Scatter Search

Rafael Martí, Ángel Corberán, and Juanjo Peiró

Contents

Abstract

Scatter search (SS) is a population-based metaheuristic that has been shown to yield high-quality outcomes for hard combinatorial optimization problems. It uses strategies for combining solution vectors, making limited use of randomization, that have proved effective in a variety of problem settings. The fundamental concepts and principles were first proposed in the 1960s and 1970s as an extension of mathematical relaxation techniques for combinatorial

R. Martí (✉) • Á. Corberán • J. Peiró
Facultat de Ciències Matemàtiques, Departament d'Estadística i Investigació Operativa, Universitat de València, Burjassot – València, Spain
e-mail: rafael.marti@uv.es; angel.corberan@uv.es; juanjo.peiro@uv.es

R. Martí et al. (eds.), *Handbook of Heuristics*,
DOI 10.1007/978-3-319-07153-4_20-1

optimization problems. Its framework is flexible, allowing the development of implementations with varying degrees of sophistication.

This chapter provides a grounding in the scatter search methodology that will allow readers to create successful applications of their own. To illustrate this, we present a scatter search implementation for a $\mathcal{NP}$-hard variant of the classic p-hub median problem, for which we describe search elements, mechanisms, and strategies to generate, combine, and improve solutions.

Keywords

Scatter search • Metaheuristic • Combinatorial optimization • p-hub

Introduction

Scatter search (SS) was conceived as an extension of a heuristic in the area of mathematical relaxation, which was designed for the solution of integer programming problems: surrogate constraint relaxation. The following three operations come from the area of mathematical relaxation, and they are the core of most evolutionary optimization methods including SS and genetic algorithms (GAs):

- Building, maintaining, and working with a population of elements (coded as vectors).
- Creating new elements by combining existing elements.
- Determining which elements are retained based on a measure of quality.

Two of the best-known mathematical relaxation procedures are Lagrangean relaxation [9] and surrogate constraint relaxation [12]. While Lagrangean approaches absorb "difficult" constraints into the objective function by creating linear combinations of them, surrogate constraint relaxations generate new constraints to replace those considered problematic. The generation of surrogate constraints also involves the combination of existing constraints using a vector of weights. In both cases, these relaxation procedures search for the best combination in an iterative manner.

Scatter search is more intimately related to surrogate relaxation procedures, because not only surrogate relaxation includes the three operations outlined above but also has the goal of generating information from the application of these operations. In the case of surrogate relaxation, the goal is to generate information that cannot be extracted from the original constraints. Scatter search takes on the same approach, by generating information through the combination of two or more solutions. It strategically explores the solution space of an optimization problem by evolving a set of *reference* points. These points define a set, known as *reference set* (*RefSet*), which consists of good solutions obtained by prior solving efforts.

SS and GAs were both introduced in the seventies. While Holland [20] introduced genetic algorithms and the notion of imitating nature and the "survival of the

fittest" paradigm, Glover [13] introduced scatter search as a heuristic for integer programming that expanded on the concept of surrogate constraints as mentioned above. Both methods fall in the category of evolutionary optimization procedures, since they build, maintain, and evolve a set (population) of solutions throughout the search. Although the population-based approach makes SS and GAs part of the so-called evolutionary methods, there are fundamental differences between the two methodologies. Typical differences between the scatter search methodology and other evolutionary methods rely on the use of randomization and the size of the population of solutions. Specifically, SS mainly implements deterministic strategies to generate, combine, and improve solutions, while other evolutionary methods, such as GAs, generally introduce random elements in their strategies. On the other hand, in SS, the *RefSet* tends to be small, usually around 10 solutions, while in GAs the population tends to be much larger, usually around 100.

The following principles can be seen as the foundations of the scatter search methodology:

- Useful information about the characteristics of optimal solutions is typically contained in a suitable diverse collection of elite solutions.
- This information is exploited by the combination of the elite solutions.
- The purpose of the combinations is to incorporate to the elite set both diversity (in terms of solutions' attributes) and quality (in terms of objective value).
- Search mechanisms and strategies are not limited to combinations but include solution generation and improvement combination methods.

The scatter search framework is flexible, allowing the development of alternative implementations with varying degrees of sophistication. It is based on the idea of limiting the scope of the search to a selective group of combination types, as a mechanism for controlling the number of possible combinations in a given reference set. A comprehensive examination of this methodology can be found in the book of Laguna and Martí in [32]. In the following subsection, we review the most relevant recent applications to illustrate the impact of this methodology.

Past and Present

As mentioned before, scatter search was first introduced in 1977 [13] as a heuristic for integer programming. However, it seems that it was never applied or studied again until 1990, when it was presented at the EPFL Seminar on Operations Research and Artificial Intelligence Search Methods (Lausanne, Switzerland). An article based on this presentation was published in 1994 [14], in which the range of applications was expanded to nonlinear (continuous) optimization problems, binary and permutation problems. The algorithmic principles and the structure of the method were finally proposed in the so-called scatter search template [15]. In a way, this template was a simplification of the previous descriptions and served as the main reference for most of the scatter search implementations up to date.

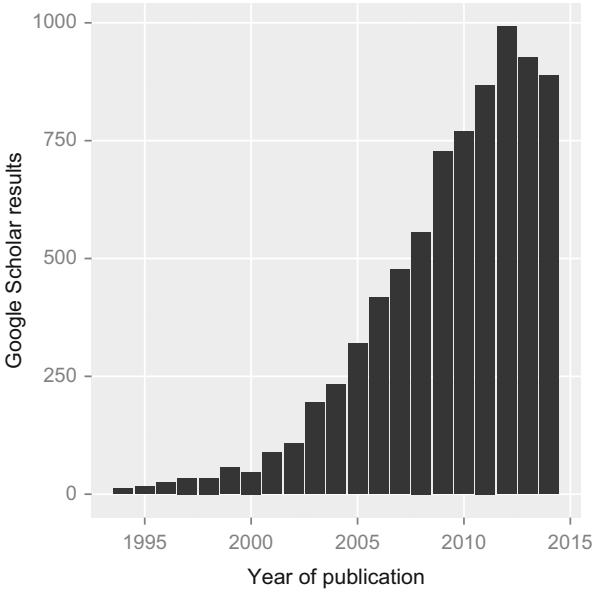

Fig. 1 Publication counts for the query "scatter search" on Google Scholar

There are two important milestones in the scatter search publications since then. In 2002, a book on this methodology was published by Kluwer [32], which included implementations in C to help the reader to create his/her own applications. In 2006, a special issue of the *European Journal of Operational Research* [1] was devoted to successful scatter search applications.

