631.62**	**846.76**	**352.85**

Observing Table 2, consistently for all quantile levels, and for VaR and CTE, NCDNN overperforms the QR+CTE model, ranging from a 1.3% loss improvement in VaR for median quantile levels to 6% loss improvement in VaR for extreme quantile levels. The improvement in CTE ranges between 1% and 2%. On top of that, QR+CTE model does not follow natural ordering of the estimated quantiles as seen in Table 1, while the other two models meet the non-crossing property.

Regarding the MCQRNN, we observe a lower improvement than with the QR+CTE when comparing to our proposed model. Table 2 shows that only for quantile level 0.5, the MCQRNN overperforms the NCDNN, but for extreme quantile levels, levels that are more of the interest of a risk analyst, the NCDNN situates as the best model, reducing losses to a 0.3% in comparison to the MCQRNN. Regarding the scorings for the CTE estimations, presents in Table 2, we observe a clear improvement for all quantile levels ranging from a 0.1% and 1.3%, which comes mainly due to the ability of the NCDNN to estimate the CTE by optimizing the loss function, and not being approximated by the MCQRNN directly as the observed average of predictions.

4.3. Estimation of the individual distribution of a driver

The non-crossing property allows us to consistently estimate the tail risk of a driver based on covariates and evaluate individual behaviour based on their profile. By removing the restriction on crossing quantiles, we can approximate the conditional distribution of a profile by estimating a set of quantile levels based on covariates. Furthermore, we can

compare observed values to the predicted distribution to assess the risk an individual contributes to a portfolio, not only in comparison to other drivers but also to the expected values of their profile. This practice, in a similar vein, has been recently employed by Bolancé, Cao and Guillen (2024), who used simple-index models to estimate the marginal effect of the heavy tail of claim costs from an insurance company in order to identify risky drivers.

In insurance, this result opens the discussion on assessing a driver's behaviour by combining driving mechanics with their profile. Considering only expected values, a driver who frequently exceeds speed limits would always be deemed high-risk and thus highly priced. However, not all drivers are the same; professional drivers and common drivers exhibit different behaviours and risk levels.

With the non-crossing dual neural network, we can estimate each individual's distribution and compare it to their observed value, thereby ranking driving behaviour and risk based on their covariates. We present an example of a good driver in Figure 5, driver # 54, colored in blue, where we estimated the values at risk, i.e., the quantiles, of the percentage of kilometers driven above speed limits. The observed value is plotted as a vertical line; an observed value below the 0.5 quantile indicates an overperforming driver contributing less risk than expected. Pricing based on these results should consider this good behaviour, potentially reducing or maintaining the risk premium to retain such customers.

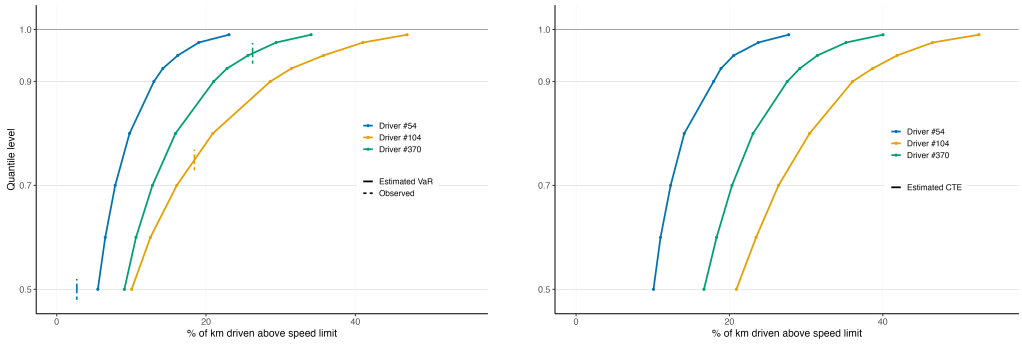

Figure 5. *Estimation of the conditional distribution of the % of kilometers driven above speed limitations for drivers #54, #104 and #370 using VaR (left panel) and CTE (right panel) estimations calculated with the non-crossing dual neural network. In vertical line the observed value.*

In Figure 5, we observe the behaviour of a bad driver, driver #104, colored in yellow, with the percentage of kilometers driven above speed limits falling between the 0.7 and 0.8 estimated quantiles. By incorporating the conditional tail expectation, we can estimate the expected value for a driver with this profile and behaviour. Since this driver's observed value is between the 0.7 and 0.8 quantiles (18% and 20% of kilometers driven above speed limits), it is expected that a driver with a similar profile and behaviour would drive between 25% and 30% of kilometers above speed limits. There-

fore, this driver should be priced higher than a model predicting the expected value, as their profile suggests they should be driving fewer kilometers above speed limits than they actually are.

Figure 5 also illustrates a driver exhibiting very poor behaviour, driver #370 colored in green. The observed value for this driver falls between the 0.95 and 0.975 predicted quantile levels, indicating significantly bad behaviour. This outcome serves as a proxy for identifying customers who are not potentially profitable for the company. In the event that such drivers file a claim, we can expect a high associated cost. Therefore, identifying and appropriately pricing these high-risk drivers is crucial to minimizing potential losses and ensuring the profitability of the insurance portfolio.

5. Discussion

The actuarial literature on usage-based insurance (UBI) is extensive and it has been intensively developed in the last twenty years. Eling and Kraft (2020) made an important contribution to summarize the corpus of knowledge by systematically reviewing 52 academic studies and industry papers published from 2000 to 2019. Many of these articles deal with the role of the distance driven as the key telematics variable for claim frequency estimation. Specifically, Lemaire, Park and Wang (2016) found that annual mileage is an extremely powerful predictor of the number of claims at-fault. In its origins, usage-based insurance was solely based on the total distance driven variable (see Litman (2007), where different types of distance-based insurance price structures are discussed).

In the machine learning literature, there are also many contributions in the field of driving pattern recognition, and how it can be transferred to building accident safety scores and to enhance insurance pricing. Weidner, Transchel and Weidner (2016) found that maneuvers, trips and trip sections as well as the total insurance period can be analyzed to gain significantly scoreable insights into individual driving behaviour. Wüthrich (2017) used high-frequency GPS location data and applied new algorithms to differentiate varying driving styles. He showed how driving styles can be categorized, so that they can be used for a regression analysis in car insurance pricing. Gao and Wüthrich (2018) introduced speed and acceleration heatmaps, categorized with the K-means algorithm to differentiate varying driving styles. Later, Gao, Meng and Wüthrich (2019) investigated the predictive power of covariates extracted from telematics car driving data using the speed-acceleration heat maps. These telematics covariates include K-means classification, principal components, and bottleneck activations from a bottleneck neural network. They found that the first principal component and the bottleneck activations give a better out-of-sample prediction for claims frequencies than other traditional pricing factors such as driver's age. Based on these numerical examples the authors recommended the use of some of these telematics covariates for car insurance pricing. The same authors (see Gao and Wüthrich (2019)) extracted feature information about speeds, acceleration, deceleration, and changes of direction from this high-frequency GPS location data and

used this information to allocate individual car driving trips to selected drivers. Geyer, Kremslehner and Muermann (geyer2020asymmetric) defined a driving factor based on overall distance driven, number of car rides, and speeding and found that speed driving factor has a significant effect on risk. More recently, Meng et al. (2022) calculated risk scores by using a supervised driving risk scoring neural network model and showed that incorporating risk scores improves the prediction performance of classical Poisson regression models for the claim frequency. With these recent references, among many others, we find evidence that risk scoring is becoming central in auto insurance together with big data analytics in this context.

We illustrate this study with a telematics dataset from a spanish insurance company. The case study is similar to what any insurer would do as a first step to score drivers, namely, to analyze the risk of a positive response associated with the risk of a claim given some covariates. Additionally, the model results could be implemented to find the factors that are significantly associated with higher or lower risk.

There are two main advantages of using NCDNN versus risk regressions. First and foremost, the non-crossing property preservation. When conditional risk evaluation is used for pricing, the analyst must estimate the VaR or the CTE given covariates at different quantile levels. Naturally, the estimations should be ordered by quantile level, but this condition does not always hold in practice. A lower quantile level fit that exceeds a higher quantile level fit would be unacceptable, especially if the fitted VaRs or CTEs are being used for pricing or loading a premium. The NCDNN solves this issue by including the natural ordering restriction on VaR and CTE estimations. Secondly, an advantage of NCDNN is the flexibility of neural networks to include cross effects and non-linearities of the covariates, that is superior to what baseline models can provide with linear specifications.

The main disadvantages of implementing a NCDNN are, however, two. On the one hand, the response variable is assumed to be a continuous response. That rules out considering discrete counts for the number of claims as a response, but it could be used for modeling the risk of severities or claim amounts. On the other hand, neural networks are still seen only as exploratory tools in insurance. The fact that they are black boxes that do not serve to write an explicit formula, makes neural network tools inappropriate to the eyes of external stakeholders and incompatible with current regulation requirements. Regulators demand that risk assessment for pricing purposes can explicitly be expressed as a function of rating factors. So, current practice does not allow a neural network fit to be used for pricing, even if explainable artificial intelligence is able to provide an approximation of the variable importance for the output. However, this area is growing very fast, and we believe that deep learning methods combined with explainable artificial intelligence will be helpful to select and even suggest possible non-linearities that explicit classical models can handle. Such procedures could be accepted by regulators as internal steps towards selecting final ratemaking factors. So, we advocate for a cooperation between methods, in particular new machine learning methods, with old methods, rather than just looking at new proposals as opposites or competitors. We also want to

mention that the exploratory tools that are offered by these new methods will certainly be helpful in the discussion of causality and discrimination in insurance, because the new models can cope with many more complexities and can help identifying the role of covariates, even if the role differs among individuals.

We believe that the merits of our proposal can also be expanded in other directions. For example, driving data will certainly be analyzed at increasingly more frequent time intervals than yearly data. That will induce more volatility in the data and, perhaps, methods that are aimed at preserving the natural orderings that we have been presenting here will be welcome if too many inconsistencies are found in the fits otherwise. While still many policyholders pay annual premiums, we are moving into pay-as-you-go schemes that permit monthly rates. Even daily or per trip rates seem plausible, nowadays so we feel that the proposed methods will be helpful in this emerging landscape. We note that the work of Guillen et al. (2021b) explains how risk regressions can be turned into conditional risk scoring that can later be used to summarize driving behaviour very efficiently. This is why we foresee a direct connection of our work to the growing area of insurance telematics.

From an applied perspective, the proposed framework could also be integrated with classical actuarial modeling approaches. In particular, the outputs of the NCDNN may be used to construct derived covariates summarizing individual risk profiles, which could then be incorporated into traditional claims-count models. Such an approach may help bridge telematics-based risk scoring and existing regulatory or operational constraints that favor standard actuarial models. Moreover, while the present work focuses on continuous risk measures, extending neural-network-based architectures to directly model discrete outcomes, such as claim counts, represents a promising direction for future research.

Moreover, while the present work focuses on point estimation of continuous risk measures such as VaR and CTE, uncertainty quantification remains an important topic for practical risk management. In principle, uncertainty around the estimated risk measures could be assessed using resampling-based approaches, such as bootstrap methods, or through Bayesian extensions of the proposed architecture, for example by introducing prior distributions on network parameters or by adopting Bayesian neural network formulations. Exploring such approaches to obtain interval estimates or posterior uncertainty measures represents a promising direction for future research and is beyond the scope of the present work. Finally, while the present study concentrates on continuous risk measures, extending neural-network-based architectures to directly model discrete outcomes, such as claim counts, also represents an appealing avenue for future research.

6. Conclusions

This paper presents a model for estimating the VaR and CTE of a response variable given a set of covariates and involving several quantile levels. The classical approach based on quantile regression is compared to a neural network approach with non-crossing prop-

erties called the NCDNN. Baseline risk regression models do not assure that the non-crossing condition holds, because estimating the VaR and the corresponding CTE at each quantile level are done separately for each quantile level at a time.

This type of analysis brings to light two main points in pricing and risk assessment. First, estimating a driver's risk solely by predicting their expected value is insufficient to capture all potential costs a company may incur. Second, a non-crossing quantile-based approach to risk estimation allows for evaluating how each driver behaves within their profile by locating their observed value between the estimated quantiles. Additionally, the estimation of Conditional Tail Expectations provides a more quantitative approach to understanding the expected value of the percentage of kilometers driven above speed limits when a specific quantile is surpassed. More examples are shown in Figure 6 in the Appendix for further analysis.

It is important to note that this algorithm is not limited to motor insurance but can be applied to any target factors with heavy-tailed distributions where extreme values are of interest to analysts. For example, we are working on estimating energy consumption using household factors, enabling energy companies to provide sufficient resources and manage reserves to prevent energy shortages. Another area of study involves applying the estimation of non-crossing quantiles to predict the conditional distribution of observations in asset pricing. Evaluating stock performance, particularly extreme values in stock prices, can be a valuable tool for identifying investment opportunities and determining underperforming or overperforming stocks. This information allows investors and regulators to manage investment portfolios more effectively in line with their expectations and objectives.

For all these reasons, we think that considering non-crossing algorithms when dealing with heavy tailed response variables in datasets that include fast dynamics, like trip driving data or daily data will be necessary.

Acknowledgements

We would like to acknowledge the Fundació Banc Sabadell: "Ajudes a la investigació 2022", the Spanish Ministry of Science grant PID2019-105986GB-C21/AEI/10.13039/501100011033, the Federación Española de Municipios y Provincias (FEMP) contract number OTR2025-26067SERVI and the NextGenerationEU grant number TED2021-130187B-I00 for their support of our research. We also thank the two anonymous reviewers for their insightful comments and suggestions, which have significantly improved the quality of this manuscript.

References

Acerbi, C. and Szekely, B. (2014). Back-testing expected shortfall. *Risk*, 27(11):76–81.
Asimit, A. V., Furman, E., Tang, Q., and Vernic, R. (2011). Asymptotics for risk cap-

ital allocations based on conditional tail expectation. *Insurance: Mathematics and Economics*, 49(3):310–324.

Baione, F. and Biancalana, D. (2021). An application of parametric quantile regression to extend the two-stage quantile regression for ratemaking. *Scandinavian Actuarial Journal*, 2021(2):156–170.

Basel Committee on Banking Supervision (2016). Minimum capital requirements for market risk. https://www.bis.org/bcbs/publ/d352.htm.

Bodnar, G. M., Hayt, G. S., and Marston, R. C. (1998). 1998 Wharton survey of financial risk management by US non-financial firms. *Financial Management*, 27(4):70–91.

Bolancé, C., Cao, R., and Guillen, M. (2024). Conditional likelihood based inference on single-index models for motor insurance claim severity. *SORT*, 48:235–258.

Bondell, H. D., Reich, B. J., and Wang, H. (2010). Noncrossing quantile regression curve estimation. *Biometrika*, 97(4):825–838.

Cannon, A. J. (2018). Non-crossing nonlinear regression quantiles by monotone composite quantile regression neural network, with application to rainfall extremes. *Stochastic Environmental Research and Risk Assessment*, 32(11):3207–3225.

Catania, L. and Luati, A. (2022). Semiparametric modeling of multiple quantiles. *Journal of Econometrics*. https://doi.org/10.1016/j.jeconom.2022.11.002.

Chen, J. M. (2018). On exactitude in financial regulation: Value-at-risk, expected shortfall, and expectiles. *Risks*, 6(2):61.

Chernozhukov, V., Fernández-Val, I., and Galichon, A. (2010). Quantile and probability curves without crossing. *Econometrica*, 78(3):1093–1125.

Ciliberti, S., Kondor, I., and Mézard, M. (2007). On the feasibility of portfolio optimization under expected shortfall. *Quantitative Finance*, 7(4):389–396.

Dette, H. and Volgushev, S. (2008). Non-crossing non-parametric estimates of quantile curves. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(3):609–627.

Dimitriadis, T. and Bayer, S. (2019). A joint quantile and expected shortfall regression framework. *Electronic Journal of Statistics*, 13(1):1823–1871.

Eling, M. and Kraft, M. (2020). The impact of telematics on the insurability of risks. *The Journal of Risk Finance*, 21(2):77–109.

Fissler, T., Merz, M., and Wüthrich, M. V. (2023). Deep quantile and deep composite triplet regression. *Insurance: Mathematics and Economics*, 109:94–112.

Fissler, T. and Ziegel, J. F. (2016). Higher order elicitability and Osband's principle. *The Annals of Statistics*, 44(4):1680–1707.

Frees, E. W., Shi, P., and Valdez, E. A. (2009). Actuarial applications of a hierarchical insurance claims model. *ASTIN Bulletin: The Journal of the IAA*, 39(1):165–197.

Gao, G., Meng, S., and Wüthrich, M. V. (2019). Claims frequency modeling using telematics car driving data. *Scandinavian Actuarial Journal*, 2019(2):143–162.

Gao, G. and Wüthrich, M. V. (2018). Feature extraction from telematics car driving heatmaps. *European Actuarial Journal*, 8:383–406.

Gao, G. and Wüthrich, M. V. (2019). Convolutional neural network classification of telematics car driving data. *Risks*, 7(1):6.

Geyer, A., Kremslehner, D., and Muermann, A. (2020). Asymmetric information in automobile insurance: Evidence from driving behavior. *Journal of Risk and Insurance*, 87(4):969–995.

Gneiting, T. (2011). Making and evaluating point forecasts. *Journal of the American Statistical Association*, 106(494):746–762.

González-Pedraz, C., Moreno, M., and Peña, J. I. (2014). Tail risk in energy portfolios. *Energy Economics*, 46:422–434.

Granger, C. W. (2010). Some thoughts on the development of cointegration. *Journal of econometrics*, 158(1):3–6.

Guillen, M., Bermúdez, L., and Pitarque, A. (2021a). Joint generalized quantile and conditional tail expectation regression for insurance risk analysis. *Insurance: Mathematics and Economics*, 99:1–8.

Guillen, M., Nielsen, J. P., Pérez-Marín, A. M., and Elpidorou, V. (2020). Can automobile insurance telematics predict the risk of near-miss events? *North American Actuarial Journal*, 24(1):141–152.

Guillen, M., Pérez-Marín, A. M., and Alcañiz, M. (2021b). Percentile charts for speeding based on telematics information. *Accident Analysis & Prevention*, 150:105865.

He, X. (1997). Quantile curves without crossing. *The American Statistician*, 51(2):186–192.

Koenker, R. (2017). Quantile regression: 40 years on. *Annual Review of Economics*, 9:155–176.