The scatter search methodology has become the method of choice for the design of solution procedures for many $\mathcal{NP}$-hard combinatorial optimization problems. Its use has been steadily increasing as shown in Figs. 1 and 2. Those figures show the number of yearly publications from 1994 to 2014 on scatter search. Figure 1 shows the number of cites or references for the query "scatter search" on Google Scholar (http://scholar.google.com), while Fig. 2 shows the counts for the same query on the Science Citation Index database of Thomson Reuter's Web of Science, which corresponds to the publications in this topic (http://thomsonreuters.com/thomson-reuters-web-of-science/). To restrict our search, we looked up on the Scopus website (http://www.scopus.com) the number of publications with the query "scatter search" in the title and found that it was 298. We have shortlisted the most recent ones (from 2012 to 2014) that refer to journal papers and deal with practical applications solved with scatter search. Table 1 summarizes the most relevant articles classified into six categories, according to the class of problem being solved. In particular we consider classical problems such as TSP, VRP, and knapsack; scheduling and sequencing problems such as flowshop, jobshop, and project scheduling; manufacturing problems ranging from NoC mapping to coal and steel production; planning

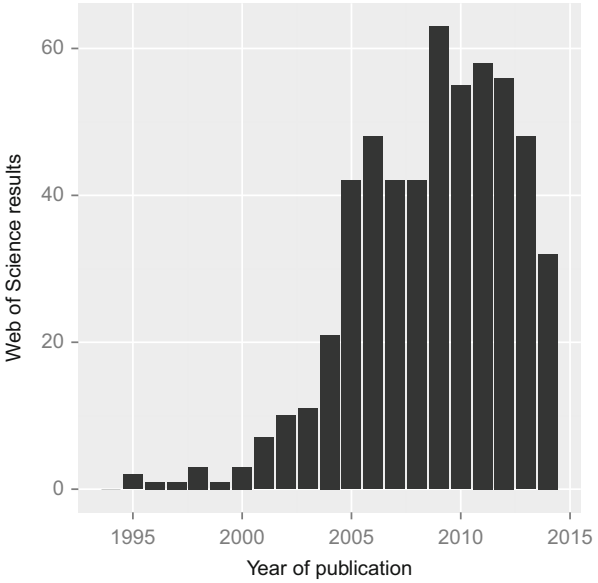

Fig. 2 Publication counts for the query "scatter search" on the Web Of Science

problems such as territory design and location problems; statistics and finance problems such as credit scoring and protein information prediction; and image registration problems such as image registration and segmentation and craniofacial superimposition.

Although we can consider that the scatter search methodology is nowadays well established, and that it has been used in many applications, if we compare SS with other metaheuristics, we can see that there is still significant room for improvement. In particular, a Scopus search with the query "genetic algorithm" returned almost 40,000 entries, while "tabu search" gave 2,273 hits. Consequently, one of this chapter's goals is to trigger the interest of researchers and practitioners to apply the scatter search methodology.

The Five Main Components

In scatter search, the search process is performed to capture information not contained separately in the original solutions. It takes advantage of auxiliary heuristic methods for selecting the elements to be combined and generating new solutions. The combination strategy is devised with the belief that this information can be better exploited when integrated rather than treated in isolation. In general, the decision rules created from such combination strategies produce better empirical outcomes than the standard applications of local decision rules.

Table 1 Journal papers published between 2012 and 2014

Classical problems	
TSP	[46]
VRP	[2, 54, 59, 61]
Linear ordering	[22]
Knapsack	[36]
4-color mapping	[51]
Cutwidth minimization	[47]
Global optimization and black-box	[23, 31]
Scheduling and sequencing	
Flowshop	[19, 44]
Assembly lines	[39]
Jobshop	[7]
Task scheduling	[27, 53]
Project scheduling	[55]
Manufacturing	
NoC mapping	[33, 34]
Aircraft conflict resolution	[37, 56]
Cellular manufacturing systems	[25]
Grid resources selection	[4]
Coal production planning	[49]
Steel industry	[41]
Disassembly sequence	[17]
SONET problems	[3]
Environmental–economic dispatch	[6]
Planning	
Layout	[26, 29, 30]
Transmission expansion planning	[18, 43]
Commercial territory design	[52]
Location	[35, 42]
Statistics and finance	
Credit scoring	[58]
Distribution fitting	[21]
Parameter determination	[38]
Protein information prediction	[28]
Image registration	
Intensity-based image registration	[57]
Image segmentation	[5]
Craniofacial superimposition	[24]

As mentioned, scatter search operates on a set of solutions, *RefSet*, combining them to create new ones. The method basically performs iterations over the *RefSet* and can be summarized in three steps: select the solutions, combine them, and update the *RefSet* with the resulting solutions. As in many other metaheuristics, an

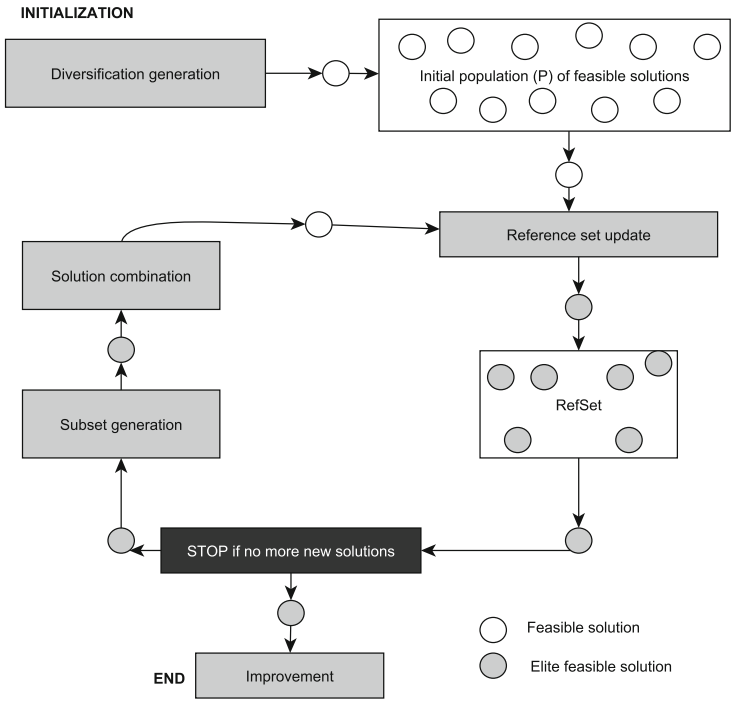

Fig. 3 Selective scatter search scheme

improvement method, or local search, can also be applied to improve the generated solutions. Here, the outcome of the combination method can be sent to a local search procedure to obtain a local optima.

Although the scatter search origins are linked with the mathematical programming area, in which convex combinations are very popular, convex and non-convex combinations are used in this method to generate new solutions. The non-convex combinations project new "centers" into regions of the solution space that are external to the original *RefSet* solutions, a strategy that is based on the principle of input diversity in the search, to avoid getting trapped in local optima.

The scatter search template provides a concise and direct description about how to implement the methodology. Figure 3 shows the scheme of our selective scatter search in which the improvement method is only applied at the end of the search. The methodology basically consists of five elements or methods and their associated strategies and may be sketched as follows:

- A diversification generation method (DGM):
  The DGM can be seen as the mechanism that creates the initial set of solutions P of an optimization problem. This is what we would call "a first generation" in the evolutionary terminology. If the attributes of this first generation are

good, there is some confidence of getting better solutions in the following iterations after strategically combining these initial solutions. The DGM typically implements frequency-based memory to generate P and employs controlled randomization. As opposed to the GAs, which usually rely in randomization to generate the initial population, the DGM samples the solution space in a systematic fashion. The size of P, $|P|$, is denoted as $Psize$.

- An improvement method (IM):

  The IM transforms a given solution of the problem, s, into an enhanced solution s'. 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 it is well known, given s, the local search tries to improve it by making "small changes" in its structure. These changes typically consist of adding elements to s, removing elements from s, changing the way in which elements are grouped in s, or changing their order, to mention a few. If an improvement is achieved in this way, then a new solution s' is obtained. This process of *small changes* is continued until no further improvement can be obtained or a time bound is elapsed. If no improvement of s was found, s would be the output.
- A reference set update method (RSUM):

  This method builds *RefSet* for the first time using a reference set creation subroutine (RSCsr) and updates *RefSet* during the search process. In the RSCsr, a given solution from P enters *RefSet* according to its quality or diversity. Many of the implementations of SS indicate that a good trade-off is to build 50 % of *RefSet* by quality criterion, and the remaining 50 % by a diversity criterion, but these proportions can be modified depending on the problem being solved.