Koenker, R. and Bassett, G. (1978). Regression quantiles. *Econometrica: Journal of the Econometric Society*, 46(1):33–50.

Kourouma, L., Dupre, D., Sanfilippo, G., and Taramasco, O. (2010). Extreme value at risk and expected shortfall during financial crisis. *Available at SSRN 1744091*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Lemaire, J., Park, S. C., and Wang, K. C. (2016). The use of annual mileage as a rating variable. *ASTIN Bulletin: The Journal of the IAA*, 46(1):39–69.

Litman, T. (2007). Distance-based vehicle insurance feasibility, costs and benefits. *Victoria*, 11.

Liu, Y. and Wu, Y. (2011). Simultaneous multiple non-crossing quantile regression estimation using kernel constraints. *Journal of Nonparametric Statistics*, 23(2):415–437.

Meng, S., Wang, H., Shi, Y., and Gao, G. (2022). Improving automobile insurance claims frequency prediction with telematics car driving data. *ASTIN Bulletin: The Journal of the IAA*, 52(2):363–391.

Moon, S. J., Jeon, J.-J., Lee, J. S. H., and Kim, Y. (2021). Learning multiple quantiles with neural networks. *Journal of Computational and Graphical Statistics*, 30(4):1238–1248.

Nolde, N. and Ziegel, J. F. (2017). Elicitability and backtesting: Perspectives for banking regulation. *The Annals of Applied Statistics*, 11(4):1833–1874.

Patton, A. J., Ziegel, J. F., and Chen, R. (2019). Dynamic semiparametric models for expected shortfall (and Value-at-Risk). *Journal of Econometrics*, 211(2):388–413.

Pérez-Marín, A. M., Guillen, M., Alcañiz, M., and Bermúdez, L. (2019). Quantile regression with telematics information to assess the risk of driving above the posted speed limit. *Risks*, 7(3):80.

Schmidt, L. and Zhu, Y. (2016). Quantile spacings: A simple method for the joint estimation of multiple quantiles without crossing. *Available at SSRN 2220901*.

Takeuchi, I. and Furuhashi, T. (2004). Non-crossing quantile regressions by svm. *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541)*, 1:401–406.

Taylor, J. W. (2019). Forecasting value at risk and expected shortfall using a semiparametric approach based on the asymmetric Laplace distribution. *Journal of Business & Economic Statistics*, 37(1):121–133.

Taylor, J. W. (2020). Forecast combinations for value at risk and expected shortfall. *International Journal of Forecasting*, 36(2):428–441.

Thaler, M., Grabec, I., and Poredoš, A. (2005). Prediction of energy consumption and risk of excess demand in a distribution system. *Physica A: Statistical mechanics and its applications*, 355(1):46–53.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30.

Vidal-Llana, X., Coia, V., and Guillen, M. (2022). Alternative scoring function specifications for estimating conditional tail expectation regression via neural networks. In *Contributions to Risk Analysis: RISK 2022*, pages 139–148. Cuadernos de la Fundación. Fundación Mapfre. https://www.fundacionmapfre.org/publicaciones/todas/contributions-risk-analysis-2022/.

Vidal-Llana, X. and Guillén, M. (2022). Cross-sectional quantile regression for estimating conditional VaR of returns during periods of high volatility. *The North American Journal of Economics and Finance*, 63:101835.

Voulodimos, A., Doulamis, N., Doulamis, A., and Protopapadakis, E. (2018). Deep learning for computer vision: A brief review. *Computational Intelligence and Neuroscience*, 2018.

Weidner, W., Transchel, F. W., and Weidner, R. (2016). Classification of scale-sensitive telematic observables for riskindividual pricing. *European Actuarial Journal*, 6:3–24.

Wüthrich, M. V. (2017). Covariate selection from telematics car driving data. *European Actuarial Journal*, 7:89–108.

Yu, K., Lu, Z., and Stander, J. (2003). Quantile regression: applications and current research areas. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 52(3):331–350.

Zhang, L. (2016). Flood hazards impact on neighborhood house prices: A spatial quantile regression analysis. *Regional Science and Urban Economics*, 60:12–19.

Ziegel, J. F., Krüger, F., Jordan, A., and Fasciati, F. (2020). Robust forecast evaluation of expected shortfall. *Journal of Financial Econometrics*, 18(1):95–120.

Appendix

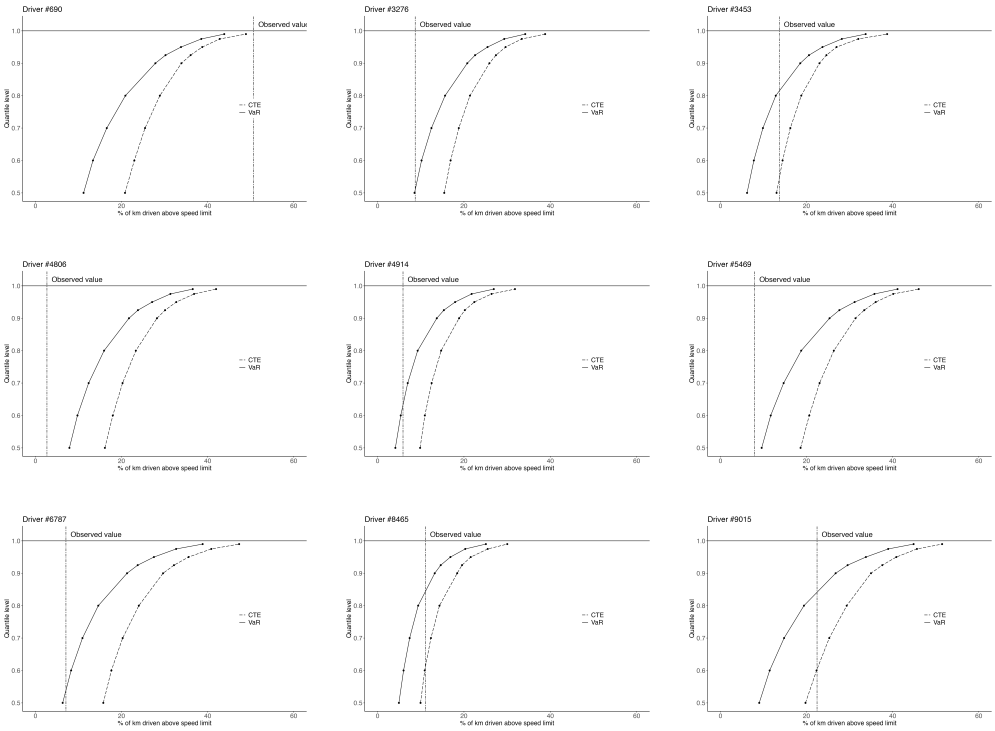

Figure 6. *Estimation of the conditional distribution of the % of kilometers driven above speed limitations for different drivers of the database using VaR and CTE estimations calculated with the non-crossing dual neural network. In vertical line the observed value.*

.

Table 3. *Mean and standard deviance of errors for the NCDNN across 50 seeds.*

Loss NCDNN	Quantile Level								
	0.5	0.6	0.7	0.8	0.9	0.925	0.95	0.975	0.99
Mean	13743.1	11391.5	8884.8	6203.5	3305.4	2523.9	1722.4	895.3	372.7
Std. Dev.	2698.5	2031.3	1429.9	879.2	401.0	285.6	179.7	90.1	35.1

SORT 50 (1) January-June 2026, 67-88 DOI: 10.57645/20.8080.02.32

An algorithm for reconciling indicators across multiple dimensions: weighted iterative proportional fitting

Jose M. Pavía[1], Josep Lledó[2] and Priscila Espinosa[3]

Abstract

Reconciling multidimensional count data across multiple sources is a common challenge in social and economic research. Iterative proportional fitting is widely used for this purpose, but aligning indices under weighted sum-convex constraints calls for a more flexible approach. We introduce the weighted iterative proportional fitting algorithm, which incorporates sum-weighted constraints to adjust indicators—such as death-risk indices by wealth, habitat, and climate—while preserving marginal consistency. Weighted iterative proportional fitting has been implemented in an R package of the same name, enabling scholars, statisticians, and policymakers' advisors, among others, to apply it easily to multidimensional data.

MSC: 62P20, 62H10, 15A23.

Keywords: WIPF, biproportional fitting, RAS, raking, matrix scaling, socioeconomic indices, official statistics.

1. Introduction

The iterative proportional fitting (IPF) algorithm is a well-established procedure for reconciling count data coming from two or more sources of information. Typically, the process involves adjusting the (non-negative) inner-entries of a matrix (or a two-way contin-

[1] Universitat de Valencia, Valencia (Spain), email: pavia@uv.es ORCID: https://orcid.org/0000-0002-0129-726X. Corresponding author.

[2] Universitat de Valencia, Valencia (Spain), email: josep.lledo@uv.es ORCID: https://orcid.org/0000-0002-7475-8549

[3] Universitat de Valencia, Valencia (Spain), email: priscila.espinosa@uv.es ORCID: https://orcid.org/0000-0002-3870-6194

Received: January 2026
Accepted: March 2026

gency table) of estimates—or entries derived from a less reliable source of information—to match some known aggregate margins (the row and column totals) coming from a more reliable data source.

Beyond this ad hoc, data-reconciliation interpretation, IPF is also closely connected to maximum likelihood estimation and constrained optimization (Zaloznik, 2011). It can be viewed both as a procedure for adjusting a multidimensional table to match known marginals while preserving the initial cell structure (i.e., prior information) as much as possible, and as a method for estimating maximum likelihood parameters in theoretical or model-based contexts; particularly within log-linear models. In the first case, IPF yields the distribution that is closest to the initial distribution in terms of relative entropy—that is, the distribution that minimizes the Kullback-Leibler divergence—subject to the marginal constraints. In the second case, however, despite the widespread belief that IPF always converges to the maximum likelihood (ML) estimate consistent with the marginal constraints, the property only holds when the initial estimates are consistent with the model structure.

The procedure, which can be traced back to Kruithof (1937) and Deming and Stephan (1940), is known by various names across different disciplines in the literature. These include biproportional fitting in statistics (Pavía, Cabrer and Sala, 2009), RAS in economics (Wiebe and Lenzen, 2016), raking in survey research (Deville, Särndal and Sautory, 1993), and matrix scaling in computer science (Kalantari and Khachiyan, 1996). Furthermore, IPF is also referred to as iterative proportional scaling (IPS), particularly in the context of ML estimation, as used in machine learning, image processing, or graphical models (Coons, Langer and Ruddy, 2024).

Despite the relevance of the approach, as demonstrated by its repeated rediscovery by different scientific communities (Allen-Zhu et al., 2017), its broad use in a multitude of problems (Idel, 2016), and its many extensions (Klimova and Rudas, 2015; Suesse et al., 2017; Fournier Gabela, 2020; von Lindheim and Steidl, 2023), a generalization that involves making adjustments after imposing constraints on weighted (convex) sums of inner values across dimensions is still missing. The aim of this paper is to address this gap by generalizing the IPF algorithm in that direction. This gives rise to a new procedure: the weighted iterative proportional fitting (WIPF) algorithm, available in an R-package of the same name (Pavía, 2026).

Our motivation for developing this new algorithm is practical rather than theoretical. During the process of estimating socio-economic death-risk indices as a function of contextual wealth, habitat size, and climate area, we experienced smaller sampling sizes (exposed-to-risk) and increasing levels of uncertainty and lack of reliability in the estimates as the population was progressively split by a larger number of factors. Given the convex aggregation relationships linking inner and outer levels of risk-indices, we found that three- and two-interaction estimates could be significantly improved by making the initial estimates congruent with the two- and one-factor estimates, and that this could be easily performed by generalizing IPF. This paper describes the new algorithm, presents the `WIPF` package, and exemplifies its use by applying it to the problem of mak-

ing congruent death-risk indices of different levels. Obviously, the usefulness of this new algorithm extends beyond this example, as it can be employed to reconcile any set of indicators that are linearly related with positive coefficients—for instance, harmonizing life expectancies jointly estimated by educational attainment and sex with their corresponding marginal life expectancies.

2. A motivating application

Understanding mortality patterns is fundamental for assessing social well-being and inequality (Lagravinese, Liberati and Resce, 2020). Differences in mortality levels across population groups provide valuable information about the distribution of health, living conditions, and social opportunity within a society (Pavía, Lledó and Roig, 2026). Assessing and quantifying these differences is essential for monitoring social progress and designing evidence-based public policies, being also of value for guaranteeing fairness in the insurance industry.

Beyond individual characteristics, contextual factors such as neighborhood wealth, habitat size, and climatic conditions exert a strong influence on mortality patterns. Populations with higher economic resources tend to enjoy better access to healthcare and healthier living environments (McMaughan, Oloruntoba and Smith, 2020; Dwyer-Lindgren et al., 2024), while rural or sparsely populated areas often face limited access to essential services (Cohen et al., 2023). Likewise, growing evidence links climatological conditions and climate change–related phenomena (e.g., extreme heat, pollution, or drought episodes) with increased mortality risks, particularly among vulnerable populations (Bell, O'Neill and Zanobetti, 2024; García-León et al., 2024; Masselot et al., 2025). Measuring and monitoring mortality through these contextual dimensions is therefore crucial to identify and address emerging inequalities driven by environmental and socioeconomic transformations.

In this context, drawing on more than four billion individual demographic records from the Spanish population (Pavía et al., 2026) and adapting the methodology proposed in Pavía and Lledó (2022), we derived a set of mortality indices by age and sex that capture how death risks vary with contextual factors such as neighborhood wealth, habitat size, and climatological conditions. These indices provide a multidimensional picture of inequality in mortality, reflecting how the social and environmental context in which individuals live is systematically related to their survival prospects.

The resulting indicators can be readily integrated into social statistics and monitoring systems to complement traditional demographic and economic measures. They can support the design and evaluation of regional and social policies aimed at reducing health inequalities, enhancing the capacity of policymakers and researchers to identify vulnerable areas and population groups.

Mortality risks also constitute the keystone of the life insurance business, and following the entry into force of the Test-Achats ruling (European Commission, 2012), EU insurers are interested in introducing easily accessible additional variables, other than

age, into their pricing and reserving processes—variables that also significantly affect mortality and survival risks—so as to better assess and segment their portfolios (Lledó and Pavía, 2026). These indices can be easily introduced into actuarial processes: by simply knowing the postal addresses of the insured, the baseline company death rate can be multiplied by the relevant index to adjust the initially assumed probabilities of death or survival, thus incorporating contextual risks.

However, estimating mortality indices that account simultaneously for several contextual dimensions poses a methodological challenge, since these factors interact and are not independent. When multiple factors are combined, small sample sizes arise, producing unstable or volatile estimates. Fortunately, indices at different aggregation levels are linearly related: outer-level indices can be expressed as convex combinations of inner risk indices. This property opens the door to more efficient estimation procedures, which is precisely where the methodological contribution of this paper lies.

In particular, following with this example, (i) symbolizing age by x and sex by s, (ii) designating by w, h and c as the generic levels of the factors contextual wealth, habitat size and climatological area, respectively (with $w = 1,\ldots;W$, $h = 1,\ldots;H$, $c = 1,\ldots,C$), and (iii) defining by $E^{(\cdot)}_{x,s}$ as the exposed-to-risk population and $I^{(\cdot)}_{x,s}$ as the corresponding index—for instance, $E^{w,h}_{x,s}$ $\left(=\sum_{c=1}^{C} E^{w,h,c}_{x,s}\right)$ denotes the size of population exposed-to-risk with age x and sex s in areas characterized by contextual wealth w and habitat size h, while $I^{w,h,c}_{x,s}$ represents the index corresponding to age x and sex s for population living in areas with contextual wealth w, habitat size h, and climatological condition c—the following theoretical relationships hold:

$$1 = \sum_{w=1}^{W} \frac{E^{w}_{x,s}}{E_{x,s}} I^{w}_{x,s}, \qquad 1 = \sum_{h=1}^{H} \frac{E^{h}_{x,s}}{E_{x,s}} I^{h}_{x,s}, \qquad 1 = \sum_{c=1}^{C} \frac{E^{c}_{x,s}}{E_{x,s}} I^{c}_{x,s},$$

$$I^{w}_{x,s} = \sum_{h=1}^{H} \frac{E^{w,h}_{x,s}}{E^{w}_{x,s}} I^{w,h}_{x,s} = \sum_{c=1}^{C} \frac{E^{w,c}_{x,s}}{E^{w}_{x,s}} I^{w,c}_{x,s},$$

$$I^{h}_{x,s} = \sum_{w=1}^{W} \frac{E^{w,h}_{x,s}}{E^{h}_{x,s}} I^{w,h}_{x,s} = \sum_{c=1}^{C} \frac{E^{h,c}_{x,s}}{E^{h}_{x,s}} I^{h,c}_{x,s},$$

$$I^{c}_{x,s} = \sum_{w=1}^{W} \frac{E^{w,c}_{x,s}}{E^{c}_{x,s}} I^{w,c}_{x,s} = \sum_{h=1}^{H} \frac{E^{h,c}_{x,s}}{E^{c}_{x,s}} I^{h,c}_{x,s},$$

$$I^{w,h}_{x,s} = \sum_{c=1}^{C} \frac{E^{w,h,c}_{x,s}}{E^{w,h}_{x,s}} I^{w,h,c}_{x,s}, \qquad I^{w,c}_{x,s} = \sum_{h=1}^{H} \frac{E^{w,h,c}_{x,s}}{E^{w,c}_{x,s}} I^{w,h,c}_{x,s}, \qquad I^{h,c}_{x,s} = \sum_{w=1}^{W} \frac{E^{w,h,c}_{x,s}}{E^{h,c}_{x,s}} I^{w,h,c}_{x,s}.$$

Unfortunately, these theoretical relationships are not preserved when assessed using estimated indices, with discrepancies tending to increase as the number of involved factors grows. To address this issue, we propose a generalization of the iterative proportional fitting procedure that incorporates weights. This leads to a new algorithm, referred to as weighted iterative proportional fitting (WIPF).

3. The weighted iterative proportional fitting algorithm

3.1. Problem setting

In this section, we detail WIPF for the case at hand, with three dimensions. For analogy with IPF, we adopt a contingency table representation and, without loss of generality, set the one-dimensional weighted sums to 1. The approach and notation can be straightforwardly generalized to any number of dimensions.