  A standard mechanism to build the initial *RefSet*, whose size will be denoted by b, follows. The construction starts with the selection of the best $b/2$ solutions from P. For each solution in $P \setminus RefSet$, the minimum of the distances 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.

  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.
- A subset generation method (SGM):

  The SGM creates subsets of two or more solutions belonging to *RefSet*. These subsets will be subjected to the next SS process, the combination method. 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 being 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.
- A solution combination method (SCM):

The SCM creates new solutions from any subset of solutions previously created by the SGM. Although the process of creating new solutions that derive from other solutions can be achieved by a black-box procedure, it is generally more effective to design the SCM as a problem-specific mechanism, because it is directly related to the way we represent a solution in a problem. Depending on the specific form of the SCM, each subset can create one or more new solutions. This creation can be done in different ways, but the most common ones are systematic combinations of decision rules, path-relinking, and randomization. We refer the reader to [16, 50] for some combination methods within SS.

In this chapter, we illustrate the SS methodology by implementing its underlying framework to solve the uncapacitated r-allocation p-hub median problem (UrApHMP). Although we target here a specific variant of the well-known p-hub median problem, our results show that the scatter search methodology is well suited for solving this type of location problems. In line with our comment above, i.e., to encourage the reader to apply this methodology, here we can find a large family of problems in which the application of SS can result in a promising research area.

The Uncapacitated *r*-Allocation *p*-Hub Median Problem

Let $G = (V, E)$ be a network with set of nodes V and set of edges E. In the uncapacitated r-allocation p-hub median problem (UrApHMP), for each pair of nodes i and $j \in V$, there is an amount of traffic t_{ij} (generally of people or goods) that needs to be transported using this network. The cost of sending a unit of traffic from i to j is denoted by c_{ij}. It is assumed that direct transportation between i and j is not possible. This well-known assumption is based on the empirical evidence that it is not physically and/or economically viable, for example, to schedule a flight between any two pairs of cities or to add a link between any two computers for data transmission, since this would require a large amount of resources. Therefore, the traffic t_{ij} is routed along a path $i \to k \to l \to j$, where nodes k and $l \in V$ are used as intermediate points for this transportation. Examples of these intermediate nodes are some connecting airports such as JFK in New York, ORD in Chicago, and LHR in London; they all serve as transit points where passengers can connect from one flight to another in order to reach their final destinations. The UrApHMP consists of choosing a set H of nodes, ($H \subseteq V$, $|H| = p$), assigning r nodes in H to each node, and minimizing the total transportation cost of all the traffics of the network. The nodes in H, which can be used as intermediate transfer points between any pair of nodes in G, are commonly called *distribution centers* or *hub nodes*. The other nodes in the network are known as *terminal nodes*. For the sake of simplicity, we call them *hubs* and *terminals*, respectively.

Three optimization subproblems arise when solving the UrApHMP: a *location* problem to choose the best locations for the hubs, an *assignment* problem of each terminal to r of the hubs, and a *routing* problem to obtain the minimum cost route to use for transporting the traffics between any given pair of nodes. Regarding

the allocation strategy in the assignment process, the UrApHMP generalizes two extensively studied versions of the p-hub location problem: the *single* and *multiple* versions. In the single version ($r = 1$), each terminal is assigned to only one of the p hubs, thus allowing to send and receive the traffics through this hub. In contrast, in the multiple version ($r = p$), each terminal can send and receive traffics through any of the p hubs. In the version considered here, the *r-allocation*, each terminal is allowed to be allocated to r of the p hubs. The motivation of this version comes from the fact that the single allocation version is too restricted for real-world situations, while the multiple allocation version results in high fixed costs and complicated networks, which does not reflect the real models either. Yaman presented in [60] a study of allocation strategies and introduced the r-allocation version of the problem. The uncapacitated r-allocation p-hub median problem is formulated in [60] in terms of the following variables: Given a node $k \in V$, $z_{kk} = 1$ if node k is set to be a hub (i.e., if a hub is located at this node), and $z_{kk} = 0$ otherwise. Given a non-hub node $i \in V$, $z_{ik} = 1$ if node i is assigned to node k and 0 otherwise. Finally, f_{ijkl} is the proportion of the traffic t_{ij} from node i to node j that travels along the path $i \to k \to l \to j$, where k and l are the nodes that will be used as hubs. With these variables, the problem is formulated as follows:

$$\text{Min} \quad \sum_{i\in V}\sum_{j\in V}\sum_{k\in V}\sum_{l\in V} t_{ij}(\chi c_{ik} + \alpha c_{kl} + \delta c_{lj}) f_{ijkl} \tag{1}$$

$$\sum_{k\in V} z_{ik} \le r, \qquad \forall i \in V \tag{2}$$

$$z_{ik} \le z_{kk}, \qquad \forall i,k \in V \tag{3}$$

$$\sum_{k\in V} z_{kk} = p \tag{4}$$

$$\sum_{k\in V}\sum_{l\in V} f_{ijkl} = 1, \qquad \forall i,j \in V \tag{5}$$

$$\sum_{l\in V} f_{ijkl} \le z_{ik}, \qquad \forall i,j,k \in V \tag{6}$$

$$\sum_{k\in V} f_{ijkl} \le z_{jl}, \qquad \forall i,j,l \in V \tag{7}$$

$$f_{ijkl} \ge 0, \qquad \forall i,j,k,l \in V \tag{8}$$

$$z_{ik} \in \{0,1\}, \qquad \forall i,k \in V, \tag{9}$$

where χ, α, and δ are unit rates for collection (origin-hub), transfer (hub-hub) and distribution (hub-destination), respectively. Note that constraints (2) ensure that each node is allocated to at most r hubs, where hubs are selected according to (3). In addition, constraint (4) limits to p the number of hubs. Finally, constraints (5),

(6), and (7) are associated with the routing of the traffic between each pair of nodes i, j through their corresponding hubs k, l.

A Small Example with Real Data

With the aim of illustrating the UrApHMP in detail, we introduce a small instance of a network with ten nodes that can be used as the input data. Table 2 contains the traffics t_{ij} and the unitary transportation costs c_{ij} for any pair of nodes (i, j) of this network. The number of nodes to be used as hubs, p, is an input of the problem, as well as r, the number of hubs a given terminal can be assigned to. Assume that $p = 3$ and $r = 2$ and that nodes 3, 6, and 8 will be used as hubs, i.e., $H = \{3, 6, 8\}$. For any pair of nodes, for example, nodes 2 and 5, $t_{2,5} = 18$ is obtained from the table, which means that there are 18 passengers that need to be transported from node 2 to node 5. To compute the cost of transporting $t_{2,5}$, we first need to assign each terminal to r of the p hubs and each hub to all the hubs including itself. For this example, let terminal 2 be assigned to hubs 3 and 6 and terminal 5 to hubs 3 and 8. If we denote by H^i the set of hubs to which node i is assigned to, $H^2 = \{3, 6\}$ and $H^5 = \{3, 8\}$. Then, to transport the corresponding traffics, we need to explore all possible paths in order to know the most inexpensive one. This information is summarized in Table 3, where costs are separated into the three terms described above: c_{ik}, c_{kl}, and c_{lj}. Then, the cost is computed for each path, by applying the