Let us assume three categorical variables X, Y and Z with R, C and L levels (from rows, columns and layers), respectively, and two 3-way arrays: an initial 3-way array, referred to as the seed, of non-negative initial indices/indicators I^0_{rcl}, $S = \left[I^0_{rcl}\right]$, and a 3-way contingency table of non-negative weights, $W = \left[w_{rcl}\right]$, where $r \in \{1,\ldots,R\}$, $c \in \{1,\ldots,C\}$ and $l \in \{1,\ldots,L\}$ correspond to the levels of the first, second and third variable, respectively. In addition, let us suppose that these initial indices correspond to estimates of the true underlying indices, I_{rcl}, which satisfy a set of weighted sum-convex constraints across their marginal dimensions:

$$\sum_{r=1}^{R} \frac{w_{r++}}{w_{+++}} I_{r**} = \sum_{c=1}^{C} \frac{w_{+c+}}{w_{+++}} I_{*c*} = \sum_{l=1}^{L} \frac{w_{++l}}{w_{+++}} I_{**l} = 1,$$

$$\begin{aligned} I_{r**} &= \sum_{c=1}^{C} \frac{w_{rc+}}{w_{r++}} I_{rc*} = \sum_{l=1}^{L} \frac{w_{r+l}}{w_{r++}} I_{r*l}, \\ I_{*c*} &= \sum_{r=1}^{R} \frac{w_{rc+}}{w_{+c+}} I_{rc*} = \sum_{l=1}^{L} \frac{w_{+cl}}{w_{+c+}} I_{*cl}, \\ I_{**l} &= \sum_{r=1}^{R} \frac{w_{r+l}}{w_{++l}} I_{r*l} = \sum_{c=1}^{C} \frac{w_{+cl}}{w_{++l}} I_{*cl}, \end{aligned} \tag{1}$$

$$I_{rc*} = \sum_{l=1}^{L} \frac{w_{rcl}}{w_{rc+}} I_{rcl}, \qquad I_{r*l} = \sum_{c=1}^{C} \frac{w_{rcl}}{w_{r+l}} I_{rcl}, \qquad I_{*cl} = \sum_{r=1}^{R} \frac{w_{rcl}}{w_{+cl}} I_{rcl},$$

where $\mathcal{M} = \left\{I_{r**}, I_{*c*}, I_{**l}, I_{rc*}, I_{r*l}, I_{*cl}\right\}$ represents the set of (known) margin indices—for which consistency across the margins is also assumed—and sub-index $+$ refers to the standard notation of summation over the corresponding sub-index, e.g., $w_{r+l} = \sum_{c=1}^{C} w_{rcl}$. Note that in this case the margins I_{r**}, I_{*c*} and I_{**l} are vectors, and the margins I_{rc*}, I_{r*l} and I_{*cl} are matrices.

To facilitate interpretation of the notation, we present a small illustrative example with three categorical variables X, Y, and Z, each taking two levels. Table 1 shows the individual indices in a long-table format, with the last column displaying the corresponding 1D marginal weighted sums. Table 2 presents the 2D marginal weighted sums, arranged in three blocks corresponding to I_{rc*}, I_{r*l}, and I_{*cl}, where each block shows the weighted averages over the remaining dimension. The denominators in these expressions

represent the appropriate sums of weights used in the marginal calculations. Together, these two tables provide a concrete illustration of how the 3-way array of indices relates to its marginal sums through the weights.

Table 1. *Illustrative 2×2×2 array of indices with 1D marginal weighted sums.*

X	Y	Z	I_{rcl}	1D Marginals
X_1	Y_1	Z_1	I_{111}	$I_{1**} = \frac{w_{111}I_{111}+w_{121}I_{121}+w_{112}I_{112}+w_{122}I_{122}}{w_{111}+w_{121}+w_{112}+w_{122}} = \frac{w_{111}I_{111}+w_{112}I_{112}+w_{121}I_{121}+w_{122}I_{122}}{w_{111}+w_{112}+w_{121}+w_{122}}$
X_1	Y_2	Z_1	I_{121}	$I_{2**} = \frac{w_{211}I_{211}+w_{221}I_{221}+w_{212}I_{212}+w_{222}I_{222}}{w_{211}+w_{221}+w_{212}+w_{222}} = \frac{w_{211}I_{211}+w_{212}I_{212}+w_{221}I_{221}+w_{222}I_{222}}{w_{211}+w_{212}+w_{221}+w_{222}}$
X_1	Y_1	Z_2	I_{112}	
X_1	Y_2	Z_2	I_{122}	$I_{*1*} = \frac{w_{111}I_{111}+w_{211}I_{211}+w_{112}I_{112}+w_{212}I_{212}}{w_{111}+w_{211}+w_{112}+w_{212}} = \frac{w_{111}I_{111}+w_{112}I_{112}+w_{211}I_{211}+w_{212}I_{212}}{w_{111}+w_{112}+w_{211}+w_{212}}$
X_2	Y_1	Z_1	I_{211}	$I_{*2*} = \frac{w_{121}I_{121}+w_{221}I_{221}+w_{122}I_{122}+w_{222}I_{222}}{w_{121}+w_{221}+w_{122}+w_{222}} = \frac{w_{121}I_{121}+w_{122}I_{122}+w_{221}I_{221}+w_{222}I_{222}}{w_{121}+w_{122}+w_{221}+w_{222}}$
X_2	Y_2	Z_1	I_{221}	
X_2	Y_1	Z_2	I_{212}	$I_{**1} = \frac{w_{111}I_{111}+w_{211}I_{211}+w_{121}I_{121}+w_{221}I_{221}}{w_{111}+w_{211}+w_{121}+w_{221}} = \frac{w_{111}I_{111}+w_{121}I_{121}+w_{211}I_{211}+w_{221}I_{221}}{w_{111}+w_{121}+w_{211}+w_{221}}$
X_2	Y_2	Z_2	I_{222}	$I_{**2} = \frac{w_{112}I_{112}+w_{212}I_{212}+w_{122}I_{122}+w_{222}I_{222}}{w_{112}+w_{212}+w_{122}+w_{222}} = \frac{w_{112}I_{112}+w_{122}I_{122}+w_{212}I_{212}+w_{222}I_{222}}{w_{112}+w_{122}+w_{212}+w_{222}}$

Table 2. *Illustrative 2×2×2 array: 2D marginal weighted sums.*

I_{rc*}		I_{r*l}		I_{*cl}	
$I_{11*} = \frac{w_{111}I_{111}+w_{112}I_{112}}{w_{111}+w_{112}}$	$I_{12*} = \frac{w_{121}I_{121}+w_{122}I_{122}}{w_{121}+w_{122}}$	$I_{1*1} = \frac{w_{111}I_{111}+w_{121}I_{121}}{w_{111}+w_{121}}$	$I_{1*2} = \frac{w_{112}I_{112}+w_{122}I_{122}}{w_{112}+w_{122}}$	$I_{*11} = \frac{w_{111}I_{111}+w_{211}I_{211}}{w_{111}+w_{211}}$	$I_{*12} = \frac{w_{112}I_{112}+w_{212}I_{212}}{w_{112}+w_{212}}$
$I_{21*} = \frac{w_{211}I_{211}+w_{212}I_{212}}{w_{211}+w_{212}}$	$I_{22*} = \frac{w_{221}I_{221}+w_{222}I_{222}}{w_{221}+w_{222}}$	$I_{2*1} = \frac{w_{211}I_{211}+w_{221}I_{221}}{w_{211}+w_{221}}$	$I_{2*2} = \frac{w_{212}I_{212}+w_{222}I_{222}}{w_{212}+w_{222}}$	$I_{*21} = \frac{w_{121}I_{121}+w_{221}I_{221}}{w_{121}+w_{221}}$	$I_{*22} = \frac{w_{122}I_{122}+w_{222}I_{222}}{w_{122}+w_{222}}$

The convex-weights are calculated by normalizing W in a margin-dependent manner, ensuring that when aggregated across the missing dimensions the normalized weights sum to one. In the `WIPF` package, this is controlled by the `normalize` argument. Alternatively, the relationships can be formulated without normalizing the weights. In this case, the requirement that all zero-dimensional target margins equal 1 is replaced by the weaker assumption that all one-dimensional weighted sums coincide. Moreover, in the special case when the weights are unitary, the problem collapses to the one corresponding to the standard IPF procedure.

Importantly, the normalized and non-normalized formulations are not, in general, equivalent up to a simple rescaling of the solution. While both formulations preserve the relative structure induced by the weights, the resulting fitted arrays tend to differ in absolute magnitude unless the weights are proportional within each marginal dimension. A small numerical illustration showing that the two formulations can lead to non-proportional solutions is provided in Appendix C. For the remainder of this exposition, we focus on the more complex case of weighted sum-convex constraints.

3.2. The algorithm

WIPF is the procedure that, given a non-empty subset $\mathcal{C} \subseteq \mathcal{M}$ of target margin constraints, iteratively updates the seed array based on the targets and weights. Although the full set of margins may not be available in a given problem, for completeness we detail below the full set of updating adjustments (steps). In particular, $\forall r,c,l$, steps at iteration $i \geq 1$ may be computed by the equations:

$$I_{rcl}^{(1)} = I_{rcl}^{i-1} \frac{I_{r**}}{I_{r**}^{i-1}}, \qquad I_{rcl}^{(2)} = I_{rcl}^{(1)} \frac{I_{*c*}}{I_{*c*}^{(1)}},$$

$$I_{rcl}^{(3)} = I_{rcl}^{(2)} \frac{I_{**l}}{I_{**l}^{(2)}}, \qquad I_{rcl}^{(4)} = I_{rcl}^{(3)} \frac{I_{rc*}}{I_{rc*}^{(3)}},$$

$$I_{rcl}^{(5)} = I_{rcl}^{(4)} \frac{I_{r*l}}{I_{r*l}^{(4)}}, \qquad I_{rcl}^{i} = I_{rcl}^{(5)} \frac{I_{*cl}}{I_{*cl}^{(5)}}.$$

where the temporal margins are calculated using the expressions defined in (1) applied to the temporarily updated indices.

Although these steps describe an updating process that first cycles through individual dimensions and then through pairs of dimensions, alternative cycling sequences are equally valid. If only a subset of the target margins is available, i.e., $\mathcal{C} \subset \mathcal{M}$, adjustments are restricted to those margins at each iteration. In other words, each iteration has as many steps as there are marginal configurations to adjust to. These iterations are repeated until the maximum difference between two consecutive iterations—whether in the indices or the constraints—does not exceed a pre-specified, sufficiently small threshold $\varepsilon > 0$. The values at the last iteration comprise the WIPF solution.

It should be noted that the WIPF solution cannot be obtained through the classical IPF procedure. Although one might attempt to recast the WIPF problem as an IPF problem—by defining the seed as the elementwise product of the initial seed array $S = [I_{rcl}^0]$ and the weight matrix $W = [w_{rcl}]$, and setting the target margins as the product of the intended constraints (for $\mathcal{C} \subseteq \mathcal{M}$) and the appropriate partial sums of W—this alternative formulation does not converge to the WIPF solution. While this can be easily illustrated with a numerical example (see Appendix A), the core reason lies in the treatment of the weights: in WIPF, weights remain fixed throughout the iterations, whereas in the IPF-based formulation, they are implicitly altered at each step. As a result, the indices produced by the latter are not weight-consistent.

Despite the differences between WIPF and classical IPF, as with classical IPF, if the initial table contains some $I_{rcl}^0 = 0$, the corresponding value will remain zero throughout the iterations. This property makes it straightforward to fit tables with structural zeros. Furthermore, in the same vein as IPF—which requires the target marginal totals to be internally consistent (i.e., in two dimensions, row and column sums must match) for convergence—WIPF requires the target marginal (convex) sums to be internally consistent. When this condition is met, convergence is typically achieved in very few iterations.

4. On the theoretical and computational properties of WIPF

In this section, we discuss the theoretical foundations of the Weighted Iterative Proportional Fitting (WIPF) algorithm. While Section 3 describes the algorithmic procedure and draws analogies with classical IPF, it is useful to formalize the conditions under which the algorithm is guaranteed to produce a well-defined solution, to clarify its uniqueness, and to relate it to standard convex optimization principles. We organize the discussion from three complementary perspectives, to subsequently end the section with computational considerations.

4.1. Existence, uniqueness, and convergence

As WIPF retains the typical multiplicative structure of IPF, it inherits the fundamental theoretical properties of IPF under standard regularity conditions (Bishop, Fienberg and Holland, 2007). Specifically, if the seed array is strictly positive and the weighted marginal constraints are mutually compatible, then a feasible solution satisfying all weighted sum constraints exists, is unique within the multiplicative family generated from the seed, and the iterative procedure converges to this solution. The justification follows from the fact that each WIPF step is a multiplicative scaling operation analogous to the updating steps of classical IPF, with margins computed as convex combinations determined by the weights. These properties hold for any set of non-negative weights, provided that the target margins are consistent and compatible.

4.2. Formulation as a weighted Kullback–Leibler minimization problem

WIPF can equivalently be expressed as the solution of a convex optimization problem that minimizes the weighted Kullback–Leibler (KL) divergence between the fitted array and the initial seed, subject to the weighted sum constraints. Formally, in the three dimensional case, if $S = \left[I^0_{ijk}\right]$, denotes the seed array and $W = \left[w_{ijk}\right]$ the array of non-negative weights, the optimization problem can be written as

$$\arg\min_{I_{ijk}>0} \sum_{i,j,k} w_{ijk} I_{ijk} \log\left(\frac{I_{ijk}}{I^0_{ijk}}\right) \quad \text{subject to the weighted marginal constraints.}$$

The strict convexity of the weighted KL objective and the convexity of the feasible set ensure the existence of a unique minimizer whenever the constraints are compatible. Note that the classical KL representation of IPF is recovered as the special case in which all weights are equal to one.

This equivalence provides an alternative perspective for understanding WIPF and facilitates connections with other convex optimization techniques. A simple example illustrating the equivalence between the solutions of this weighted KL optimization problem and the normalized WIPF solution is provided in Appendix B.

4.3. Bregman projection interpretation

Finally, WIPF can be also interpreted as a sequence of Bregman projections onto convex sets defined by the weighted marginal constraints, using the corresponding generalized weighted Kullback–Leibler divergence (Kurras, Greenewald and Grosse, 2015) as the generating function:

$$D^{w}_{\mathrm{KL}}(I \,\|\, I^{0}) = \sum_{i,j,k} w_{ijk} \left[I_{ijk} \log \frac{I_{ijk}}{I^{0}_{ijk}} - I_{ijk} + I^{0}_{ijk} \right].$$

Each iterative adjustment of a particular margin corresponds to projecting the current array onto the associated constraint set in the space of positive arrays, ensuring consistency while maintaining multiplicative relationships with the seed. From this perspective, WIPF can be seen as an alternating projection method, where the weighted KL divergence defines the geometry of the space. Convergence follows from general results on cyclic Bregman projections onto convex sets (Censor and Zenios, 1997), which apply because the weighted marginal sets are convex and compatible. This interpretation provides both a geometric intuition for convergence and a theoretical framework linking WIPF with a broader class of iterative projection methods in convex analysis.

4.4. Computational considerations

From an algorithmic standpoint, WIPF performs multiplicative updates across the entries of the array in a manner analogous to classical IPF. Consequently, the computational complexity of WIPF can be assessed drawing on the existing IPF literature. Each iteration requires $\mathcal{O}(T \cdot M)$ operations, where T is the total number of entries in the array and M is the number of marginal constraints applied per iteration (Bishop et al., 2007). Although the calculation of weighted marginal sums introduces additional per-cell multiplications compared to classical IPF, this only affects the constant factor and does not change the asymptotic complexity.

The number of iterations needed for convergence depends on the initial seed, the weight distribution, and the chosen tolerance, but is typically modest when the margins are compatible. These considerations suggest that WIPF is computationally tractable for most practical multiway arrays encountered in applied settings, with solutions typically obtained within a few seconds on a standard laptop, even for problems involving thousands of entries and dozens of constraints.

5. The `WIPF` package

The `WIPF` package provides a general and flexible implementation of the weighted iterative proportional fitting (WIPF) algorithm for arrays of arbitrary dimension, together with a set of auxiliary functions intended to facilitate its practical use. Its functions

operate on an initial seed array and an array of non-negative weights of the same dimension. Given a set of target margins—possibly of different orders and with some margins missing—the algorithm iteratively updates the seed so that the weighted sums over the specified dimension index subsets match the supplied margins, whenever these are compatible with the weights. The implementation supports arbitrary combinations of one-dimensional, two-dimensional, and higher-order margins, making it suitable for high-dimensional applications.

The core functionality of the package is the `WIPF` function, which applies the algorithm to arrays of dimension $N \geq 2$. The one-dimensional case is not handled by `WIPF`, as it reduces to a trivial rescaling problem; this setting is instead covered by the function `WIPF1`. For convenience and computational efficiency, dedicated shortcuts are also provided for the two- and three-dimensional cases through the functions `WIPF2` and `WIPF3`, respectively. These functions follow the same logic as the general routine but exploit the specific structure of low-dimensional arrays, requiring simpler inputs.

When the supplied margins are mutually incompatible given the weights, the package includes procedures to restore compatibility prior to the main fitting step. Such adjustments are not performed silently: the user is informed through a warning message whenever they occur, and the full output of the functions report the magnitude of the corresponding margin corrections. In such cases, lower-dimensional WIPF routines are applied recursively to adjust the margins themselves, following a well-defined updating order based on the dimensionality and ordering of the associated index sets. Lower-dimensional margins are made compatible first, with row-margins taking preference over column-margins, which in turn take precedence over layer-margins, and so on. This strategy ensures that the algorithm remains well-defined and convergent even when the initial margin specifications cannot be jointly satisfied. Convergence is ensured because each pre-adjustment applies `WIPF` to a lower-dimensional problem, which inherits the same theoretical properties described in Section 4.

As one of the intended uses of WIPF is the reconciliation of statistical indices, it is not uncommon to encounter slightly incompatible margins due to statistical uncertainty or, as illustrated in the example presented in the next section, to rounding in both the weights and the initial estimated indices. In this context, the availability of automated compatibility adjustments is particularly valuable. Furthermore, when the weighted sum of one-dimensional marginal indices does not equal one, `WIPF1` can be used to enforce this condition, since margins of dimension zero are not allowed in `WIPF`. The functions also incorporate utilities to check dimensional consistency across inputs and to assess margin deviations on convergence.