Table 2 Example of a traffic and cost matrices derived from an AP instance

	i, j	1	2	3	4	5	6	7	8	9	10
t_{ij}	1	75	37	55	19	20	18	57	17	19	16
	2	26	38	25	27	18	23	38	20	12	17
	3	67	39	51	22	24	21	71	20	23	19
	4	17	33	19	25	16	23	40	23	11	19
	5	17	25	21	19	23	18	73	20	24	19
	6	12	18	12	16	11	17	37	22	10	18
	7	82	78	90	68	95	73	312	99	110	110
	8	35	42	36	48	36	58	173	90	41	92
	9	12	12	15	10	19	11	68	15	24	20
	10	22	25	23	28	23	33	109	51	30	63
c_{ij}	1	0	20	16	23	23	30	32	33	36	36
	2	20	0	20	13	25	15	30	25	38	31
	3	16	20	0	14	7	19	16	18	21	21
	4	23	13	14	0	14	7	18	13	27	18
	5	23	25	7	14	0	16	9	13	14	14
	6	30	15	19	7	16	0	16	8	25	13
	7	32	30	16	18	9	16	0	9	10	7
	8	33	25	18	13	13	8	9	0	18	5
	9	36	38	21	27	14	25	10	18	0	14
	10	36	31	21	18	14	13	7	5	14	0

Table 3 Possible paths and their associated costs between terminals 2 and 5

Path	Collect	Transfer	Dist.	Cost computation
$2 \rightarrow 3 \rightarrow 8 \rightarrow 5$	20	18	13	$(3 \times 20) + (0.75 \times 18) + (2 \times 13) = 99.50$
$2 \rightarrow 3 \rightarrow 3 \rightarrow 5$	20	0	7	$(3 \times 20) + (0.75 \times 0) + (2 \times 7) = 74.00$
$2 \rightarrow 6 \rightarrow 3 \rightarrow 5$	15	19	7	$(3 \times 15) + (0.75 \times 19) + (2 \times 7) = 73.25$
$2 \rightarrow 6 \rightarrow 8 \rightarrow 5$	15	8	13	$(3 \times 15) + (0.75 \times 8) + (2 \times 13) = 77.00$

unit rate coefficients $\chi = 3$, $\alpha = 0.75$, and $\delta = 2$. As can be seen in this table, path $2 \rightarrow 6 \rightarrow 3 \rightarrow 5$ is the one with the minimum transportation cost (73.25). It is therefore the selected path to route the traffic $t_{2,5}$. Note that there is a path $(2 \rightarrow 3 \rightarrow 3 \rightarrow 5)$ in which both terminals share a common hub, but it is not the most inexpensive one.

The process of checking the paths between any two pair of nodes $i, j \in V$ needs to be computed for any possible assignments and for any possible set of hubs, in order to find the combination minimizing the total cost of transporting all the traffics in the network. This massive calculation gives an idea of the combinatorial nature of the problem. Even when the set of hubs is given, the subproblem of assignment of terminals to hubs is also $\mathcal{NP}$-hard [40].

To illustrate how large is the search space that we are exploring in this problem, consider a medium size instance in which $n = 100$, $p = 5$, and $r = 3$. The location problem, in which we have to select the 5 hubs out of the 100 nodes, gives us 75,287,520 combinations. Then, for each combination we have to solve the assignment problem, in which we have to assign each node to 3 of the 5 hubs selected, giving 10 combinations for each node, which makes 950 possibilities. Finally, for each combination, and each assignment, we have to solve the routing problem. To do that, we have to consider that each node is assigned to 3 hubs, and, therefore, for each pair of nodes, we have 9 routes, which makes a total of 89,100 possible routes in this network with 100 nodes. The total number of combinations, or solutions in the search space, is therefore $75{,}287{,}520 \times 950 \times 89{,}100 = 6.3 \times 10^{15}$. This example also illustrates that the main source of difficulty comes from the first problem, the location, which contributes with the largest factor in this computation.

Scatter Search for the *p*-Hub Problem

As described in section "The Diversification Generator Method", and shown in Fig. 3, the SS method starts by generating a set P of diverse solutions. Then, b of them are selected to create *RefSet*. In our implementation for the UrApHMP, we follow the standard design of selecting the best $\frac{b}{2}$ solutions in P and then the most diverse $\frac{b}{2}$ solutions with respect to the solutions already in *RefSet* (see section "Reference Set Initialization"). The main loop of the method consists of

applying the combination method to all pairs of solutions in *RefSet*, and update this set with the new solutions obtained from these combinations. The method finishes when no new solutions are added to *RefSet*.

The Diversification Generator Method

Recall that three optimization subproblems arise when solving the UrApHMP. We base our solution representation in these three subproblems: location, assignment, and routing. Specifically, a solution s is coded as:

- An array H containing the set of nodes to be used as hubs.
- For any node i of the network, an array $H^i \subseteq H$ containing the hubs to which node i is assigned to.
- For any given pair of nodes (i, j), the pair of hubs that have been used to route t_{ij}.

With this representation, it is easy to see that a straightforward approach to construct a feasible solution for the problem is to select first the nodes to be used as hubs, to assign then each terminal to r of the p hubs, and, finally, to route the traffics through the network. Following this rationale, we create a population P of feasible solutions. To do this, we have developed and tested three constructive methods. Two of them are based on a greedy randomized construction [10,11] based on different expressions for the evaluation of the candidate nodes to be hubs. The third one is simply a random construction to provide diversity to P. Each hub of a solution is selected by evaluating all nodes with respect to a greedy function g that measures their attractiveness to be hub. Only the q elements with best g values are placed in a restricted candidate list (RCL), where q is a search parameter. Lower q values favor greedy selection. Then, an element in the RCL is randomly selected, according to a uniform distribution, to become part of the solution. The values of g are updated at each iteration to reflect the changes brought on by the selection of previous elements.

Let $h \in V$ be a candidate node to be a hub. In that case, it would be used for the transportation of the traffics among some terminals, let say the $\lfloor \frac{n}{p} \rfloor$ terminals $(i_1, \ldots, i_{\lfloor \frac{n}{p} \rfloor})$ with the lowest assignment cost to h. We then compute $g(h)$ as

$$g(h) = \sum_{s=1}^{s=\lfloor \frac{n}{p} \rfloor} \text{cost}(i_s, h), \quad \forall h \in V,$$

where $\text{cost}(i, h)$ represents the assignment cost of terminal i to hub h, and can be computed in two ways, leading to constructive methods DGM1 and DGM2.

DGM1 In this method, $\text{cost}(i,h)$ is computed as the cost of sending and receiving the traffics of i through h, i.e., $\text{cost}(i,h) = c_{ih}\overrightarrow{T_i} + c_{hi}\overleftarrow{T_i}$, where $\overrightarrow{T_i} = \sum_{j\in V} t_{ij}$ is the sum of all the traffics from i to all nodes j and $\overleftarrow{T_i} = \sum_{j\in V} t_{ji}$ is the sum of all the traffics from all nodes j to node i.

DGM2 In this method, $\text{cost}(i,h)$ is computed considering the discounting factors that are usually present in the UrApHMP. To do this, we modify the cost expression to $\text{cost}(i,h) = \chi c_{ih}\overrightarrow{T_i} + \dfrac{\alpha+\delta}{2} c_{hi}\overleftarrow{T_i}$.

DGM3 The third method to generate solutions is based on the notion of constructing solutions by simply generating random sets of p hubs. Its purpose is to create a limited amount of solutions not guided by an evaluation function to just bring diversity into P.

Once the p hubs are selected, the next step is to allocate r of the p hubs to each terminal. For any terminal i, we compute the following estimation of the assignment cost of i to a hub h as

$$\text{assignment}(i,h) = c_{ih}\overrightarrow{T_i} + \sum_{j\in V} c_{hj}t_{ij}.$$

Then, we assign i to the hub h_a with the lowest assignment cost, and the above expression is updated to reflect previous assignments:

$$\text{assignment}(i,h) = c_{ih}\overrightarrow{T_i} + \sum_{j\in V\setminus H^i} c_{hj}t_{ij} - \sum_{u\in H^i} c_{iu}t_{iu}, \quad \forall h \in H \setminus H^i.$$

This process is performed in a greedy way, selecting at each iteration the lowest assignment cost. Note that the assignment expressions are estimations, because they assume that there is only one hub h in the path between any pair of nodes i and j, which is not necessarily true.

Finally, we route all the traffics at their minimum cost. For each pair of nodes i and j, we have to determine the hubs $k \in H^i$ and $l \in H^j$ minimizing the routing cost of the traffic sent from i to j. In mathematical terms, given $k \in H^i$ and $l \in H^j$, we denote as $\text{routecost}_{ij}(k,l)$ the cost of transporting the traffics from i to j through hubs k and l, i.e.,

$$\text{routecost}_{ij}(k,l) = t_{ij}(\chi c_{ik} + \alpha c_{kl} + \delta c_{lj}).$$

The routing cost from i to j, routecost_{ij}, is then obtained by searching the hubs $k \in H^i$ and $l \in H^j$ minimizing the expression above, i.e.,

$$\text{routecost}_{ij} = \min_{k\in H^i, l\in H^j} \text{routecost}_{ij}(k,l).$$

Note that the time complexity of evaluating the routing costs if $\chi \neq \delta$ is $\mathcal{O}(n^2r^2)$, since the path from j to i does not necessarily involve the same hubs, k and l, than the path $i \to k \to l \to j$.

Once the p hubs have been located, r hubs have been assigned to each node, and all the traffics have been routed, we have a feasible solution s for the UrApHMP. It is denoted as $s = (H, A)$, and its cost by $f(s)$, where $H = \{h_1, \ldots, h_p\} \subseteq V$ is the set of hubs in the solution and A is the matrix whose rows contain the r hubs assigned to each node.

Reference Set Initialization

As mentioned above, to create the reference set, *RefSet*, the notion of *best* solutions is not limited to their quality, as measured by the objective function, since it also considers their diversity. In particular, our method first chooses $\frac{b}{2}$ solutions attending to their quality. We simply order the solutions in P by their costs, and introduce them, one by one, in *RefSet*, if there is no other solution already in *RefSet* with the same cost. We stop this selection process after examining the 50 % of the solutions in P, even if those selected are less than $\frac{b}{2}$ solutions. The rest of the solutions of the *RefSet* are then selected from P by a diversity criterion as follows.

Given $s \notin RefSet$ and $t \in RefSet$, let $C = \{h : h \in H_s \cap H_t\}$ be the set of common hubs in solutions s and t. We define $d_H(s,t) = p - |C|$ as the number of hubs in s not present in t (and vice versa). Note that the lower the value of $d_H(s,t)$ is, the closer s and t are. To select the solution $s \in P$ to be included in *RefSet*, we define $\text{dist}(s, RefSet) = \min_{t \in RefSet} d_H(s,t)$. This distance is computed for all solutions in P, and the solution s^* with maximum value of $\text{dist}(s, RefSet)$ is introduced in *RefSet*.

The Subset Generation Method

The SGM we propose works by generating subsets defined by any two different solutions in *RefSet*. To avoid the repetition of previously generated subsets in previous iterations, each subset is generated only if at least one of its solutions was introduced in *RefSet* in the preceding iteration. Suppose that m subsets were not considered for this reason, then the number of resulting subsets at each iteration for which the combination method will be applied is $\frac{b^2-b}{2} - m$.

The Solution Combination Method

Other methodologies, such as genetic algorithms, base their designs on "generic" combination procedures, also called context independent or black-box operators. On

the contrary, scatter search is based on specific combination methods that exploit the characteristics of the problem being solved. The solutions obtained from the combinations in each subset are stored in a set called $Pool$.

Let $U = \{h : h \in H_s \cup H_t\}$ and $I = \{h : h \in H_s \cap H_t\}$. We propose two combination methods for a pair (s, t) of solutions in a subset. They are described in what follows:

- **Method 1**: This method is applied when $|U| > p$. It creates a solution with the hubs in U. It can be considered as a "convex combination" of s and t. In particular, it selects the p elements in U with best evaluation to be hubs, where candidates are evaluated with the same g function introduced in the diversification generation method.
- **Method 2**: This method is applied when $|I| < p$. It creates a solution by including all the elements in I as hubs and selects the rest $p - |I|$ hubs from $V \setminus I$. As in Method 1, candidate hubs are evaluated with g and the best ones are selected in a greedy fashion. The output can be considered as a "non-convex combination" of solutions s and t.

The Reference Set Update Method

The *RefSet* update operation consists of maintaining a record of the b best solutions found so far by the procedure. The issues related to this updating function are straightforward: All the solutions in *RefSet* that are worse than those in the current *Pool* will be replaced by these ones, with the aim of keeping in *RefSet* the b best and most different solutions found so far. Let $s \in Pool$ and $w \in RefSet$ such that the objective function value of s is less than that of w. In this case, s will replace w if s is different from all other solutions in *RefSet*. Note that the update method focuses on quality. In other words, although *RefSet* was created considering both quality and diversity, when updating it, we only consider the objective function value of the candidate solutions obtained from the combination method. In particular, once all the combinations have been performed and the new solutions have been moved to $Pool$, the new *RefSet* simply contains the best b solutions in the set formed with the previous $RefSet \cup Pool$. This strategy, based on quality, favors the convergence of the method.

The Improvement Method

As mentioned in section "The Uncapacitated r-Allocation p-Hub Median Problem," three optimization subproblems arise when solving the UrApHMP. Since we solve the routing subproblem optimally, we propose two improvement procedures based on local search strategies for the other two subproblems: LS_H for the hub selection

and LS_A for the terminal allocations. Both are based on the local search procedures proposed in [48].

LS_H implements a classical exchange procedure in which a hub h_i is removed from H and a non-hub $h'_i \in N \backslash H$ replaces h_i, thus obtaining $H' = \{h_1, h_2, \ldots, h'_i, \ldots, h_p\}$. When hub h_i is replaced with h'_i, we have to reevaluate the hub assignment for the vertices assigned to h_i. Moreover, the routes for the traffics are not necessarily optimal for the new set of hubs H', and, hence, they have to be recomputed. The local search procedure LS_H performs moves as long as the cost improves. We have implemented here the so-called *first strategy*, in which the first improving move in the neighborhood is performed.

The local search procedure LS_A is similar to LS_H, but it considers changes in the assignment of terminals to hubs. In particular, for a node i with $H^i = \{h_{i1}, \ldots, h_{ia}, \ldots, h_{ir}\}$, this procedure exchanges an assigned hub with a non-assigned one. In mathematical terms, we replace h_{ia} with $\bar{h}_{ia} \in H \backslash H^i$, thus obtaining $\bar{H}^i = \{h_{i1}, \ldots, \bar{h}_{ia}, \ldots, h_{ir}\}$. As in LS_H, the method performs moves while improving and returns the local optimum reached as its output.