To support data manipulation, the package includes auxiliary functions to convert between array- and tabular-based representations. In particular, `array2df` transforms an N-dimensional array into a long-format data frame, while `df2array` performs the inverse operation, reconstructing an array from a data frame with factor columns and a column of values. These utilities are especially useful when seed arrays or margins are obtained from external data sources in tabular form, or when `WIPF` outputs need to be exported to other environments in tabular form.

The inputs required by the `WIPF` function are summarized in Table 3. These include an initial seed array, the associated weights, a list of target margins, and a set of indices identifying the dimensions to which each margin corresponds. Additional arguments control normalization, convergence tolerance, and the maximum number of iterations.

Table 3. *Inputs of the `WIPF` function.*

Argument	Description
`seed`	An *N*-dimensional array of non-negative values providing the initial configuration.
`weights`	An *N*-dimensional array of non-negative weights associated with the entries of `seed`.
`margins`	A list of lower-dimensional arrays containing the target weighted margins.
`indices`	A list of index vectors identifying the dimensions corresponding to each element of `margins`.
`normalize`	Logical indicator specifying whether weights are normalized before computing weighted sums.
`tol`	Convergence tolerance for the iterative algorithm.
`maxit`	Maximum number of allowed iterations.
`full`	Logical indicator specifying whether the more complete output should be saved.

Source: compiled by the authors.

Table 4. *Outputs of the `WIPF` function.*

Output	Description
`sol`	An array with the same dimension as `seed`, containing the solution at convergence (or at the final iteration).
`iter`	Number of iterations performed by the algorithm.
`margins`	A list of arrays with the margins effectively used to reach the solution; margins that are compatible with the weights coincide with the original inputs.
`dev.margins`	A list of arrays, structured as `margins`, containing the maximum absolute deviations between target margins and the corresponding weighted sums of `sol`.
`dev.congruence`	A list of arrays, structured as `margins`, containing the differences between the original target margins and the adjusted margins actually used in the fitting procedure.
`inputs`	A list collecting all input objects used in the call to `WIPF`.

Source: compiled by the authors.

The full output of the function (see Table 4) includes the fitted array, the number of iterations required for convergence, and detailed information on the final deviations: between the original and (if necessary) adjusted target margins and between the margins finally used as constraints and corresponding weighted sums implied by the fitted array.

In short, the `WIPF` functions are designed to work directly with multidimensional arrays and collections of lower-dimensional margins, following the theoretical framework described in the previous sections. Taken together, these components make the `WIPF` package a flexible and transparent tool for applying weighted proportional fitting methods in multidimensional settings.

6. `WIPF` in action: adjusting death-risk indices

To illustrate the procedure and the use of the package, we adjust raw initial estimates of risk indices for Spanish males aged 70, cross-classified by four levels of habitat size (`H1`, `H2`, `H3`, `H4`) and four levels of contextual wealth (`W1`, `W2`, `W3`, `W4`). Table 5 reports the initial and WIPF-adjusted estimates in the left and right panels, respectively, rounded to four decimal places. The central panel provides the weights used in the adjustment process.

Box 1 presents the reproducible code, which starts by installing the package (line 1) and loading it in the active R session (line 2). The `WIPF` package is available under the General Public License (GPL ≥ 2) on the Comprehensive R Archive Network (CRAN) at `<https://CRAN.R-project.org/package=WIPF>`.

Since, due to rounding, the initial row and column marginal death-risk estimates are incompatible with the weights as their weighted sums do not match unity exactly, WIPF in one dimension (`WIPF1`) is first applied to adjust the margins to unitary weighted sums (see lines 7 and 8 in Box 1). Subsequently, two-dimensional WIPF (`WIPF2`) is used to obtain fully compatible two-factor death-risk indices (line 9).

More generally, in practical applications we recommend performing the adjustments hierarchically, as implemented by default in the `WIPF` functions: first ensuring that the weighted sum of first-order estimates equals the dimension-one margins; next, adjusting second-order estimates using the corrected first-order values; and proceeding analogously for higher-order terms.

After this process, final estimates of the marginal and joint risk indices for habitat size and contextual wealth are obtained (see Table 5). For instance, the final estimates indicate that among men aged 70, and relative to the average, the wealthiest men living in rural areas exhibit a mortality risk that is almost 18% lower, whereas the poorest men in urban areas face a risk that is more than 22% higher.

To quantify the magnitude of the adjustment introduced by WIPF in this example, we compare the raw and adjusted indices. The relative changes are generally small, with an average absolute relative change of about 1.15%. The largest adjustment corresponds to a relative increase of approximately 3.53%, while most entries change by less than 1.30%. These results indicate that the procedure preserves the overall structure of the

original indices while ensuring compatibility with the imposed margins. Naturally, the magnitude of the adjustments is case-dependent and reflects how compatible the original indices are with the marginal constraints given the weights.

Code example

```
install.packages("WIPF")
library(WIPF)
# Initial row margins
row.margin0 <- c(0.9687, 1.0286, 1.0097, 0.9921)
# Initial column margins
col.margin0 <- c(1.1388, 1.0366, 0.9702, 0.8549)
# Initial inner indices:  seed
cross0 <- matrix(c(1.0487, 0.9835, 0.9175, 0.8297,
                   1.1123, 1.0653, 0.9949, 0.8733,
                   1.1986, 1.0461, 0.9773, 0.8517,
                   1.1812, 1.0594, 1.0023, 0.8578),
                   nrow = 4, byrow = TRUE)
# Weights
weights <- matrix(c(140378, 149253, 112978, 54399,
                    142328, 149483, 114749, 80757,
                    133018, 120618, 133842, 150811,
                    67276, 74263, 91499, 213701),
                    nrow = 4, byrow = TRUE)
# Adjustment of margins, to unitary weighted sum
row.margin <- WIPF1(seed = row.margin0,
                    weights = rowSums(weights),
                    margin = 1)
col.margin <- WIPF1(seed = col.margin0,
                    weights = colSums(weights),
                    margin = 1)
# Adjustment of inner indices
cross <- WIPF2(seed = cross0, weights = weights,
               margin1 = row.margin,
               margin2 = col.margin)
```

Box 1: Reproducible code for the example developed in Table 5.

For clarity of exposition, the numerical example presented here focuses on the two-dimensional case. Although the motivating application is inherently multidimensional, displaying higher-dimensional contingency structures within the constraints of a two-dimensional page would require large and cumbersome tables that may obscure the main ideas. Hence, interested readers are referred to the accompanying documentation of the functions `WIPF` and `WIPF3` in the package (Pavía, 2026) for illustrative examples of three-dimensional applications of the algorithm.

Table 5. *Example of adjusted death-risk indices using WIPF for Spanish males aged 70.*

	Raw initial estimates					Weights				Final adjusted estimates				
	W1	**W2**	**W3**	**W4**		**W1**	**W2**	**W3**	**W4**	**W1**	**W2**	**W3**	**W4**	
H1	1.0487	0.9835	0.9175	0.8297	0.9687	140378	149253	112978	54399	1.0563	0.9818	0.9115	0.8201	0.9680
H2	1.1123	1.0653	0.9949	0.8733	1.0286	142328	149483	114749	80757	1.1189	1.0621	0.9871	0.8621	1.0279
H3	1.1986	1.0461	0.9773	0.8517	1.0097	133018	120618	133842	150811	1.2066	1.0437	0.9703	0.8414	1.0090
H4	1.1812	1.0594	1.0023	0.8578	0.9921	67276	74263	91499	213701	1.2229	1.0871	1.0235	0.8715	0.9914
	1.1388	1.0366	0.9702	0.8549						1.1393	1.0371	0.9707	0.8553	

Source: compiled by the authors using data from Spain between 2010 and 2019.

Note: The final solution reflects adjustments to both the marginal and inner indices. The indices were adjusted hierarchically: first, the initial row and column marginal death-risk estimates were scaled to yield unitary weighted sums; then, the inner estimates were adjusted conditionally on these corrected margins.

7. Conclusions

The iterative proportional fitting (IPF) procedure is a widely-established algorithm for adjusting an initial cross-classification of counts to align with specified target marginal distributions while preserving the original association structure. Despite numerous generalizations of IPF proposed over time, a method that allows differential weighting of cells has been lacking. This paper addresses this gap by introducing the Weighted Iterative Proportional Fitting (WIPF) algorithm and implementing it in an R package of the same name. This novel approach can be applied to adjust an initial set of indices/indicators subject to weighted sum-convex constraints across marginal dimensions.

The WIPF algorithm can be applied to adjust multidimensional sets of indices or indicators while preserving marginal consistency. For example, it can be used to reconcile economic indicators defined across multiple dimensions, such as region and demographic group, or to harmonize poverty measures disaggregated simultaneously by region and age group.

Another possible application arises in consumer choice modeling (Qu et al., 2025), where data can be arranged in a two-dimensional array representing quantities of products purchased across different consumer groups. This array may need to be adjusted to match a pre-specified marginal consisting of total product sales. In this context, weights can represent product prices, and the balancing procedure ensures consistency between observed quantities, prices, and aggregate spending patterns.

An important avenue for future research concerns the systematic study of the statistical properties of the WIPF procedure. In many practical applications, the initial seed arrays used as inputs are themselves estimated from data and therefore subject to sampling variability. Understanding how such uncertainty propagates through the WIPF adjustment, and how it affects the resulting reconciled indices, deserves further investigation. In particular, simulation-based studies could explore how the performance of WIPF varies with the number of dimensions, the number of categories across margins, the structure of the weights, and the magnitude of sampling variability in the initial

indices. Such analyses would provide valuable insights into the bias, variance, and robustness properties of the method in different empirical settings.

Another promising direction concerns the extension of the method to settings with missing or unreliable cells in the initial arrays. In many empirical contexts, some cells may be unobserved or estimated with different levels of reliability. Incorporating cell-specific measures of uncertainty or confidence into the adjustment procedure could allow the algorithm to regulate the magnitude of changes across cells according to the quality of the underlying information. Developing such reliability-aware reconciliation procedures represents a natural extension of the WIPF framework.

Appendix A. Numerical counterexample on the non-equivalence between recast IPF and WIPF

A natural question is whether the weighted iterative proportional fitting (WIPF) procedure proposed in this paper can be reproduced by applying the classical IPF algorithm to a suitably transformed problem. In particular, one might conjecture that the weighted problem can be converted into a standard IPF problem by redefining the seed as the elementwise product of the original seed and the weights, and redefining the margins as the product of the initial target margins and the corresponding partial sums of weights.

To illustrate the non-equivalence between the two approaches, consider the following simple numerical example. Let the seed matrix and the weights be

$$S = \begin{pmatrix} 0.8 & 1.3 \\ 1.6 & 0.9 \end{pmatrix}, \qquad W = \begin{pmatrix} 500 & 9000 \\ 8000 & 1000 \end{pmatrix}.$$

Suppose that the weighted row and column constraints are both equal to the vector $(1,1)$. Then the solutions obtained using the recast IPF formulation and the WIPF procedure (rounded to four decimals) are

$$\hat{S}_{\text{rIPF}} = \begin{pmatrix} 0.0133 & 0.5002 \\ 0.4462 & 0.0403 \end{pmatrix}, \qquad \hat{S}_{\text{WIPF}} = \begin{pmatrix} 0.4922 & 1.0282 \\ 1.0317 & 0.7461 \end{pmatrix}.$$

The two fitted matrices are clearly different. This simple example therefore shows that WIPF cannot, in general, be reproduced by applying classical IPF to a reweighted seed and correspondingly transformed margins.

The results can be easily verified using the reproducible code provided in Box 2.

Reproducible code Appendix A

```
# Data
seed <- matrix(c(0.8, 1.3, 1.6, 0.9),
                  ncol = 2, byrow = TRUE)
weights <- matrix(c(500, 9000, 8000, 1000),
                     ncol = 2, byrow = TRUE)
margin.c <- margin.r <- c(1, 1)
weights <- weights/sum(weights)

# IPF recast solution
install.packages("mipfp")
library(mipfp)
sol.rIPF <- Ipfp(seed * weights,
                    target.list = list(1, 2),
                    target.data = list(margin.r *
                         rowSums(weights),
                         margin.c * colSums(weights)) )
round(sol.rIPF[[1]], 4)

# WIPF solution
install.packages("WIPF")
library(WIPF)
sol.WIPF <- WIPF2(seed, weights, margin.r,
                      margin.c)
round(sol.WIPF, 4)
```

Box 2: Reproducible code for the example developed in Appendix A.

Appendix B. Numerical illustration: Weighted KL formulation of WIPF

This appendix illustrates, using the example introduced in Appendix A, that the weighted iterative proportional fitting (WIPF) solution can be obtained, when no null indices are involved, as the solution of a weighted Kullback–Leibler (KL) minimization problem whose objective function is given by

$$\min_{I_{ij}>0} \sum_{i,j} w_{ij} I_{ij} \log \frac{I_{ij}}{I^0_{ij}},$$

subject to the corresponding weighted marginal constraints. In our example, these constraints are $(I_{11}w_{11}+I_{12}w_{12})/(w_{11}+w_{12})=1$, $(I_{21}w_{21}+I_{22}w_{22})/(w_{21}+w_{22})=1$, $(I_{11}w_{11}+I_{21}w_{21})/(w_{11}+w_{21})=1$, and $(I_{12}w_{12}+I_{22}w_{22})/(w_{12}+w_{22})=1$.

This result can be easily illustrated using the reproducible code provided in Box 3, which solves the corresponding KL-based optimization problem in R using the `Rsolnp` package.

Reproducible code Appendix B

```
# Package
install.packages("Rsolnp")
library(Rsolnp)

# Data
I0v <- c(0.8, 1.3, 1.6, 0.9)
wv <- c(500, 9000, 8000, 1000)
wv <- wv / sum(wv)

# Objective function
fobj <- function(x) sum(wv * x * log(x/I0v))

# Constraints
heq <- function(x){
 c( (wv[1]*x[1] + wv[2]*x[2]) / sum(wv[1:2]) - 1,
    (wv[3]*x[3] + wv[4]*x[4]) / sum(wv[3:4]) - 1,
    (wv[1]*x[1] + wv[3]*x[3]) / sum(wv[c(1,3)]) - 1) }

# Solution
sol <- Rsolnp::solnp(
 pars = c(0.5,0.5,0.5,0.5),
 fun = fobj,
 eqfun = heq,
 eqB = rep(0,3),
 LB = rep(1e-12,4) )

matrix(round(sol[[1]], 4), 2, 2)
```

Box 3: Reproducible code for the example formulated as a weighted KL optimization problem.

Appendix C. Effect of weight normalization on WIPF solutions

In the WIPF algorithm, the choice of normalized versus non-normalized weights systematically affects the resulting solution. Normalized weights ensure that, for each marginal dimension, the sum of the weights across the missing dimensions equals one. In this formulation, the target margins are preserved exactly in the scale of the seed array.

Non-normalized weights retain the original magnitudes of the weights. In this case, the resulting solution generally differs in scale from the normalized version: while the

relative distribution of values within each marginal remains consistent with the weight ratios, the absolute magnitudes of the fitted entries may change. Equivalence up to a simple rescaling does not generally hold unless all weights along a given marginal are proportional. Therefore, practitioners should be aware that the `normalize` argument in the R implementation can produce genuinely different solutions.

In practice, the normalized formulation is recommended when the goal is to maintain comparability with classical IPF and to preserve the interpretability of the fitted array in the same units as the seed. Conversely, the non-normalized formulation may be preferable when absolute magnitudes need to reflect the raw weights.

To illustrate the non-equivalence up to a simple rescaling, we consider the same small numerical example analyzed in the previous appendixes and compare the WIPF solutions obtained with normalized and non-normalized weights.

As shown by the solutions of the example, the two solutions are not proportional, highlighting the practical impact of the normalization choice.

$$\hat{S}_{\text{nWIPF}} = \begin{pmatrix} 1.7293 & 1.9595 \\ 2.2044 & 0.8647 \end{pmatrix}, \qquad \hat{S}_{\text{WIPF}} = \begin{pmatrix} 0.4922 & 1.0282 \\ 1.0317 & 0.7461 \end{pmatrix}.$$

The corresponding R code to reproduce both solutions (normalized and non-normalized) is provided in Box 4.

Reproducible code Appendix C

```
# Data
seed <- matrix(c(0.8, 1.3, 1.6, 0.9),
                  ncol = 2, byrow = TRUE)
weights <- matrix(c(500, 9000, 8000, 1000),
                      ncol = 2, byrow = TRUE)
margin.c <- margin.r <- c(1, 1)
weights <- weights/sum(weights)


# WIPF non-normalized solution
install.packages("WIPF")
library(WIPF)
sol.nWIPF <- WIPF2(seed, weights, margin.r,
                       margin.c, normalize = FALSE)
round(sol.nWIPF, 4)


# WIPF normalized solution
sol.WIPF <- WIPF2(seed, weights, margin.r,
                       margin.c, normalize = TRUE)
round(sol.WIPF, 4)
```

Box 4: Reproducible code for the example developed in Appendix C.

Acknowledgements

The authors wish to thank two anonymous reviewers and the editor for valuable comments and suggestions, and Marie Hodkinson for revising the English of the paper. The usual disclaimers apply.

Funding

This work was supported by Generalitat Valenciana (Conselleria de Educación, Cultura y Universidades) under Grant CIACIO/2023/031; and Fundación MAPFRE under Grant "Modelización espacial e intra-anual de la mortalidad en España. Una herramienta automática para el cálculo de productos de vida".

Disclosure statement

The authors report there are no competing interests to declare.

References

Allen-Zhu, Z., Li, Y., Oliveira, R. and Wigderson, A. (2017). Much faster algorithms for matrix scaling. *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, 890–901.

Bell, M. L., O'Neill, M. S. and Zanobetti, A. (2024). Climate change, extreme heat, and health. *New England Journal of Medicine* 390, 1793–1801.

Bishop, Y., Fienberg, S. E. and Holland, P. W. (2007). *Discrete Multivariate Analysis: Theory and Practice*. Springer.

Censor, Y. and Zenios S. A. (1997). *Parallel Optimization: Theory, Algorithms, and Applications*. Oxford University Press.

Cohen, S. A., Nash, C. C., Byrne, E. N. and Greaney, M. L. (2023). Income and rural-urban status moderate the association between income inequality and life expectancy in US census tracts. *Journal of Health, Population and Nutrition* 42, 24.

Coons, J. Y., Langer, C. and Ruddy, M. (2024). Classical iterative proportional scaling of log-linear models with rational maximum likelihood estimator. *International Journal of Approximate Reasoning* 164, 109043.

Deming, W. E. and Stephan, F. F. (1940). On a least squares adjustment of a sampled frequency table when the expected marginal totals are known. *The Annals of Mathematical Statistics* 11, 427–444.

Deville, J. C., Särndal, C.-E. and Sautory, O. (1993). Generalized raking procedures in survey sampling. *Journal of the American Statistical Association* 88, 1013–1020.

Dwyer-Lindgren, L., Baumann, M. M., Li, Z., Kelly, Y. O., Schmidt, C., Searchinger, C., La Motte-Kerr, W., Bollyky, T. J., Mokdad, A. H. and Murray, C. J. (2024). Ten

Americas: A systematic analysis of life expectancy disparities in the USA. *The Lancet* 404(10469), 2299–2313.

European Commission (2012). Guidelines on the application of Council Directive 2004/113/EC to insurance, in the light of the judgment of the Court of Justice of the European Union in Case C-236/09 (Test-Achats). *Official Journal of the European Union* C, 1–11.

Fournier Gabela, J. G. (2020). On the accuracy of gravity-RAS approaches used for inter-regional trade estimation: evidence using the 2005 inter-regional input–output table of Japan. *Economic Systems Research* 32, 521–539.

García-León, D., Masselot, P., Mistry, M. N., Gasparrini, A., Motta, C., Feyen, L. and Ciscar, J. C. (2024). Temperature-related mortality burden and projected changes under climate change scenarios. *The Lancet Public Health* 9, e644–e653.

Idel, M. (2016). A review of matrix scaling and Sinkhorn's normal form for matrices and positive maps. *arXiv preprint* arXiv:1609.06349.

Kalantari, B. and Khachiyan, L. (1996). On the complexity of nonnegative-matrix scaling. *Linear Algebra and its Applications* 240, 87–103.

Kurras, P., Greenewald, K. and Grosse, R. (2015). A simple and provably good projection update for iterative proportional fitting. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 38, 570–578. PMLR.

Klimova, A. and Rudas, T. (2015). Iterative proportional scaling for curved exponential families. *Scandinavian Journal of Statistics* 42, 832–847.

Kruithof, J. (1937). Telefoonverkeersrekening. *De Ingenieur* 52, E15–E25.

Lagravinese, R., Liberati, P. and Resce, G. (2020). Measuring health inequality in US: A composite index approach. *Social Indicators Research* 147, 921–946.

Lledó, J. and Pavía, J. M. (2026). Integrating income-indexes into life insurance and financial products. *Financial Innovation* 12, 94.

Masselot, P., Mistry, M. N., Rao, S., Huber, V., Monteiro, A., Samoli, E., Stafoggia, M., de'Donato, F., Garcia-Leon, D., Ciscar, J. C., Feyen, L., Schneider, A., Katsouyanni, K., Vicedo-Cabrera, A. M., Aunan, K. and Gasparrini, A. (2025). Estimating future heat-related and cold-related mortality under climate change, demographic and adaptation scenarios in 854 European cities. *Nature Medicine* 31, 1294–1302.

McMaughan, D. J., Oloruntoba, O. and Smith, M. L. (2020). Socioeconomic status and access to healthcare: Interrelated drivers for healthy aging. *Frontiers in Public Health* 8, 231.

Pavía, J. M. (2026). *WIPF: Weighted Iterative Proportional Fitting*. R-package, version 0.1.0-4. `https://CRAN.R-project.org/package=WIPF`.

Pavía, J. M., Cabrer, B. and Sala, R. (2009). Updating Input-Output matrices: Assessing alternatives through simulation. *Journal of Statistical Computation and Simulation* 79, 1467–1498.

Pavía, J. M. and Lledó, J. (2022). Estimation of the combined effects of ageing and seasonality on mortality risk: An application to Spain. *Journal of the Royal Statistical Society Series A: Statistics in Society* 185(2), 471–497.

Pavía, J. M., Lledó, J., Espinosa, P. and Pérez, V. (2026). *Modelización espacial e intra-anual de la mortalidad en España. Una herramienta automática para el cálculo de productos de vida*. Madrid: Fundación Mapfre.

Pavía, J. M., Lledó, J. and Roig, R. (2026). The cost of inequality: Income and life expectancy at birth in Spain. *Health Policy* 169, 105616.

Qu, Z., Galichon, A., Gao, W. and Ugander, J. (2025). On Sinkhorn's algorithm and choice modeling. *Operations Research*, online available. 10.1287/opre.2023.0596.

Suesse, T., Namazi-Rad, M. R., Mokhtarian, P. and Barthélemy, J. (2017). Estimating cross-classified population counts of multidimensional tables: An application to regional Australia to obtain pseudo-census counts. *Journal of Official Statistics* 33, 1021–1050.

von Lindheim, J. and Steidl, G. (2023). Generalized iterative scaling for regularized optimal transport with affine constraints: Application examples. *arXiv preprint* arXiv:2305.07071v1.

Wiebe, K. S. and Lenzen, M. (2016). To RAS or not to RAS? What is the difference in outcomes in multi-regional input–output models? *Economic Systems Research* 28, 383–402.

Zaloznik, M. (2011). *Iterative Proportional Fitting. Theoretical Synthesis and Practical Limitations*. PhD dissertation, University of Oxford.

SORT 49 (1) January-June 2025, 89-116 DOI: 10.57645/20.8080.02.33

A gentle introduction to deep neural networks for operations researchers

Pau Amaré[1] and Jordi Castro[2,3,4]

Abstract

Computing a neural network is, in essence, a large unconstrained optimization problem, though this fact is often obscured by machine learning jargon. We present a gentle introduction to deep learning for operations researchers, describing in a didactic manner the underlying optimization problem and providing examples developed from scratch. These examples are solved using both standard modeling languages and modern machine learning frameworks. We conclude by discussing applications of neural networks in operations research, illustrated through a concrete example involving the knapsack problem.

MSC: 90C30, 90C90.

Keywords: Operations research, nonlinear optimization, deep learning, neural networks, optimization modeling languages, machine learning frameworks.

1. Introduction

Neural networks (NNs) are one of the main tools in machine learning and data science, and they form the core computational model behind modern large language models (LLMs), such as ChatGPT. As we will show, computing a NN reduces to solving a very large unconstrained optimization problem. However, this fact is not always apparent from the notation and jargon used in machine learning. In this work, we aim to fill this gap by providing a gentle introduction to (deep) NNs for operations researchers.

[1] School of Mathematics and Statistics, Universitat Politècnica de Catalunya, Barcelona, Catalonia.

[2] Department of Statistics and Operations Research, Universitat Politècnica de Catalunya, Barcelona, Catalonia.

[3] Institute of Mathematics of the Universitat Politècnica de Catalunya (IMTech), Barcelona, Catalonia.

[4] Corresponding author

Received: January 2026
Accepted: March 2026

A NN can be defined as a function $y = f(x;\theta)$ that depends on a set of parameters $\theta \in \mathbb{R}^n$. Its purpose is to model the nonlinear relationship between the input and output vectors x and y. If n is large enough, any continuous function can be approximated by a NN; this result, known as the *universal approximation theorem*, was proven in Cybenko (1989); Hornik, Stinchcombe and White (1989). For instance, the number of parameters in modern NNs underlying LLMs is on the order of billions, and there are estimates that the latest version of ChatGPT ranges from one to a few trillion parameters. Therefore, to find the best values of θ, a huge unconstrained optimization problem must be solved.

There are excellent monographs on deep learning and NNs, focusing both on methodology (Goodfellow, Bengio and Courville, 2016) and software (Chollet, 2017), as well as surveys and monographs on optimization methods for NNs (Bottou, Curtis and Nocedal, 2018; Sra, Nowozin and Wright, 2011). Unlike those references, this manuscript is intended for an audience seeking a quick, gentle, and practical introduction to NNs that combines theory with examples developed from scratch and solved using both standard tools in operations research (e.g., the AMPL modeling language (Fourer, Gay and Kernighan, 2003)) and state-of-the-art machine learning frameworks (e.g., Google's TensorFlow (Abadi et al., 2016)).

This work focuses on two of the simplest types of NNs: feedforward and convolutional neural networks (CNNs). For more advanced NN functions (such as encoder–decoder models and transformers, which underlie LLMs), we refer the reader to Goodfellow et al. (2016); Sutskever, Vinyals and Le (2014); Vaswani et al. (2017).

Although NNs rely on optimization methods to compute their best parameters, once trained they can be applied to approximate, accelerate, or guide traditional operations research (OR) algorithms. The survey Bengio, Lodi and Prouvost (2021) reviews recent applications of machine learning techniques to combinatorial optimization problems. A short discussion of these topics is provided in the last part of the paper, together with an illustrative example involving the knapsack problem.

The remainder of the paper is structured as follows. Section 2 describes the structure of a feedforward NN. Section 3 presents two illustrative applications of feedforward NNs with different model complexity. Section 4 introduces CNNs, which are particularly well suited for image data. Section 5 presents an application of CNNs. Finally, Section 6 discusses applications of NNs in OR.

2. Feedforward neural networks

Mathematically, a NN can be defined as a function

$$f : \mathbb{R}^d \to \mathbb{R}^k$$

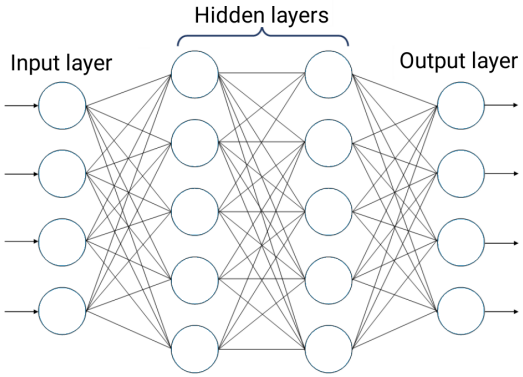

Figure 1. *Example of a 4-layer feedforward NN.*

that models the nonlinear relationship between an input vector $x \in \mathbb{R}^d$ and an output vector $y \in \mathbb{R}^k$. In other words,

$$y = f(x;\theta), \tag{1}$$

where $\theta \in \mathbb{R}^n$ denotes the set of parameters that determine the specific form of the function. A set of points $\{(x^i \in \mathbb{R}^d, y^i \in \mathbb{R}^k), i = 1,\dots,p\}$, called the *training set* in machine learning jargon, is used to compute the best θ through an optimization procedure that minimizes the discrepancies between $f(x^i;\theta)$ and y^i. Another set of points, called the *testing set*, is used to evaluate the accuracy of the computed parameter values θ.

The NN is composed of several elements, which are described in the following subsections.

2.1. Nodes and layers

A NN is composed of a number of *nodes* distributed across several layers. The number of layers, l, determines the depth of the network. We denote by d_j the number of nodes in layer j, for $j = 1,\dots,l$. When the number of layers l is large, the network is said to be deep, and we then speak of *deep learning*.

The first layer is called the *input layer*. This layer simply receives the input and passes it to the next layer (possibly reshaping it when the input is not already a vector). It consists of $d_1 = d$ nodes, matching the dimension of the input vector x or of its reshaped version. The layers from the second to the penultimate are called *hidden layers* (or intermediate layers). They analyze and process the data, and then send it to the next layer. The last layer is the *output layer*, which provides the final result of the network's computation. It consists of $d_l = k$ nodes, corresponding to the size of the network's output vector.

Nodes within the same layer are not connected to each other, but nodes in one layer may be connected to all the nodes of the next layer. This particular structure is known as a *feedforward* NN. Figure 1 shows an example of a 4-layer feedforward NN.

2.2. Parameters and mathematical formulation

Connections between nodes of different layers are represented by arcs joining pairs of nodes in consecutive layers. Each arc connecting node u in layer $j-1$ with node v in layer j, for $j \geq 2$, has an associated *weight* $w_{uv} \in \mathbb{R}$. These arcs are analogous to the synapses linking neurons in the nervous system. In addition, each node v also has a parameter $b_v \in \mathbb{R}$, called the *bias*. Nodes in the first layer usually do not have a bias.

Any node v in layer $j \geq 2$ receives the sum of the output values of all nodes from the previous layer $j-1$, each multiplied by the corresponding weight, plus the bias term b_v. Denoting by $x_h^{(j-1)}$, $h = 1, \ldots, d_{j-1}$, the output values of the nodes in layer $j-1$, the result $z_v^{(j)}$ of this linear combination is

$$z_v^{(j)} = \sum_{h=1}^{d_{j-1}} w_{u_h v} x_h^{(j-1)} + b_v, \quad j = 2, \ldots, l,$$

where (u_h, v) denotes the arc connecting the h-th node of layer $j-1$ with node v in layer j. We can compute the vector $z^{(j)}$ of these linear combinations for all nodes v in an arbitrary layer j as

$$z^{(j)} = W^{(j)} x^{(j-1)} + b^{(j)}, \quad j = 2, \ldots, l, \tag{2}$$

where $W^{(j)} \in \mathbb{R}^{d_j \times d_{j-1}}$ are the weights between layers $j-1$ and j, $b^{(j)} \in \mathbb{R}^{d_j}$ are the biases of layer j, and $x^{(j-1)}$ is the output vector of the nodes in layer $j-1$, for $j = 2, \ldots, l$. The vector $x^{(0)}$ is equal to the input x, the point at which the NN is evaluated. The output of the first layer, $x^{(1)}$, is equal to $x^{(0)}$ if layer 1 is the identity function; otherwise, $x^{(1)}$ is a reshaped version of $x^{(0)}$. In any case, layer 1 does not introduce any parameters into the NN. Hence, the parameters of the NN are $\theta = (W^{(j)}, b^{(j)}, j = 2, \ldots, l) \in \mathbb{R}^n$, with $n = \sum_{j=2}^{l} (d_j d_{j-1} + d_j)$.

The values $z^{(j)}$ must be nonlinearly transformed before being sent to the nodes of the next layer $j+1$. Otherwise, the overall NN would be just a composition of linear functions, hence linear, and could not model nonlinear relationships between x and y. For this purpose, a nonlinear function, called the *activation function*,

$$g^{(j)} : \mathbb{R} \to \mathbb{R},$$

is applied to all components of $z^{(j)}$, so that the output values $x^{(j)}$ at the nodes of layer $j \geq 2$ are

$$x^{(j)} = g^{(j)}(z^{(j)}) = g^{(j)}\left(W^{(j)} x^{(j-1)} + b^{(j)}\right) = \varphi^{(j)}(x^{(j-1)}), \tag{3}$$

that is, each layer $j \geq 2$ of the NN is a function $\varphi^{(j)} : \mathbb{R}^{d_{j-1}} \to \mathbb{R}^{d_j}$, defined as

$$x^{(j)} = \varphi^{(j)}(x^{(j-1)}) = g^{(j)}\left(W^{(j)} x^{(j-1)} + b^{(j)}\right) \quad j = 2, \ldots, l. \tag{4}$$

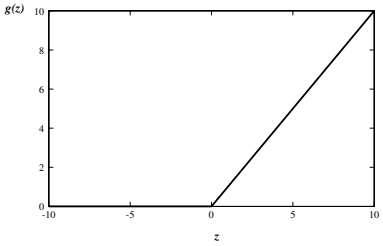

Figure 2. *ReLU.*

Therefore, the NN can be viewed as the composition of these functions:

$$f(x;\theta) = \varphi^{(l)}\big(\varphi^{(l-1)}(\dots\varphi^{(2)}(\varphi^{(1)}(x))\dots)\big), \tag{5}$$

where $\varphi^{(1)}$ is either the identity function or a function that reshapes the input (e.g., flattening a matrix or tensor into a vector). The next subsection provides an overview of several activation functions frequently employed in NNs.

2.3. Activation functions

As stated above, *activation functions* introduce nonlinearity at each level of the NN. The activation function of layer j is denoted by $g^{(j)}$. Each layer may use a different activation function, although it is common to employ the same function for all hidden layers, one for the input layer, and another for the output layer. Among the most commonly used activation functions are the following:

Identity. This activation function is

$$g(z) = I(z) = z, \qquad z \in \mathbb{R}, \tag{6}$$

and it is commonly used in layers with no trainable parameters.

ReLU. The *rectified linear unit* (know as ReLU) function is defined as:

$$g(z) = \mathrm{ReLU}(z) = \max(0,z) = \begin{cases} 0, \text{ if } z \le 0, \\ z, \text{ if } z > 0. \end{cases} \tag{7}$$

This function, shown in Figure 2, is not differentiable at $z = 0$. To avoid numerical issues with the optimization methods used to compute the optimal parameters θ of the NN, the derivative at 0 is defined as $g'(0) = 0$, that is

$$g'(z) = \begin{cases} 0, \text{ if } z \le 0, \\ 1, \text{ if } z > 0. \end{cases} \tag{8}$$

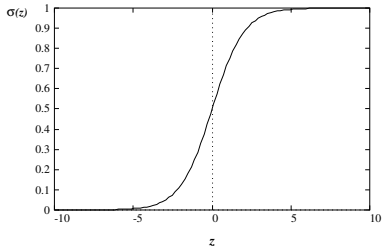

Figure 3. *Sigmoid.*

ReLU is one of the most widely used activation functions in NNs, especially in hidden layers. Since the function is zero for negative values, only a few nodes (or neurons) are activated, which makes computations more efficient. Moreover, the derivative never vanishes for positive z, improving the efficiency of the optimization methods used to compute the network parameters (which, as described later, rely on the gradient of the objective function).

Sigmoid.

The *sigmoid* or logistic function, denoted by σ, is defined as

$$\sigma(z) = \frac{1}{1+e^{-z}}, \qquad z \in \mathbb{R}. \tag{9}$$

This function, shown in Figure 3, has range $(0,1)$, is monotonically increasing, and changes very sharply in the interval $(-2,2)$, where its derivative is large. An appealing property of this function is that its derivative can be expressed directly in terms of $\sigma(z)$:

$$\sigma'(z) = \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right) = \sigma(z)(1-\sigma(z)),$$

which made it a very attractive activation function in early NNs implementations, as it simplifies computations.

Since this function returns values between 0 and 1, it is used in the output layer of models that predict probabilities, as well as in hidden layers. Its main drawback is that for large $|z|$, the derivative $\sigma'(z)$ approaches 0, slowing down the optimization algorithms used to compute the network parameters, which rely on the gradient of the objective function.

Softmax

The *softmax* function generalizes the sigmoid function to a k-dimensional vector and returns a probability distribution; therefore it depends on more than one variable. It is used in the output layer l when classifying among k classes, as it predicts

the probability associated with each class.