Computational Experiments

This section describes the computational experiments performed to first find the parameter values of the scatter search method, and then study its performance, especially with respect to previous approaches. The procedure has been implemented in C using GCC 4.8.2, and the results reported in this section have been obtained with an Intel Core i7–3770 at 3.40 GHz and 16 GB of RAM under Ubuntu 14.04 GNU/Linux – 64-bit operating system. The metrics we use to measure the performance of the algorithms are:

- Dev: Average percentage deviation with respect to the best solution found (or from the optimal solution, if available).
- # Best: Number of best solutions found.
- CPU: Average computing time in seconds.

We have tested our algorithms on the three sets of instances previously reported [8, 45, 48]. The **CAB** (Civil Aviation Board) data set, based on airline passenger flows among some important cities in the United States, consists of 23 instances with 25 nodes and $p = \{1, \ldots, 5\}$ and $r = \{1, \ldots, p\}$. The **AP** (Australian Post) data set is based on real data from the Australian postal service. We have extended this set by generating, from the original file, 311 instances with $40 \leq n \leq 200$ and $1 \leq p \leq 20$. The third data set, **USA423**, consists of a data file concerning 423 cities in the United States, where real distances and passenger flows for an accumulated 3 months' period are considered. From the original data, 30 instances have been generated with $p \in \{3, 4, 5, 6, 7\}$. The total set of 264 instances has been divided

into three subsets: small ($25 \leq n \leq 50$), medium ($55 \leq n \leq 100$), and large ($105 \leq n \leq 200$). The entire set of instances is available at www.optsicom.es.

The experiments are divided into two main blocks. The first block, described in section "Parameter Calibration", is devoted to study the behavior of the components of the solution procedure, as well as to determine the best values for the search parameters. The second block of experiments, reported in section "Comparison with a GRASP Algorithm", compares our procedure with the best published methods.

Parameter Calibration

The first set of experiments to calibrate our method is performed on a subset of 47 instances: five instances from the CAB set with $n = 25$ and 42 instances with $40 \leq n \leq 200$ from the AP set. We refer to these 47 instances as the *training set* and to the remaining instances as the *testing set*.

The Size of *P*

First, we study the value of the parameter used in the diversification generator method, *Psize*, which determines the number of solutions in *P*. We have tested four values: 50, 100, 150, and 200. For each instance, we generate $PSize/3$ solutions with each diversification generation method (DGM1, DGM2, and DGM3, respectively). In order to evaluate DGM methods in isolation (without the combination and improvement methods), this experiment only considers the constructive phase. The results on the training set are shown in Table 4.

As expected (see Table 4), the best solutions in terms of quality are obtained with $Psize = 200$. In this case, the algorithm obtains an average percentage deviation of 1.6 % and 20 best-known solutions. The CPU time for $Psize = 200$ is still reasonable, with virtually no difference in small and medium instances. Although for the large instances, the CPU values are slightly larger, we consider that the improvement of the results deserves the CPU effort, so we set $Psize = 200$ in the rest of the experiments.

The Size of *RefSet*

In order to study the size of *RefSet*, b, we include all the elements of scatter search in the algorithm except the local search procedures (to avoid the effects of

Table 4 Calibration of *Psize*

		Dev (%)				CPU				# best			
Size	# inst	50	100	150	200	50	100	150	200	50	100	150	200
Small	8	3.8	2.5	0.7	1.6	0.01	0.02	0.03	0.03	0	3	5	4
Medium	27	3.8	3.9	2.6	1.4	0.08	0.16	0.24	0.33	1	4	8	14
Large	12	4.2	1.9	2.0	2.2	0.47	0.95	1.45	1.96	3	4	3	2
Summary	47	3.9	3.2	2.1	1.6	0.17	0.34	0.51	0.69	4	11	16	20

Table 5 Calibration of b

		Dev (%)						CPU						# best					
Size	# inst	5	6	7	8	9	10	5	6	7	8	9	10	5	6	7	8	9	10
Small	8	0.1	0.1	0.1	0.0	0.0	0.1	0.04	0.03	0.04	0.05	0.05	0.06	7	7	7	8	8	7
Medium	27	3.1	2.4	2.5	1.5	1.0	0.8	0.31	0.35	0.38	0.42	0.45	0.51	4	6	6	12	12	12
Large	12	1.6	1.7	1.6	0.4	0.3	0.7	1.81	1.97	2.11	2.32	2.57	2.65	3	5	6	9	10	7
Summary	47	2.2	1.8	1.8	1.0	0.7	0.6	0.65	0.71	0.77	0.84	0.92	0.98	14	18	19	29	30	26

Table 6 Comparison of strategies SS^A and SS^B

		Dev (%)		CPU		# best	
Size	# inst	SS^A	SS^B	SS^A	SS^B	SS^A	SS^B
Small	8	0.0	0.6	0.23	0.06	8	4
Medium	27	0.0	0.1	6.32	1.01	27	17
Large	12	0.0	0.1	57.15	8.84	12	7
Summary	47	0.0	0.2	18.26	2.85	47	28

the improvement methods). Table 5 shows the results for different values of this parameter ($b = 5, 6, \ldots, 10$). As it can be seen, the higher the value of b, the better are the results. Obviously, this implies higher computing times. So, we choose $b = 6$, since this value results in a good trade-off between solution quality and CPU time.

Local Search Methods

We now study the effect of the improvement procedures described in section "The Improvement Method". Recall that two local searches have been proposed, one for the hub selection (LS_H) and another for the terminal allocations (LS_A). Although the standard SS design specifies to apply the improvement method to all the solutions resulting from the combination method, considering that the local searches for the p-hub problem are quite time consuming, we limit their application to the best solutions across all the global iterations. In particular, we only apply LS_H and LS_A to the solutions in the final *RefSet* (i.e., just before the end of the algorithm). To evaluate the efficiency of this selective strategy, we have studied the following two variants:

SS^A: LS_H and LS_A are applied to all the solutions in the final *RefSet*.
SS^B: LS_H and LS_A are applied only to the best solution in the final *RefSet*.

Table 6 shows the results obtained on the 47 instances of the training set with both methods. The results indicate that applying the improvement methods to all the solutions in *RefSet* substantially improves their value. Variant SS^A exhibits a larger number of best solutions found compared to variant SS^B. Nevertheless, this exhaustive application of the local searches results in much higher CPU times (from

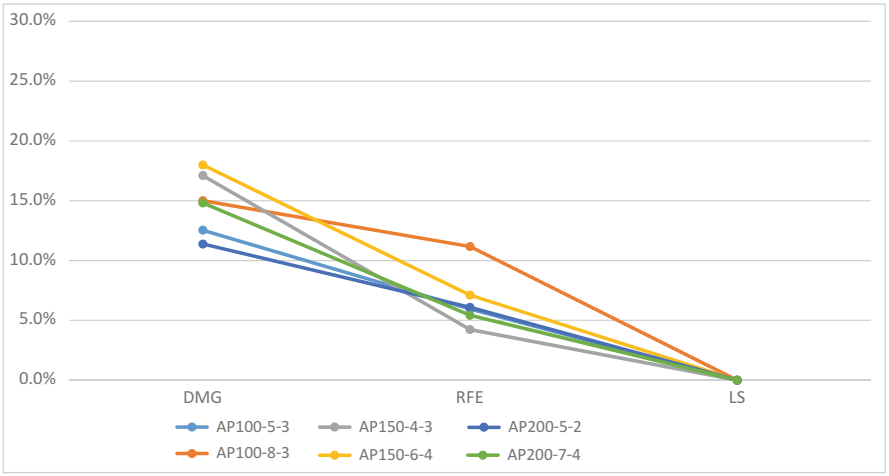

Fig. 4 Search profile for six different instances

2.85 to 18.26 s in average). Despite the fact that variant SS^B is not able to get some of the best-known solutions, its average deviation is very small (from 0.1 % to 0.6 %). We keep both variants in the following comparison with a previous method.