Unlike previous activation functions, softmax is a vector-valued function

$$g^{(l)} : \mathbb{R}^k \to [0,1]^k,$$

which maps the vector of values $z = (z_1, \ldots, z_k)$ at the nodes of the last layer. Each component $g_i^{(l)}$ is defined as

$$g_i^{(l)}(z) = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}}, \qquad i = 1, \ldots, k. \tag{10}$$

Clearly, $g_i^{(l)}(z) \in [0,1]$, for all $i = 1, \ldots, k$, and $\sum_{i=1}^k g_i^{(l)}(z) = 1$, thus providing a valid probability distribution.

2.4. Optimization problem and objective function

Given a NN with a certain number of layers and nodes, suitable activation functions, and corresponding weights and biases, together with a training set $\{(x^i \in \mathbb{R}^d, y^i \in \mathbb{R}^k), i = 1, \ldots, p\}$, for a given input vector x^i the network returns a response $f(x^i;\theta)$, which may be a scalar or a vector depending on whether the output layer has one or several nodes. This output vector depends on the network parameters θ. The goal is for $f(x^i;\theta)$ to be as close as possible to the true values y^i in the training set. When the discrepancies between $f(x^i;\theta)$ and y^i are small or close to zero, the network makes a good prediction. The function that measures these discrepancies is the objective function of the optimization problem used to compute the best parameter values θ; in machine learning jargon, this function is called the *loss function*. The optimization problem can thus be formulated as

$$\min_{\theta \in \mathbb{R}^n} F(\theta) = \frac{1}{p} \sum_{i=1}^{p} F_i(\theta), \tag{11}$$

where $F_i : \mathbb{R}^n \to \mathbb{R}$ measures the particular discrepancy between $f(x^i;\theta)$ and y^i at point i, and $F : \mathbb{R}^n \to \mathbb{R}$ computes the average overall discrepancy for all points in the training set.

Two of the most commonly used functions $F_i(\theta)$ to compute the discrepancy are:

Mean squared error. The *mean squared error* (MSE) is simply the square of the Euclidean norm of $f(x^i;\theta) - y^i$:

$$F_i(\theta) = MSE(f(x^i;\theta), y^i) = \|f(x^i;\theta) - y^i\|^2 = (f(x^i;\theta) - y^i)^\top (f(x^i;\theta) - y^i).$$

Cross-entropy. *Cross-entropy* is used when the last layer of the network has k nodes and the network output $f(x^i;\theta) \in \mathbb{R}^k$ for input x^i represents a probability distri-

bution, that is, $f(x^i;\theta)_j \geq 0$ for $j = 1,\ldots,k$ and $\sum_{j=1}^{k} f(x^i;\theta)_j = 1$. The values $f(x^i;\theta)_j$ give the probability that point i belongs to class j, which is the case when the activation function of the last layer is softmax. Cross-entropy measures the difference between the predicted distribution $f(x^i;\theta)$ and the true distribution y^i, and is defined as:

$$F_i(\theta) = H(f(x^i;\theta), y^i) = -\sum_{j=1}^{k} y^i_j \log(f(x^i;\theta)_j). \tag{12}$$

2.5. Optimization algorithms

Although both MSE and cross-entropy are convex functions with respect to $f(x^i;\theta)$, since $f(x;\theta)$, as defined in (5), is a composition of l nonlinear and nonconvex functions due to the activation functions, the objective function in (11) becomes a highly nonlinear, nonconvex function of θ. Therefore, optimization methods for solving the unconstrained problem (11) are only expected to find a local minimum rather than a global solution. Most current methods used for the optimization of (11) in state-of-the-art machine learning frameworks rely on variants of the first-order *steepest descent* method, commonly referred to as the *gradient* method, originally proposed by Cauchy (1847). Second-order methods (i.e., methods that compute or approximate second derivatives), such as Newton or limited-memory quasi-Newton methods, have also been tested, but only on small NNs (Bottou et al., 2018). Here, we focus exclusively on the gradient method.

Unconstrained optimization methods for solving (11) are iterative algorithms that generate a sequence of points $\{\theta^t\}_{t\geq 0}$ according to

$$\theta^{t+1} = \theta^t + \alpha^t \Delta\theta^t, \tag{13}$$

where $\Delta\theta^t \in \mathbb{R}^n$ is the search direction and $\alpha^t \in \mathbb{R}$ is the step length at iteration t. The step length is referred to as the *learning rate* in machine learning jargon.

Convergence to the solution θ^* is guaranteed if the objective function, the search direction, and the step length satisfy certain conditions. Among these conditions are the following: the objective function must be smooth and bounded below; the search direction must be a descent direction; and the step length must satisfy sufficient decrease conditions, known as the Armijo-Wolfe conditions (Luenberger and Ye, 2008; Nocedal and Wright, 2006). In machine learning, these conditions are only loosely enforced when solving (11) due to the very large scale of the problem, where n can be on the order of billions. In particular, the step length (learning rate) is not checked to satisfy the Armijo-Wolfe conditions and is instead adjusted heuristically.

Algorithm *Stochastic/mini-batch gradient for NNs*
Set starting point θ
Set mini-batch size $p' > 0$ for set $\mathcal{P}'$
Set T number of epochs
for $t = 1, \ldots, T$
for $j = 1, \ldots, \left\lceil \frac{p}{p'} \right\rceil$
Select subset of points $\mathcal{P}'$
Compute $\nabla F_{\mathcal{P}'}(\theta)$
Compute step length (learning rate) α
$\theta := \theta - \alpha \nabla F_{\mathcal{P}'}(\theta)$
end_for
end_for
Return: θ
End_algorithm

Figure 4. *The stochastic gradient method.*

The gradient method takes as its search direction the negative gradient of the objective (or loss) function in (11) at the current point:

$$\Delta\theta^t = -\nabla F(\theta^t) = \frac{1}{p} \sum_{i=1}^{p} \nabla F_i(\theta^t). \tag{14}$$

Computing $\nabla F(\theta^t)$ therefore requires the gradient $\nabla_\theta f(x;\theta)$ of the NN function (5) for all p points in the dataset. The challenging computation of $\nabla_\theta f(x;\theta)$ is carried out in machine learning frameworks using an algorithm known as *backpropagation*. Backpropagation is often mistaken for the optimization algorithm in NNs, whereas it is in fact the procedure used to compute the gradient.

In practice, instead of computing the p gradients $\nabla F_i(\theta^t)$, $i = 1, \ldots, p$, for the complete set of points $\mathcal{P} = \{1, \ldots, p\}$, it is common to consider only a subset $\mathcal{P}' \subseteq \mathcal{P}$ of $p' < p$ points, so that an estimate of the negative gradient is used as the search direction:

$$\Delta\theta^t = -\nabla F_{\mathcal{P}'}(\theta^t) = -\frac{1}{|\mathcal{P}'|} \sum_{i \in \mathcal{P}'} \nabla F_i(\theta^t). \tag{15}$$

In machine learning, the set $\mathcal{P}'$ is called the *mini-batch* set. When $p' = 1$ this procedure is known as the *stochastic gradient*; if $p > p' > 1$ it is called the *mini-batch gradient*; and when $p' = p$ (i.e., the classical gradient method) it is referred to as the *full-batch gradient*. The stochastic/mini-batch gradient algorithm is outlined in Figure 4. The procedure

has two main loops: the outer loop, called *epochs* in machine learning, corresponds to the iterations of the (full-batch) gradient method, while at each iteration of the inner loop the algorithm selects a subset $\mathcal{P}'$ of p' points to compute the search direction. Each epoch thus consists of $\left\lceil \frac{p}{p'} \right\rceil$ iterations, where the final mini-batch may contain fewer than p' points. The number of epochs is usually predetermined (parameter T in the algorithm of Figure 4), although early stopping may be used when the iterates stabilize.

More efficient variants of the standard algorithm in Figure 4 can be obtained either by modifying the search direction or by adapting the computation of the step length. Among the former are the *gradient with momentum* (Polyak, 1964) and the *Nesterov accelerated method* (Nesterov, 1983), which combine the current and previous gradients to determine the search direction. Among the latter, which compute adaptive step lengths, we find *AdaGrad*, *RMSProp*, and *Adam*; the last of these is one of the most widely used methods in modern machine learning frameworks (see Goodfellow et al. (2016) for details).

3. Two illustrative applications of feedforward neural networks

The next two subsections present two classic applications of feedforward NNs: the *XOR (exclusive or)* function and the *fashion-MNIST* problem. The XOR model is solved from scratch using both AMPL and Google's TensorFlow, whereas the fashion-MNIST problem, due to its greater complexity, is solved only with TensorFlow.

3.1. Neural network for XOR

The XOR problem was used in several of the seminal papers on NNs to illustrate their capabilities (Minsky and Papert, 1969; Rumelhart, Hinton and Williams, 1986). The XOR (exclusive or) is a binary operation that, given two binary inputs, returns 1 ("true") if and only if exactly one input is 1, and 0 ("false") otherwise. It is defined as XOR : $\{0,1\}^2 \to \{0,1\}$, with truth table shown below:

x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

The training set for this problem contains the four points of the domain, so $p = 4$, and the dataset is

$$\{(x^i = (x_1^i, x_2^i), y^i), i = 1, \ldots, 4\} = \{((0,0),0), ((0,1),1), ((1,0),1), ((1,1),0)\}.$$

We consider a NN with $l = 3$ layers; an input layer, one hidden layer, and an output layer. Figure 5 shows the structure of the network. The number of nodes of the first

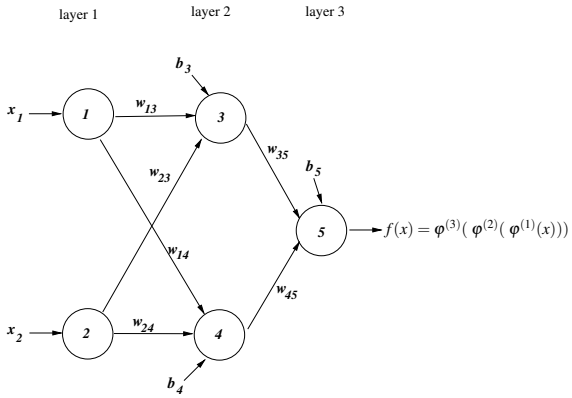

Figure 5. *XOR NN.*

(input) and third (output) layers are $d_1 = 2$ and $d_3 = 1$, matching the input and output dimensions of the XOR function. The hidden (second) layer has $d_2 = 2$ nodes. For the activation functions, we choose the identity for the first and third layers, and the sigmoid for the second layer.

The XOR NN is the composition of three functions $f(x) = \varphi^{(3)}(\varphi^{(2)}(\varphi^{(1)}(x)))$, where $x \in \mathbb{R}^2$. From (5), we have that $\varphi^{(1)} : \mathbb{R}^2 \to \mathbb{R}^2$ is the identity function, and, since $x^{(0)} = x$, it follows that

$$x^{(1)} = \varphi^{(1)}(x^{(0)}) = x^{(0)} = x, \tag{16}$$

that is, the first layer simply passes the input vector to the second layer. $f(x) = \varphi^{(3)}(\varphi^{(2)}(\varphi^{(1)}(x)))$, In the second layer, we have $\varphi^{(2)} : \mathbb{R}^2 \to \mathbb{R}^2$. Since $g^{(2)}(z) = \sigma(z)$, and the matrix of weights and vector of biases are

$$W^{(2)} = \begin{pmatrix} w_{13} & w_{23} \\ w_{14} & w_{24} \end{pmatrix}, \qquad b^{(2)} = \begin{pmatrix} b_3 \\ b_4 \end{pmatrix},$$

we obtain

$$\begin{aligned} x^{(2)} &= \varphi^{(2)}(x^{(1)}) = g^{(2)}(W^{(2)}x^{(1)} + b^{(2)}) = \begin{pmatrix} g^{(2)}(w_{13}x_1 + w_{23}x_2 + b_3) \\ g^{(2)}(w_{14}x_1 + w_{24}x_2 + b_4) \end{pmatrix} = \\ &= \begin{pmatrix} 1/(1 + e^{-(w_{13}x_1 + w_{23}x_2 + b_3)}) \\ 1/(1 + e^{-(w_{14}x_1 + w_{24}x_2 + b_4)}) \end{pmatrix}. \end{aligned} \tag{17}$$

Finally, in the third (output) layer we have $\varphi^{(3)} : \mathbb{R}^2 \to \mathbb{R}$, and, as a modeling decision, we choose the node to have no *bias*, i.e., $b_5 = 0$. Since $g^{(3)}(z) = I(z)$, and

$$W^{(3)} = \begin{pmatrix} w_{35} & w_{45} \end{pmatrix}, \qquad b^{(3)} = (b_5) = (0),$$

we obtain

$$f(x) = x^{(3)} = \varphi^{(3)}(x^{(2)}) = g^{(3)}(W^{(3)}x^{(2)} + b^{(3)}) = w_{35}x_1^{(2)} + w_{45}x_2^{(2)}. \tag{18}$$

Combining (16), (17), and (18), the final expression of the XOR NN is:

$$f(x) = \varphi^{(3)}(\,\varphi^{(2)}(\,\varphi^{(1)}(x))) = \frac{w_{35}}{1+e^{-(w_{13}x_1+w_{23}x_2+b_3)}} + \frac{w_{45}}{1+e^{-(w_{14}x_1+w_{24}x_2+b_4)}}. \tag{19}$$

Taking the MSE as the objective (loss) function, the resulting optimization problem is:

$$\min \frac{1}{4}\sum_{i=1}^{4}(f(x^i)-y^i)^2 = \frac{1}{4}\sum_{i=1}^{4}\left(\left(\sum_{k=3}^{4}\frac{w_{k5}}{1+e^{-(\Sigma_{j=1}^{2}w_{jk}x_j^i+b_k)}}\right)-y^i\right)^2 \tag{20}$$

```
1  param d >= 1, integer; #dimension input points
2  param p >= 1, integer; #number points
3  param y {i in 1..p};   #output vector
4  param x {1..p,1..d};   #input points
5  param nodes;           #number of nodes/neurons
6  set LINKS within (1..nodes cross 1..nodes); # set of links between nodes
7  param init{LINKS};     # initial values of weights
8  # variables
9  var w{(i,j) in LINKS} := init[i,j]; # weights
10 var b{1..nodes}; # biases
11 # Objective function
12 minimize MSE: 1/p*sum{i in 1..p} ( (sum{k in {3,4}} (w[k,5]*(1/(1+exp(-(
       b[k]+sum{j in 1..2} w[j,k]*x[i,j])))))) - y[i])^2 ;
```

Figure 6. *AMPL model for the XOR NN.*

Figure 6 shows the AMPL model for the XOR NN. The network topology is defined by the number of nodes (line 5) and the set of links (line 7). The optimization variables (the weights and biases) are declared in lines 9 and 10. The objective function in line 12 corresponds exactly to (20). Initial values for the weights are provided in the data file shown in Figure 7. Different initial values may lead to different solutions, as the objective function is nonlinear and nonconvex.

Solving the problem with MINOS, one of AMPL's available solvers (which implements, for unconstrained optimization problems, a quasi-Newton method based on the BFGS formula), we obtain an objective (loss) function value of $6.4 \cdot 10^{-13}$ in only 33 iterations, with the following optimal weights and biases:

$$W^{(2)} = \begin{pmatrix} 14.19 & 11.56 \\ 0.96 & 0.96 \end{pmatrix}, \qquad W^{(3)} = \begin{pmatrix} 5.85 & -6.71 \end{pmatrix}, \qquad b^{(2)} = \begin{pmatrix} 0.29 \\ 0 \end{pmatrix}.$$

```
param d := 2;
param p  := 4;
param nodes := 5;
set LINKS := (1,3) (1,4) (2,3) (2,4) (3,5) (4,5) ;
param init :=
          1 3  1
          1 4  0.1
          2 3  0.1
          2 4  0.1
          3 5  0.1
          4 5  -1.0 ;
param y :=
    1      0
    2      1
    3      1
    4      0 ;
param x :   1     2  :=
    1       0     0
    2       0     1
    3       1     0
    4       1     1 ;
```

Figure 7. *AMPL data for the XOR NN.*

Evaluating the function $f(x)$ at the four input points we obtain:

$$f(x) = \begin{pmatrix} -10^{-6} \\ 1.00 \\ 1.00 \\ 10^{-6} \end{pmatrix} \approx \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = y,$$

showing the function is an excellent approximation to XOR. Having predicted and input values that are too similar (a situation known as *overfitting*) is not always desirable, since the results for new different testing points may be incorrect. This issue is not relevant for the XOR problem, since we only have the four training points.

Although modeling languages such as AMPL greatly simplify the formulation and solution of optimization problems (in particular, the symbolic computation of derivatives), implementing a deep NN (i.e., one with many layers) can be challenging, as the objective function is a composition of multiple functions, each potentially using a different activation function. To address this complexity, several machine learning frameworks have been developed, specifically designed for modeling deep NNs. One of the most widely used is Google's TensorFlow (Abadi et al., 2016).

Figure 8 shows the implementation of the XOR problem using the Python interface to TensorFlow, called *Keras* (Chollet, 2017). Line 2 of the code imports the TensorFlow package. Lines 4–7 define the training set and force the testing set to be identical to the training set. Lines 9–11 describe the structure of the network by defining the second and

```
1  import numpy as np
2  import tensorflow as tf
3  # XOR data
4  X_train = np.array([ [ 0, 0], [0 ,1] ,[1 ,0], [1 ,1]]);
5  y_train= np.array([0,1,1,0]);
6  X_test= X_train
7  y_test= y_train
8  # create NN
9  network = tf.keras.models.Sequential([
10   tf.keras.layers.Dense(2,activation='sigmoid',input_shape=(2,)),
11   tf.keras.layers.Dense(1, activation='linear', use_bias= False)
12 ])
13 # information about NN structure
14 network.summary()
15 # create optimization problem
16 network.compile(optimizer='sgd',
17                 loss='mean_squared_error',
18                 metrics=['accuracy'])
19 #solve optimization problem
20 network.fit(X_train, y_train, epochs=100, batch_size=4)
21 #retrieve optimal solution
22 network.get_weights()
23 # test NN
24 results= network.evaluate(X_test, y_test)
25 print('test␣loss,␣test␣accuracy:', results)
```

Figure 8. *Python code using TensorFlow for the XOR NN.*

third layers; the first layer is omitted, as it simply passes the input data to the second layer. The second layer, defined in line 10, contains two nodes with the sigmoid activation function and receives two values from the first layer. The output layer, defined in line 11, consists of a single node with an identity activation function (*linear* activation in Keras/TensorFlow notation) and no bias term. Line 14 produces a summary of the network structure, including the number of parameters to be optimized. This summary, shown in Figure 9, indicates that there are six parameters in the second layer (four weights in $W^{(2)}$ and two biases in $b^{(2)}$) and two parameters in the output layer (the two weights in $W^{(3)}$, since $b^{(3)} = 0$). In this case, the summary does not provide information about the first layer because it is simply the identity function. Lines 6–8 define the optimization problem, setting the MSE as the objective function and using *sgd* (stochastic gradient descent) as the optimization method. In addition to the objective function value, another metric, the *accuracy* (defined as the fraction of correctly classified points in the testing set), is specified in line 8 to monitor the optimization process. The optimization itself is performed in line 20, considering $p' = 4$ as the size of the mini-batch set $\mathcal{P}'$, and $T = 100$ epochs. After the optimization, the optimal parameters are retrieved in line 22, and the accuracy and objective function value for the test data are reported in lines 24–25.

```
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 2)                 6

dense_1 (Dense)              (None, 1)                 2

=================================================================
Total params: 8
Trainable params: 8
Non-trainable params: 0
_________________________________________________________________
```

Figure 9. *Summary of the structure of the XOR NN.*

With the TensorFlow code of Figure 8, we obtain a solution with an objective function value of 0.41, an accuracy of 0.5, and the following optimal parameters:

$$W^{(2)} = \begin{pmatrix} 0.01 & 0.15 \\ 0.05 & 0.14 \end{pmatrix}, \qquad W^{(3)} = \begin{pmatrix} 0.01 & -0.20 \end{pmatrix}, \qquad b^{(2)} = \begin{pmatrix} 0.62 \\ -0.70 \end{pmatrix}.$$

From the accuracy value, it is evident that this solution is far from optimal. Indeed, evaluating the NN function at the four input points yields

$$f(x) = \begin{pmatrix} 0.0108 \\ 0.1486 \\ 0.0543 \\ 0.1417 \end{pmatrix} \neq \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = y.$$

This result indicates that the quasi-Newton method used by MINOS clearly outperforms stochastic gradient descent as an optimization algorithm for this small NN. To obtain a better solution with TensorFlow, the size of the network must be increased. In particular, by replacing the sigmoid activation function of the second layer with a ReLU, increasing the number of nodes in the second layer to 15 (resulting in an optimization problem with 62 parameters), and raising the number of epochs to 500, we obtain an accuracy of 1.0, although $f(x)$ still exhibits significant discrepancies with y. In spite of these limitations, stochastic gradient descent and its variants are currently essential for the optimization of very large NNs.

3.2. Neural network for the fashion-MNIST problem

The fashion-MNIST problem was introduced in Xiao, Rasul, Vollgraf (2017). The goal of this problem is to classify a set of images of garments into $k = 10$ categories: "T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal", "Shirt", "Sneaker", "Bag", and "Ankle boot". The dataset consists of 70,000 images (60,000 for training and 10,000 for testing), where each image is represented by a 28×28 matrix of pixels, and each pixel is a number in [0,255] corresponding to a grayscale intensity. In other words, the training set consists of 60,000 matrices of size 28×28, and a vector $y \in \mathbb{R}^{60,000}$, where

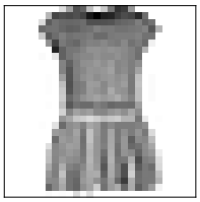

Figure 10. *Example of a garment of category "Dress".*

each component $y^i \in \{1,\ldots,10\}$ denotes the garment category. Figure 10 illustrates one of the images, corresponding to a garment of category "Dress".

For this multiclass classification problem, we consider a NN that, given a 28×28 matrix corresponding to an image, returns a probability distribution $f(x) \in \mathbb{R}^{10}$, where the predicted garment category is selected as $\arg\max_{i=1,\ldots,10} f(x)_i$. Figure 11 shows the Python code for this NN. Lines 4–10 read the dataset, declare the category names, and scale the pixel values from the range [0,255] to [0,1]. Lines 12–16 describe the structure of the network, which contains three layers. The input layer (line 13) simply reshapes the input matrix into a vector of $28 \cdot 28 = 784$ components, so this first layer has 784 nodes; this operation is called *flatten* in the code. The hidden layer (line 14) contains 128 nodes and uses a ReLU activation function. The output layer (line 15) has 10 nodes and applies a softmax activation function. Lines 20–22 formulate the optimization problem, where *adam* (a modification of stochastic gradient descent) is used as the optimization algorithm, and cross-entropy is the objective function. More precisely, line 21 sets *sparse categorical cross-entropy*, which means that for an input image i with output value $y^i = j$, where $j \in \{1,\ldots,k\}$, this value is transformed into $y^i = e_j$, with e_j being the j-th vector of the k-dimensional identity matrix. As in the XOR example, besides the objective function value, the accuracy is also used as a metric to monitor the optimization process (line 22). The optimization problem is solved in line 24 considering 10 epochs. Finally, the accuracy is computed and reported in lines 26–27.

Figure 12 shows the output produced by line 18 of the code in Figure 11. It can be observed that the input layer has no parameters, as it simply recasts the input matrix as a vector. The second layer has 100,480 parameters to optimize, corresponding to $784 \times 128 = 100,352$ weights plus 128 biases. The last layer contains 1,290 parameters, $128 \times 10 = 1,280$ weights plus 10 biases. Overall, the optimization problem involves 101,770 variables.

Once the optimization problem is solved, we obtain an optimal solution with an objective function value of 0.2391 and an accuracy of 0.9098. The accuracy for the testing data, reported by line 27, is 0.8797; that is, 87.97% of the test images are correctly classified. Figure 13 shows the predictions made by the NN on a subset of 25 test images. All 25 images are correctly classified.

```
import numpy as np
import tensorflow as tf
#load fashion-MNIST data
fashion_mnist = tf.keras.datasets.fashion_mnist
(x_train, y_train),(x_test, y_test) = fashion_mnist.load_data()
#name labels
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
#scale pixels 0-255 to 0-1
x_train, x_test = x_train / 255.0, x_test / 255.0
#create NN
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])
#information about NN structure
model.summary()
#create optimization problem
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
#solve optimization problem
model.fit(x_train, y_train, epochs=10, validation_data=(x_test,y_test))
#test NN
results= model.evaluate(x_test, y_test)
print('test loss, test acc:', results)
```

Figure 11. *Python code using TensorFlow for the fashion-MNIST NN.*

4. Convolutional neural networks

CNNs (LeCun et al., 1998) are an extension of feedforward NNs, particularly suitable for image data (e.g., pictures, videos, and similar types of structured grids). CNNs introduce new types of layers in addition to the fully connected ones described in previous sections. Two of the most important types are the *convolutional* and *pooling* layers, which are outlined in the following two subsections.

4.1. Convolutional layers

Convolutional layers are applied to images, which can be represented as three-dimensional arrays of size $t_1 \times t_2 \times t_3$ (or higher); such structures are referred to as *tensors*. Commonly, the last dimension has $t_3 = 3$, meaning that the image is composed of three $t_1 \times t_2$ matrices of pixels. Each of these three matrices is referred to as a *channel*, corresponding to the RGB (red, green, blue) components of the image. The entry (i, j) in each channel gives the amount of red, green, or blue in pixel (i, j) of the image.

```
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0

dense_2 (Dense)              (None, 128)               100480

dense_3 (Dense)              (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
```

Figure 12. *Summary of the structure of the fashion-MNIST NN.*

Figure 13. *Predictions for some images.*

Convolutional layers consist of a set of matrices (called *kernels* or *filters*) of dimension $h_1 \times h_2 \times h_3$, where $h_1 = h_2 = h$ for a chosen h (typically 3, 5, or 7), and $h_3 = t_3$ (in the case of RGB images, $h_3 = t_3 = 3$). These kernels scan the tensor that defines the image in order to detect patterns. For the case of RGB images, the scan is performed over subtensors of dimension $h \times h \times 3$, moving from the first corner of the tensor (top left of first channel), to the last one (bottom right of last channel), and computing a linear combination of the subtensor with the kernel at each position. The kernel values are the parameters to be optimized in the NN.

Figure 14 illustrates the convolutional operation for the case of an image with only one 4×4 channel, considering a 3×3 kernel K. For instance, for an image with the following particular channel

$$A = \begin{pmatrix} 0.5 & 0.7 & 0.2 & 1 \\ 0.3 & 0.2 & 0.3 & 0.7 \\ 0.1 & 0.8 & 0.1 & 0.9 \\ 0 & 0.4 & 0.3 & 1 \end{pmatrix}, \tag{21}$$

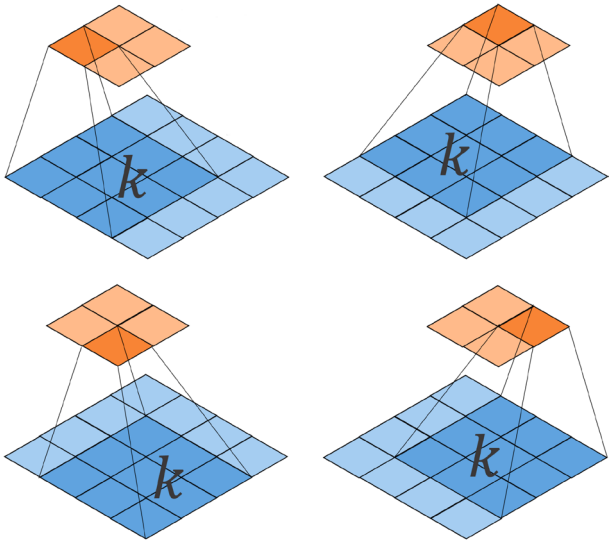

Figure 14. *Example of convolutional layer*

and a kernel 3×3,

$$K = \begin{pmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{pmatrix},$$

the convolutional operation would provide a matrix R of size 2×2 with the following four entries:

- For top left 3×3 submatrix of A: $R_{11} = 0.5k_{11} + 0.7k_{12} + 0.2k_{13} + 0.3k_{21} + 0.2k_{22} + 0.3k_{23} + 0.1k_{31} + 0.8k_{32} + 0.1k_{33}$.
- For top right 3×3 submatrix of A: $R_{12} = 0.7k_{11} + 0.2k_{12} + 1k_{13} + 0.2k_{21} + 0.3k_{22} + 0.7k_{23} + 0.8k_{31} + 0.1k_{32} + 0.9k_{33}$.
- For the bottom left 3×3 submatrix of A: $R_{21} = 0.3k_{11} + 0.2k_{12} + 0.3k_{13} + 0.1k_{21} + 0.8k_{22} + 0.1k_{23} + 0k_{31} + 0.4k_{32} + 0.3k_{33}$.
- For the bottom right 3×3 submatrix of A: $R_{22} = 0.2k_{11} + 0.3k_{12} + 0.7k_{13} + 0.8k_{21} + 0.1k_{22} + 0.9k_{23} + 0.4k_{31} + 0.3k_{32} + 1k_{33}$.

The resulting matrix after the convolution operation with the kernel has dimension $(t_1 - h + 1) \times (t_2 - h + 1)$. If layer j of the NN is a convolutional layer with d_j nodes, then the number of resulting matrices in this layer is d_j. Since for each of these d_j nodes, kernels of size $h \times h \times t_3$ are required, plus one bias, the total number of NN parameters to optimize for layer j is $d_j(t_3h^2 + 1)$. In state-of-the-art NN frameworks, the kernel is also allowed to move more than one pixel (horizontally and vertically) across the channel matrices during the convolution operation. This pixel step size is called the *stride*. To simplify the exposition, we assume a stride of one in the examples above and below.

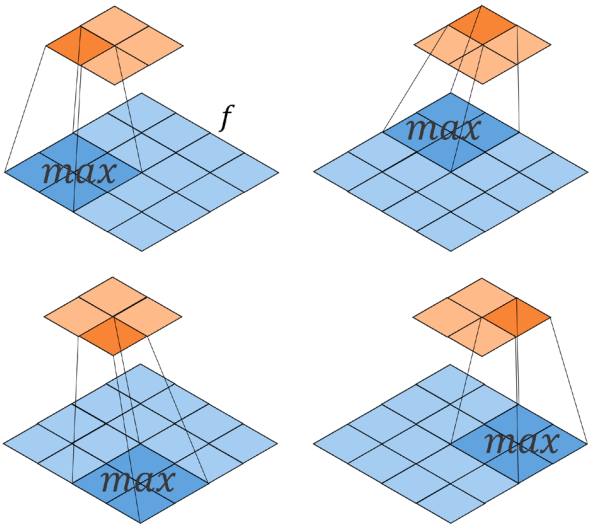

Figure 15. *Example of max pooling layer*

After the convolution operation is performed, the activation function of the layer is applied to each entry of the resulting matrix.

4.2. Pooling layers

Pooling layers are used to reduce the dimensionality of the data, allowing faster computations, freeing memory, and helping to prevent overfitting. These layers do not add new parameters to the NN; they simply apply a specific function to submatrices of the input matrices. Commonly used functions include the max function, which returns the maximum value of each submatrix, and the average function, which returns the mean of its values. For example, if layer j of the NN is a pooling layer that receives matrices of size $t \times t$ from layer $j-1$, and the size of the pooling operation is h (assuming h divides t), then the matrices generated by layer j will have dimension $t/h \times t/h$. Pooling layers are typically placed after convolutional layers to reduce their output dimensions. Figure 15 illustrates a max pooling operation of size 2 on a 4×4 matrix.

As an example, for matrix (21), applying a max pooling operation of size 2 yields the resulting matrix

$$R = \begin{pmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{pmatrix}, \text{ where } \begin{array}{ll} R_{11} = \max\{0.5,0.7,0.3,0.2\}, & R_{12} = \max\{0.2,1,0.3,0.7\}, \\ R_{21} = \max\{0.1,0.8,0,0.4\}, & R_{22} = \max\{0.1,0.9,0.3,1\}. \end{array}$$

Although the max function is not differentiable, NNs are still optimized using stochastic gradient methods, since it is piecewise linear and differentiable almost everywhere.