To complement the analysis above, we now explore the contribution on the quality of the final solution of the three main methods in SS: construction, combination, and improvement. Figure 4 shows, for six representative instances, the value of the best solution obtained with the diversification generation method (DGM), the value of the best solution obtained with the combination method (REF), and the value after the application of the local search to the solutions in the final *RefSet*. This clearly illustrates how the three methods contribute to reach the final high-quality solutions.

Comparison with a GRASP Algorithm

After the calibration process described before, we now compare the performance of our SS algorithms with the GRASP proposed in [48], which as far as we know is the best heuristic algorithm for the UrApHMP. All the methods under comparison are run in the same computer on the entire set of 194 instances. The results of both scatter search variants, SS^A and SS^B, and the GRASP are summarized in Table 7. This table clearly shows that our SS procedures outperform the previously proposed GRASP method in terms of quality and number of best solution found. In particular, both SS procedures obtain a larger number of best solutions and a lower average deviation than GRASP. SS^B is very competitive since it is able to obtain very good solutions in short computing times.

Table 7 Comparison between the SS and a GRASP method

		Dev (%)			CPU			# best		
Size	# inst	SS^A	SS^B	GRASP	SS^A	SS^B	GRASP	SS^A	SS^B	GRASP
Medium	116	0.00	0.10	0.28	3.8	0.6	17.1	98	73	18
Large	48	0.00	0.09	0.31	34.3	4.5	31.6	43	21	5
Extra-large	30	0.21	4.42	4.54	679.5	91.4	607.4	19	4	11
Summary	194	0.03	0.77	0.95	115.8	15.6	112.0	160	98	34

Conclusions

In this chapter, we summarize our experiences on the subject of implementing scatter search. We have experimented with this methodology for a number of years, and as a result of implementing different search strategies, and performing extensive experimental testing, we have learned some valuable lessons condensed above. As it is well known, metaheuristic methodologies are based on principles and not necessarily on theory that can be spelled out with theorems and proofs. Hence, to illustrate and prove the efficiency of search mechanisms, strategies, and key elements in scatter search, we have shown how to solve an important and difficult combinatorial optimization problem: the uncapacitated r-allocation p-hub median problem. In particular, we describe in detail how to construct, improve, and combine solutions for this $\mathcal{NP}$-hard problem in an efficient way, i.e., with fast methods that are able to produce high-quality solutions. These research opportunities carry with them an emphasis on producing systematic and strategically designed rules, rather than following the policy of relegating decisions to random choices, as often is fashionable in evolutionary methods.

Acknowledgements This work was supported by the Spanish Ministerio de Economia y Competitividad (projects TIN-2012-35632-C02 and MTM-2012-36163-C06-02 and grant BES-2013-064245) and by the Generalitat Valenciana (project Prometeo 2013/049).

Cross-References

▸ Evolution Strategies
▸ Genetic Algorithms
▸ Network Optimization
▸ Implementation of Metaheuristics

References

1. Martí R (ed) (2006) Scatter search methods for optimization. Volume 169 of feature cluster of the European Journal of Operational Research. Elsevier, Amsterdam
2. Alkhazaleh HA, Ayob M, Ahmad Z (2013) Scatter search for solving team orienteering problem. Res J Appl Sci 8(3):181–190
3. Bernardino AM, Bernardino EM, Sánchez-Pérez JM, Gómez-Pulido JA, Vega-Rodríguez MA (2012) Solving sonet problems using a hybrid scatter search algorithm. In: Madani K, Dourado CA, Rosa A, Filipe J (eds) Computational intelligence. Volume 399 of studies in computational intelligence. Springer, Berlin/Heidelberg, pp 81–97
4. Botón-Fernández M, Vega-Rodríguez MA, Prieto-Castrillo F (2013) An efficient and self-adaptive model based on scatter search: solving the grid resources selection problem. In: Moreno-Díaz R, Pichler F, Quesada-Arencibia A (eds) Computer aided systems theory – EUROCAST 2013. Volume 8111 of lecture notes in computer science. Springer, Berlin/Heidelberg, pp 396–403
5. Bova N, Ibáñez Ó, Cordón Ó (2013) Image segmentation using extended topological active nets optimized by scatter search. IEEE Comput Intell Mag 8(1):16–32
6. de Athayde Costa e Silva M, Klein CE, Mariani VC, Dos Santos Coelho L (2013) Multiobjective scatter search approach with new combination scheme applied to solve environmental/economic dispatch problem. Energy 53:14–21
7. Engin O, Ylmaz MK, Baysal ME, Sarucanl A (2013) Solving fuzzy job shop scheduling problems with availability constraints using a scatter search method. J Multiple-Valued Log Soft Comput 21(3–4):317–334
8. Ernst AT, Krishnamoorthy M (1996) Efficient algorithms for the uncapacitated single allocation p-hub median problem. Locat Sci 4(3):139–154
9. Everett H (1963) Generalized Lagrangean multiplier method for solving problems of optimal allocation of resources. Oper Res 11:399–417
10. Feo TA, Resende MGC (1995) Greedy randomized adaptive search procedures. J Glob Optim 6(2):109–133
11. Festa P, Resende MGC (2011) Grasp: basic components and enhancements. Telecommun Syst 46(3):253–271
12. Glover F (1965) A multiphase-dual algorithm for the zero-one integer programming problem. Oper Res 13:879–919
13. Glover F (1977) Heuristics for integer programming using surrogate constraints. Decis Sci 8(7):156–166
14. Glover F (1994) Tabu search for nonlinear and parametric optimization with links to genetic algorithms. Discret Appl Math 49(1–3):231–255
15. Glover F (1998) A template for scatter search and path relinking. In Hao JK, Lutton E, Ronald E, Schoenauer M, Snyers D (eds) Artificial Evolution. Volume 1363 of Lecture Notes in Computer Science. Springer, Berlin/Heidelberg, pp 13–54.
16. Glover F, Laguna M, Martí R (2000) Fundamentals of scatter search and path relinking. Control Cybern 29(3):652–684
17. Guo XW, Liu SX, Wang DZ (2012) Scatter search for solving multi-objective disassembly sequence optimization problems. J Northeast Univ 33(1):56–59
18. Habibi MR, Rashidinejad M, Zeinaddini-Meymand M, Fadainejad R (2014) An efficient scatter search algorithm to solve transmission expansion planning problem using a new load shedding index. Int Trans Electr Energy Syst 24(2):153–165
19. Hariharan R, Golden Renjith Nimal RJ (2014) Solving flow shop scheduling problems using a hybrid genetic scatter search algorithm. Middle East J Sci Res 20(3):328–333
20. Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor
21. Hu L, Jiang Y, Zhu J, Chen Y (2013) Hybrid of the scatter search, improved adaptive genetic, and expectation maximization algorithms for phase-type distribution fitting. Appl Math Comput 219(10):5495–5515

22. Huacuja HJF, Valdez GC, Rangel RAP, Barbosa JG, Reyes LC, Valadez JMC, Soberanes HJP, Villanueva DT (2012) Scatter search with multiple improvement methods for the linear ordering problem. Malays J Comput Sci 25(2):76–89
23. Hvattum LM, Duarte A, Glover F, Martí R (2013) Designing effective improvement methods for scatter search: An experimental study on global optimization. Soft Comput 17(1):49–62
24. Ibáñez O, Cordón O, Damas S, Santamaría J (2012) An advanced scatter search design for skull-face overlay in craniofacial superimposition. Expert Syst Appl 39(1):1459–1473
25. Jabal-Ameli MS, Moshref-Javadi M (2014) Concurrent cell formation and layout design using scatter search. Int J Adv Manuf Technol 71(1–4):1–22
26. Jabal-Ameli MS, Moshref-Javadi M, Tabrizi BB, Mohammadi M (2013) Cell formation and layout design with alternative routing: a multi-objective scatter search approach. Int J Ind Syst Eng 14(3):269–295
27. Jaradat G, Ayob M, Ahmad Z (2014) On the performance of scatter search for post-enrolment course timetabling problems. J Comb Optim 27(3):417–439
28. Kashefi AH, Meshkin A, Zargoosh M, Zahiri J, Taheri M, Ashtiani S (2013) Scatter-search with support vector machine for prediction of relative solvent accessibility. EXCLI J 12:52–63
29. Kothari R, Ghosh D (2014) A scatter search algorithm for the single row facility layout problem. J Heuristics 20(2):125–142
30. Krishnan M, Karthikeyan T, Chinnusamy TR, Venkatesh Raja K (2012) A novel hybrid metaheuristic scatter search-simulated annealing algorithm for solving flexible manufacturing system layout. Eur J Sci Res 73(1):52–61
31. Laguna M, Gortázar F, Gallego M, Duarte A, Martí R (2014) A black-box scatter search for optimization problems with integer variables. J Glob Optim 58(3):497–516
32. Laguna M, Martí R (2002) Scatter search: methodology and implementations in C. Kluwer Academic, Norwell
33. Le Q, Yang G, Hung W, Guo W (2012) Reliable NoC mapping based on scatter search. In: Liu B, Ma M, Chang J (eds) Information computing and applications. Volume 7473 of lecture notes in computer science. Springer, Berlin/Heidelberg, pp 640–647
34. Le Q, Yang G, Hung WNN, Zhang X, Fan F (2014) A multiobjective scatter search algorithm for fault-tolerant noc mapping optimisation. Int J Electron 101(8):1056–1073
35. Li J, Prins C, Chu F (2012) A scatter search for a multi-type transshipment point location problem with multicommodity flow. J Intell Manuf 23(4):1103–1117
36. Li VC, Liang YC, Sun YY, Chen YS (2012) A scatter search method for the multidimensional knapsack problem with generalized upper bound constraints. J Chin Inst Ind Eng 29(8):559–571
37. Li ZZ, Song XY, Sun JZ, Huang ZT (2012) A scatter search methodology for the aircraft conflict resolution problem. In: Huang D, Jiang C, Bevilacqua V, Figueroa JC (eds) Intelligent computing technology. Volume 7389 of lecture notes in computer science. Springer, Berlin/Heidelberg, pp 18–24
38. Lin SW, Chen SC (2012) Parameter determination and feature selection for c4.5 algorithm using scatter search approach. Soft Comput 16(1):63–75
39. Liu Q, Wang WX, Zhu KR, Zhang CY, Rao YQ (2014) Advanced scatter search approach and its application in a sequencing problem of mixed-model assembly lines in a case company. Eng Optim 46(11):1485–1500
40. Love RF, Morris JG, Wesolowski GO (1988) Facilities location: models and methods. Elsevier, New York
41. Lv Y, Wang G, Tang L (2014) Scenario-based modeling approach and scatter search algorithm for the stochastic slab allocation problem in steel industry. ISIJ Int 54(6):1324–1333
42. Martí R, Corberán Á, Peiró J (2015) Scatter search for an uncapacitated p-hub median problem. Comput Oper Res 58(0):53–66
43. Meymand MZ, Rashidinejad M, Khorasani H, Rahmani M, Mahmoudabadi A (2012) An implementation of modified scatter search algorithm to transmission expansion planning. Turk J Electr Eng Comput Sci 20(Suppl.1):1206–1219

44. Naderi B, Ruiz R (2014) A scatter search algorithm for the distributed permutation flowshop scheduling problem. Eur J Oper Res 239(2):323–334
45. O'Kelly ME (1987) A quadratic integer program for the location of interacting hub facilities. Eur J Oper Res 32(3):393–404
46. Pantrigo JJ, Duarte A (2013) Low-level hybridization of scatter search and particle filter for dynamic TSP solving. In: Alba E, Nakib A, Siarry P (eds) Metaheuristics for dynamic optimization. Volume 433 of studies in computational intelligence. Springer, Berlin/Heidelberg, pp 291–308
47. Pantrigo JJ, Martí R, Duarte A, Pardo EG (2012) Scatter search for the cutwidth minimization problem. Ann Oper Res 199(1):285–304
48. Peiró J, Corberán A, Martí R (2014) GRASP for the uncapacitated r-allocation p-hub median problem. Comput Oper Res 43(1):50–60
49. Pendharkar PC (2013) Scatter search based interactive multi-criteria optimization of fuzzy objectives for coal production planning. Eng Appl Artif Intell 26(5–6):1503–1511
50. Resende MGC, Ribeiro CC, Glover F, Martí R (2010) Scatter search and path-relinking: fundamentals, advances, and applications. In: Gendreau M, Potvin JY (eds) Handbook of metaheuristics. International series in operations research & management science, vol 146. Springer, New York, pp 87–107
51. Sadiq AT, Sagheer AM, Ibrahim MS (2012) Improved scatter search for 4-colour mapping problem. Int J Reasoning-based Intell Syst 4(4):221–226
52. Salazar-Aguilar MA, Ríos-Mercado RZ, González-Velarde JL, Molina J (2012) Multiobjective scatter search for a commercial territory design problem. Ann Oper Res 199(1):343–360
53. Shen Z, Zou H, Sun H (2012) Task scheduling for imaging reconnaissance satellites using multiobjective scatter search algorithm. In: Li Z, Li X, Liu Y, Cai Z (eds) Computational intelligence and intelligent systems. Communications in computer and information science. Springer, Berlin/Heidelberg, pp 240–249
54. Tan Y, Cheng TCE, Ji M (2014) A multi-objective scatter search for the ladle scheduling problem. Int J Prod Res 52(24):7513–7528
55. Tang J, Zhang G, Zhang B, Cen H (2012) Resource-constrained project scheduling using electromagnetism-based scatter search. J Comput Inf Syst 8(12):5219–5227
56. Tavakkoli-Moghaddam R, Ranjbar-Bourani M, Amin GR, Siadat A (2012) A cell formation problem considering machine utilization and alternative process routes by scatter search. J Intell Manuf 23(4):1127–1139
57. Valsecchi A, Damas S, Santamaría J, Marrakchi-Kacem L (2014) Intensity-based image registration using scatter search. Artif Intell Med 60(3):151–163
58. Wang J, Hedar AR, Wang S, Ma J (2012) Rough set and scatter search metaheuristic based feature selection for credit scoring. Expert Syst Appl 39(6):6123–6128
59. Xu Y, Qu R (2012) A hybrid scatter search meta-heuristic for delay-constrained multicast routing problems. Appl Intell 36(1):229–241
60. Yaman H (2011) Allocation strategies in hub networks. Eur J Oper Res 211(3):442–451
61. Zhang T, Chaovalitwongse WA, Zhang Y (2012) Scatter search for the stochastic travel-time vehicle routing problem with simultaneous pick-ups and deliveries. Comput Oper Res 39(10):2277–2290