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import numpy as np
#load data
(train_images, train_labels), (test_images, test_labels) = datasets.
    cifar10.load_data()
#name labels
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
# scale pixels from [0,255] to [0,1]
train_images, test_images = train_images / 255.0, test_images / 255.0
#create NN
m = models.Sequential()
m.add(layers.Conv2D(32,(3,3),activation='relu',input_shape=(32,32,3)))
m.add(layers.MaxPooling2D((2, 2)))
m.add(layers.Conv2D(64, (3, 3), activation='relu'))
m.add(layers.MaxPooling2D((2, 2)))
m.add(layers.Conv2D(64, (3, 3), activation='relu'))
m.add(layers.Flatten())
m.add(layers.Dense(64, activation='relu'))
m.add(layers.Dense(10,activation='softmax'))
#information about NN structure
m.summary()
#create optimization problem
m.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
#solve optimization problem
m.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
#test NN
results = m.evaluate(test_images,  test_labels, verbose=2)
print('test loss, test acc:', results)
```

Figure 16. *Python code using TensorFlow for the CIFAR-10 CNN.*

5. An illustrative application of convolutional neural networks

To illustrate the use of CNNs we consider the CIFAR-10 dataset (Krizhevsky, 2009). CIFAR-10 contains 60,000 color images, 50,000 for training and 10,000 for testing. Each image consists of three 32×32 pixels matrices, each matrix corresponding to one of the RGB channels. As in the fashion-MNIST problem, the goal is to classify the color images into $k = 10$ categories: "airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship", and "truck".

The CNN for the CIFAR-10 problem is a composition of nine functions, that is, $f(x) = \varphi^{(9)}(\varphi^{(8)}(...\varphi^{(2)}(\varphi^{(1)}(x))...))$, where each function (layer) is as follows:

- The first (input) layer simply takes the three RGB channels of an image and passes them to the second layer. Hence, $\varphi^{(1)} : \mathbb{R}^{32\times32\times3} \to \mathbb{R}^{32\times32\times3}$.
- The second layer is a convolutional layer with 32 kernels (and thus 32 nodes) of

size $3 \times 3 \times 3$, followed by a ReLU activation function. The 32 output matrices have dimensions 30×30. Hence, $\varphi^{(2)} : \mathbb{R}^{32\times32\times3} \to \mathbb{R}^{30\times30\times32}$. The NN parameters to be optimized include the kernel weights and the node biases, which amount to $32 \cdot 3 \cdot 3 \cdot 3 + 32 = 896$.

- The third layer is a max pooling layer of size 2×2; thus, the dimensions of the matrices generated by the second layer are halved. Therefore, $\varphi^{(3)} : \mathbb{R}^{30\times30\times32} \to \mathbb{R}^{15\times15\times32}$.
- The fourth layer is another convolutional layer with 64 kernels (64 nodes) of size $3 \times 3 \times 32$, with a ReLU activation function. The 64 output matrices have dimensions 13×13. Hence, $\varphi^{(4)} : \mathbb{R}^{15\times15\times32} \to \mathbb{R}^{13\times13\times64}$. The NN parameters to be optimized include the kernel weights and the 64 node biases, which amount to $64 \cdot 3 \cdot 3 \cdot 32 + 64 = 18,496$.
- The fifth layer is another max pooling layer of size 2×2. Therefore, it halves the dimensions of the matrices generated by the fourth layer, and $\varphi^{(5)} : \mathbb{R}^{13\times13\times64} \to \mathbb{R}^{6\times6\times64}$.
- The sixth layer is the third convolutional layer, with 64 kernels (and thus 64 nodes) of size $3 \times 3 \times 64$, followed by a ReLU activation function. The 64 matrices generated by this layer have dimensions 4×4. Therefore, $\varphi^{(6)} : \mathbb{R}^{6\times6\times64} \to \mathbb{R}^{4\times4\times64}$. The NN parameters to be optimized are the kernel weights and the 64 node biases, that is $64 \cdot 3 \cdot 3 \cdot 64 + 64 = 36,928$.
- The seventh layer simply recasts the 64 4×4 matrices produced by the previous layer into a vector (flatten operation) of size $4 \cdot 4 \cdot 64 = 1,024$. Hence, $\varphi^{(7)} : \mathbb{R}^{4\times4\times64} \to \mathbb{R}^{1,024}$.
- The last two layers (eighth and ninth) are standard fully connected layers of a feedforward NN. The first of them has 64 nodes and a ReLU activation function. Therefore, it adds $1,024 \cdot 64 = 65,536$ weights and 64 biases (a total of 65600 parameters) to the optimization problem. The last layer has 10 nodes and a softmax activation function, thus adding $64 \cdot 10 + 10 = 650$ parameters to the NN. Hence, $\varphi^{(8)} : \mathbb{R}^{1,024} \to \mathbb{R}^{64}$ and $\varphi^{(9)} : \mathbb{R}^{64} \to \mathbb{R}^{10}$.

From the previous description, the total number of parameters in the optimization problem is $896 + 18,496 + 36,928 + 65,600 + 650 = 122,570$. Figure 16 shows the Python code for solving the CIFAR-10 problem using TensorFlow. The structure of the CNN is defined in lines 12–20, one line per layer, except for line 13, which defines the first convolutional layer and implicitly the input layer as well. The rest of the code follows the same structure as in the fashion-MNIST example. The summary of the CNN produced by line 22 is shown in Figure 17; it can be observed that the dimensions and number of parameters to be optimized in the different layers are exactly as described above.

Once the optimization problem is solved using the cross-entropy objective function, we obtain an objective function (loss) value of 0.5891 and accuracies of 0.7889 and 0.6976 for the training and testing sets, respectively. Figure 18 shows the predictions made by the CNN for a subset of images. Although some errors remain, most images are correctly classified.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36,928
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 64)	65,600
dense_1 (Dense)	(None, 10)	650

Total params: 122,570 (478.79 KB)
Trainable params: 122,570 (478.79 KB)
Non-trainable params: 0 (0.00 B)

Figure 17. *Summary of the CNN.*

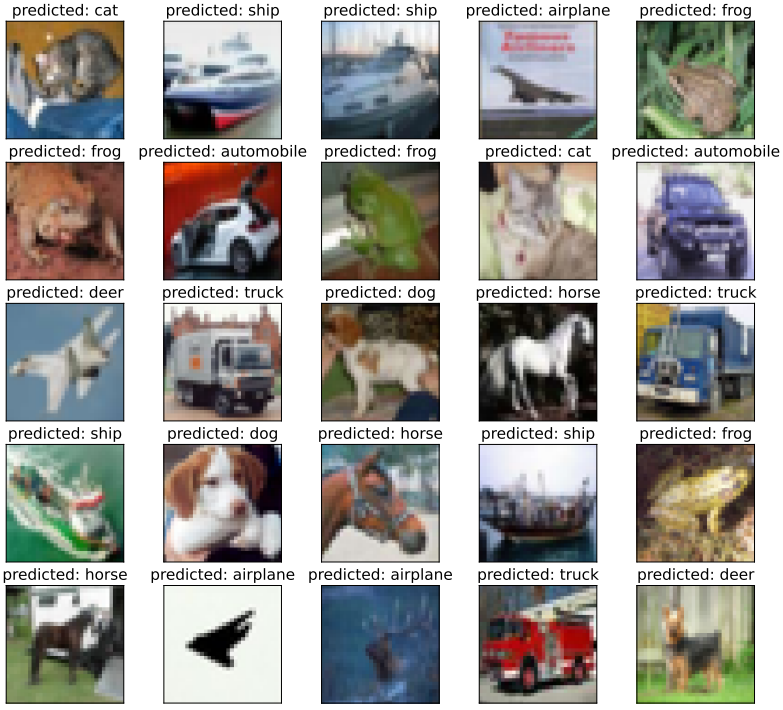

Figure 18. *Predictions for some images.*

6. Applications of neural networks in operations research

NNs have several applications in OR algorithms, for instance:

- *Predicting good solutions or heuristics.* NNs can predict near-optimal decisions (e.g., routing choices or resource allocations) instead of solving optimization models from scratch.
- *Learning objective or constraint structures.* In complex systems where the true objective or constraints are unknown or hard to model (for example, demand forecasting in supply chains), NNs can learn these relationships directly from data.
- *Combining with classical solvers (hybrid approaches).* For example, a NN can help generate feasible starting points or parameter settings for mixed-integer programming solvers, cutting planes, or metaheuristics.

Most applications of NNs in OR focus on heuristics for combinatorial optimization problems, including vehicle routing, knapsack, and assignment problems. These NN-based heuristics may follow several strategies, two of the most common being:

- *Supervised learning* (or *data-driven* approaches): a NN is trained on pairs of problem instances and their corresponding (optimal or heuristic) solutions, and later used to predict near-optimal solutions for new instances.
- *Reinforcement learning:* the NN acts as an agent that iteratively learns to make good decisions by interacting with an environment.

A discussion of these NN-based heuristics can be found in the recent survey Bengio et al. (2021).

For illustration purposes, the next subsection shows how NNs can be used to construct a simple data-driven heuristic for the knapsack problem. It is worth remarking that our purpose is not to promote the use of NNs for OR problems. Indeed, NNs have several limitations in this context: (1) they usually provide only approximate solutions rather than optimal ones; (2) they require large training datasets; and (3) the interpretability and feasibility of their solutions can be problematic for strict OR models with hard constraints. In addition, for most problems, classical optimization solvers (such as CPLEX) remain far superior.

6.1. Supervised learning for the 0–1 knapsack problem

Consider a dataset of N independent instances of the 0–1 knapsack problem. Each instance j contains n items with given values $v_i^j > 0$, weights $w_i^j > 0$, and capacity $C^j > 0$, for $i = 1, \ldots, n$ and $j = 1, \ldots, N$. Therefore, the optimization problem for instance j is:

$$\begin{aligned} \max_{x^j} \quad & \sum_{i=1}^{n} v_i^j x_i^j \\ \text{s. to} \quad & \sum_{i=1}^{n} w_i^j x_i^j \le C^j, \\ & x_i^j \in \{0,1\}, \quad i = 1,\dots,n. \end{aligned} \tag{22}$$

For each instance j, an optimal binary solution $x^{*j} \in \{0,1\}^n$ can be obtained exactly using a standard solver (such as CPLEX). These optimal solutions are used as training labels in a supervised learning framework.

For every item i in instance j, we define a feature vector $\mathbf{z}_i^j \in \mathbb{R}^4$ as

$$\mathbf{z}_i^j = \left(v_i^j,\, w_i^j,\, v_i^j/w_i^j,\, w_i^j/C^j\right), \tag{23}$$

and associate it with the corresponding optimal label x_i^{*j}. The resulting training set is therefore

$$\left\{(\mathbf{z}_i^j, x_i^{*j}),\, i = 1,\dots,n,\; j = 1,\dots,N\right\}, \tag{24}$$

which contains Nn labeled samples.

A small feedforward NN $f : \mathbb{R}^4 \to (0,1)$ is trained to approximate the probability that item i belongs to the optimal knapsack solution. Given the feature vector $\mathbf{z}_i^j$, the network predicts $f(\mathbf{z}_i^j;\theta)$, which is interpreted as the probability of including item i in the solution. The model parameters θ are estimated by minimizing a suitable objective (loss function) over the entire dataset, as done in the examples of the previous sections.

Once trained, the network can be used to make fast approximate decisions for new knapsack instances. Given the predicted probabilities $f(\mathbf{z}_i^j)$ for a new instance j, a feasible binary solution is obtained by sorting items according to $f(\mathbf{z}_i^j)$ (or the ratio $f(\mathbf{z}_i^j)v_i^j/w_i^j$) and selecting them greedily as long as the capacity constraint $\sum_{i=1}^{n} w_i^j x_i^j \le C^j$ is satisfied.

The previous heuristic has been implemented in Python. The full code is not included here for space reasons, but it is freely available from the corresponding author. AMPL and CPLEX are used to define and solve knapsack problems exactly in order to obtain a training set of $N = 500$ instances, each with $n = 10$ items (thus $500 \cdot 10 = 5{,}000$ data points), and a testing set of 50 additional instances with the same number of items. The NN considered has four layers: an input layer with four nodes, two hidden layers with 64 nodes and ReLU activation functions, and an output layer with one node and a sigmoid activation that returns the estimated probability of an item being in the optimal solution of its corresponding instance.

For the $50 \cdot 10 = 500$ items in the testing set, using the naive rule that selects an item if its predicted probability $f(\mathbf{z}_i^j)$ exceeds 0.5, the accuracy of the NN lies in the range 85%–90%; that is, 85%–90% of the items predicted as optimal by the NN are indeed optimal in the exact solution of the instance. Using the more sophisticated greedy heuristic described above (i.e., sorting items by $f(\mathbf{z}_i^j)$ and selecting them greedily as long as the capacity constraint is satisfied) leads to better results. For example, for

the instance with objective coefficients $v = [82,70,8,67,28,99,71,25,13,54]$, weights $w = [14,14,18,11,9,3,3,10,6,7]$, and capacity $C = 38$, the heuristic produces the solution $x = [1,0,0,1,0,1,1,0,0,1]$, which coincides with the optimal solution returned by CPLEX.

This simple supervised learning approach illustrates how NNs can approximate classical optimization rules (e.g., selecting items by decreasing value-to-weight ratio) through data-driven training, while providing a bridge between exact combinatorial optimization and predictive modeling.

7. Conclusions

In a didactic manner, this work has illustrated the origin of the optimization problems underlying modern deep NNs. We have shown that, due to the large composition of functions involved, standard modeling languages used in OR and optimization (such as AMPL) can only be conveniently applied to shallow NNs, and that specialized tools (such as TensorFlow) are required to model and optimize more complex architectures. Intentionally, we have left aside more theoretical aspects, aiming instead to provide an accessible introduction to the field of deep NNs for operations researchers.

In addition, a simple NN-based heuristic for the knapsack problem has illustrated how neural models can be combined with classical OR tools to obtain fast, data-driven approximations to combinatorial optimization problems. Because NNs are trained through stochastic gradient–based optimization, these heuristics can be interpreted as an approach based on unconstrained optimization for generating approximate solutions to combinatorial optimization problems. This connection emphasizes once more the central role of optimization in the design, training, and practical use of modern neural network techniques.

8. Acknowledgments

Pau Amaré was supported by the UPC scholarship 9637. Jordi Castro was supported by the MCIN/AEI/10.13039/501100011033/FEDER,EU grant PID2022-139219OB-I00.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y. and Zheng, X. (2016). TensorFlow: a system for large-scale machine learning. In: *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2016)*, 265–283. USENIX Association.

Bengio, Y., Lodi, A. and Prouvost, A. (2021). Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2), 405–421.

Bottou, L., Curtis, F.E. and Nocedal, J. (2018). Optimization methods for large-scale machine learning. *SIAM Review*, 60(2), 223–311.

Cauchy, A. (1847). Méthode générale pour la résolution des systèmes d'équations simultanées. *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences*, 25, 536–538.

Chollet, F. (2017). *Deep Learning with Python*. Manning Publications, Shelter Island, NY, USA.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4), 303–314.

Fourer, R., Gay, D. M. and Kernighan, B. W. (2003). *AMPL: A Modeling Language for Mathematical Programming (2nd ed.)*. Thomson Brooks/Cole, Pacific Grove, CA, USA.

Goodfellow, I., Bengio, Y. and Courville, A. (2016). *Deep Learning*. MIT Press, Cambridge, MA, USA.

Hornik, K., Stinchcombe M. and White H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5), 359–366.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical Report, Department of Computer Science, University of Toronto.

LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.

Luenberger, D.G. and Ye, Y. (2008). *Linear and Nonlinear Programming (3rd ed.)*. Springer, New York, USA.

Minsky, M. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA.

Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27(2), 372–376.

Nocedal, J. and Wright, S.J. (2006). *Numerical Optimization, 2nd Ed.*. Springer, New York, USA.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5), 1–17.

Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986). Learning representations by back-propagating errors. *Nature*, 323, 533–536.

Sra, S., Nowozin, S. and Wright, S.J. (eds.) (2011). *Optimization for Machine Learning*. MIT Press, Cambridge, MA, USA.

Sutskever, I., Vinyals, O. and Le, Q.V. (2014). Sequence to sequence learning with neural networks. In: *Advances in Neural Information Processing Systems (NeurIPS)*, 27.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In: *Advances in Neural Information Processing Systems (NeurIPS)*, 30.

Xiao, H., Rasul, K. and Vollgraf, R. (2017). Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *arXiv preprint* arXiv:1708.07747.

Information for authors

Author Guidelines

SORT accepts for publication only original articles that have not been submitted simultaneously to any other journal in the areas of statistics, operations research, official statistics or biometrics. Furthermore, once a paper is accepted it must not be published elsewhere in the same or similar form.

SORT is an **Open Access** journal which **does not** charge publication **fees**.

Articles should be preferably of an applied nature and may include computational or educational elements. Publication will be exclusively in English. All articles will be forwarded for systematic peer review by independent specialists and/or members of the Editorial Board.

Submission of papers must be in electronic form only at our **RACO** (Revistes Catalanes en Accés Obert) submission site. Initial submission of the paper should be a single document in **PDF** format, including all **figures and tables** embedded in the main text body. **Supplementary material** may be submitted by the authors at the time of submission of a paper by uploading it with the main paper at our RACO submission site. **New authors**: please register. Upon successful registration you will be sent an e-mail with instructions to verify your registration.

The article should be prepared in **double-spaced** format, using a **12-point** typeface. SORT strongly recommends the use of its LaTeX template.

The **title page** must contain the following items: title, name of the author(s), professional affiliation and complete address, and an abstract (75–100 words) followed by the keywords and MSC2010 Classification of the American Mathematical Society.

Before submitting an article, the author(s) would be well advised to ensure that the text uses **correct English**. Otherwise the article may be returned for language improvement before entering the review process. The article must also use inclusive language, that is, language that avoids the use of certain expressions or words that might be considered to exclude particular groups of people, especially gender-specific words, such as “man”, “mankind”, and masculine pronouns, the use of which might be considered to exclude women. The article should also inform about whether the original data of the research takes gender into account, in order to allow the identification of possible differences.

Bibliographic references within the text must follow one of these formats, depending on the way they are cited: author surname followed by the year of publication in parentheses [e.g., Mahalanobis (1936) or Rao (1982b)])]; or author surname and year in parentheses, without comma [e.g. (Mahalanobis 1936) or (Rao 1982b) or (Mahalanobis 1936, Rao 1982b)]. The complete reference citations should be listed alphabetically at the end of the article, with multiple publications by a single author listed chronologically. Examples of reference formats are as follows:

- Article: Casella, G. and Robert, C. (1998). Post-processing accept-reject samples: recycling and rescaling. Journal of Computational and Graphical Statistics, 7, 139–157.

- Book: Gelman, A., Carlin, J. B., Stern, H. S. and Rubin, D. B. (2003). Bayesian Data Analysis, 2nd Ed. Chapman & Hall / CRC, New York.

- Chapter in book: Engelmann, B. (2006). Measures of a rating’s discriminative power-applications and limitations. In: Engelmann, B. and Rauhmeier, R. (eds), The Basel Risk Parameters: Estimation, Validation, and Stress Testing. Springer, New York.

- Online article (put issue or page numbers and last accessed date): Marek, M. and Lesaffre, E. (2011). Hierarchical generalized linear models: The R package HGLMMM. Journal of Statistical Software, 39 (13). http://www.jstatsoft.org/v39/i13. Last accessed 28 March 2011.

Explanatory footnotes should be used only when absolutely necessary. They should be numbered sequentially and placed at the bottom of the corresponding page. **Tables and figures** should also be numbered sequentially.

Papers should not normally exceed about **25 pages** of the **PDF** format (**40 pages** of the format provided by the SORT **LaTeX** template) including all figures, tables and references. Authors should consider transferring content such as long tables and supporting methodological details to the online supplementary material on the journal's web site, particularly if the paper is long.

Authors must indicate the **source of funding** for the articles.

Authors can **preprint** their manuscripts during the submission process and showcase their work to the global research community, before it is accepted or published. This can be done in any non-commercial preprint server such as **archiv**.

Once the article has positively passed the first review round, the executive editor assigned with the evaluation of the paper will send comments and suggestions to the authors to improve the paper. At this stage, the executive editor will ask the authors to submit a revised version of the paper using the SORT **LaTeX** template.

Once the article has been accepted, the journal editorial office will **contact the authors** with further instructions about this final version, asking for the source files.

The Library of Catalonia (depending on the Government of Catalonia) periodically records SORT website and stores it indefinitely in the repository **PADICAT** (Patrimoni Digital de Catalunya: https://www.padicat.cat/)

Submission Preparation Checklist

As part of the submission process, authors are required to check off their submission's compliance with all of the following items, and submissions may be returned to authors that do not adhere to these guidelines.

1. The submitted manuscript follows the guidelines to authors published by SORT
2. Published articles are under a Creative Commons License BY-NC-ND
3. Font size is 12 point
4. Text is double-spaced
5. Title page includes title, name(s) of author(s), professional affiliation(s), complete address of corresponding author
6. Abstract is 75-100 words and contains no notation, no references and no abbreviations
7. Keywords and MSC2010 classification have been provided
8. Bibliographic references are according to SORT's prescribed format
9. English spelling and grammar have been checked
10. Manuscript is submitted in PDF format

Creative Commons License

Copyright notice and author opinions

Authors transfer the exploitation rights of their works to the journal. The Institut d'Estadística de Catalunya holds the copyright ownership of the contents published in the journal. Authors may deposit a copy of their works in repositories, as specified in the self-archiving policy. Published articles represent the author's opinions; the journal SORT does not necessarily agree with the opinions expressed in the published articles.

Self-archiving policy

The journal SORT allows the deposit and dissemination of the published version of the contributions in any institutional, subject and/or multidisciplinary repository. The repository must contain the information about the publication in the journal and the corresponding link. The journal allows the deposit and dissemination of article preprints, but recommends being linked to the published version.

Statement of ethics and good practices

As a journal co-edited by the Universitat de Barcelona, SORT declares that follows its "Declaration of Ethics and Good Practices for Scientific Journals" (https://diposit.ub.edu/dspace/handle/2445/97665)

SORT Statistics and Operations Research Transactions
Institut d'Estadística de Catalunya (Idescat)
Via Laietana, 58 - 08003 Barcelona. SPAIN
Tel. +34-93.557.30.76
sort@idescat.cat

How to cite articles published in SORT

Commenges, D. (2003). Likelihood for interval-censored observations from multi-state models. *SORT*, 27 (1), 1-12.