DEPARTMENT OF ENGINEERING MANAGEMENT

Solution space analysis for the bike request scheduling problem

Nicholas Vergeylen, Kenneth Sörensen & Daniel Palhazi Cuervo

DEPARTMENT OF ENGINEERING MANAGEMENT

Solution space analysis for the bike request scheduling problem

Nicholas Vergeylen, Kenneth Sörensen & Daniel Palhazi Cuervo

RESEARCH PAPER 2018-005
FEBRUARY 2018

D/2018/1169/005

Solution space analysis for the bike request scheduling problem

Nicholas Vergeylen* Kenneth Sörensen
Daniel Palhazi Cuervo

Department of Engineering management
ANT/OR — University of Antwerp Operations Research Group
University of Antwerp, Belgium

The Bike Request Scheduling Problem (BRSP) is a recently introduced combinatorial optimization problem the aim of which is to assign a number of predefined bicycle pick-or-drop instructions, called requests, to a set of vehicles and determine the sequence (routes) in which the vehicles should perform the requests. A natural and elementary operation in heuristics for the BRSP is request insertion (RI), the operation of inserting a request in a specific position in a route. Solutions of the BRSP are connected in the solution space through RI operations. This results in a distance between solutions called the RI distance. A better understanding of the solution space and the RI operator is necessary to improve both problem understanding and solution methods using RI. Therefore, we first perform a solution space cardinality analysis. Secondly, we formulate properties of the RI-metric and, finally, we visualize the solution space.

We demonstrate that the original BRSP model formulation yields a very large solution space, a problem that we solve by introducing symmetry-breaking constraints. Furthermore, we propose an expression for the cardinality of the solution space, and derive lower bounds which show that enumeration is intractable and random search is generally ineffective. Finally, we visualize the solution space of the BRSP using the RI distance, which can help understand the effects of other, more complex operators and of introducing algorithm components such as evaluation functions. A few alternatives of evaluation functions are developed and analyzed.

Keywords: Combinatorial Optimization, Combinatorial Complexity, Bike Request Scheduling Problem, Bicycle Repositioning, Multi Dimensional Scaling

*Email: nicholas.vergeylen@uantwerpen.be, kenneth.sorensen@uantwerpen.be, daniel.palhazicuervo@uantwerpen.be

1 Introduction: bicycle repositioning

Bicycle Sharing Systems (BSSs) are a recent and popular mode of public transportation (DeMaio, 2009; Midgley, 2011; OBIS, 2011; ITDP, 2014). In a BSS, users can pick up a bicycle at one of the many stations distributed across the city and leave it at another station closer to their destination. Since stations tend to become either full or empty, most BSSs operate a fleet of repositioning vehicles to move bikes between full and empty stations. This operation, called bicycle repositioning, gives rise to a planning problem in which the routes of the repositioning vehicles are determined, as well as the number of bikes to pick up and drop off at the various stations.

In Sörensen and Vergeylen (2015) a new approach is proposed to tackle the problem of bicycle repositioning by decomposing it into two distinct subproblems: (1) request generation, i.e., the generation of loading or unloading requests (essentially an order to pick up or drop off a certain number of bikes at a certain station within a certain time window), and (2) request selection and routing, i.e., the selection of requests, their assignment to vehicles, and the scheduling of requests within each vehicle route. The second problem, which is called the bike request scheduling problem (BRSP), assumes that all requests are known and aims to find a solution that minimizes the priority-weighted number of unscheduled requests, without violating vehicle capacities and request time windows. A few authors have proposed a similar decomposition approach: Caggiani and Ottomanelli (2012, 2013) and Regue and Recker (2014). A much more common approach, which combines both decisions in a single optimization problem, is followed by Benchimol et al. (2011), Contardo et al. (2012), Chemla et al. (2013), Raviv and Kolka (2013), Raviv et al. (2013), Kloimüllner et al. (2014), Pfrommer et al. (2014), Rainer-Harbach et al. (2014), Forma et al. (2015), Li et al. (2016), Zhang et al. (2016), Cruz et al. (2017), Ho and Szeto (2017) and Schuijbroek et al. (2017).

Understanding the BRSP implies understanding the sources of combinatorial explosion. The BRSP was proven to be NP hard (Sörensen and Vergeylen, 2015) and experiments suggest that small instances (i.e., in the order of 20 requests) are generally intractable for exact methods. However, it is not clear to which extent parameters such as the fleet size, the layout of the network of stations, and the request attributes, as well as their interactions, influence the size of the solution space. In this paper, we therefore perform a cardinality analysis to provide a better understanding of how the various instance parameters affect the number of possible solutions. This analysis shows that the original MILP formulation of the BRSP results in an excessively large solution space. To reduce the number of solutions, we therefore propose a series of symmetry-breaking constraints.

The cardinality analysis, corroborated by some preliminary experiments, suggests that heuristics are necessary to solve BRSP instances of practical size. In those heuristics, solutions will be manipulated by one or more heuristic operators. The most elementary operation is Request Insertion (RI), i.e., the insertion of a single unscheduled request in a specific position of a vehicle route, or the insertion of a single scheduled request in the pool of unscheduled requests. RI gives rise to a graph, linking nodes (solutions) if they can be transformed into one another by a single RI operation, i.e., by adding or removing a single request. The constructed graph implies a distance between any two solutions equal to the minimal number of RI operations required to transform one solution into the other. Understanding the RI operator and its effect on the solution space helps to understand solution algorithms that make use of this elementary component. Therefore, in this paper, we derive three useful properties of the RI operator. Moreover, we demonstrate how the RI-connected solution space can be visualized, providing additional insights.

This paper is structured as follows: a MILP formulation of the BRSP is given in Section 2. Furthermore, discussions on the solution space cardinality and model formulation improvements are presented in Section 3. The RI-connected solution space is discussed in Section 4: some properties are formulated and the RI-connected solution space is visualized. Finally, main conclusions and pointers for future research are presented in Section 5.

2 The BRSP MILP formulation

In the bike request scheduling problem (BRSP) a list of bicycle drop-off or pick-up instructions, called requests, is given. A request is an instruction to pick up or drop off a number of bikes at a certain station and within a certain time window. The characteristics of a request are shown in Table 1. A list of vehicles with identical capacity, as well as the travel time between each pair of stations, is also given.

Table 1: Attributes of a request in the bike request scheduling problem

Request attribute	Description
Station ID	Unique identifier of a station within the instance
Quantity	Number of bikes to (un)load (< 0 when unloading)
Early arrival time	Earliest allowed time to start serving a request
Late arrival time	Latest allowed time to start serving a request
Droptime	Vehicle time consumption to serve a request
Priority	Relative importance of the request, higher is better

A solution of the BRSP consists of one, possibly empty, sequence of requests for each vehicle. Requests that are not assigned to a vehicle are unscheduled. The objective function of the BRSP, which should be minimized, is the priority-weighted number of unscheduled requests. The start of the service time of a scheduled request should lie between the earliest and latest arrival time at that request. The number of bikes carried by each vehicle may never fall below zero or exceed the capacity of the vehicle. The notion of a depot is not explicitly modelled in the BRSP, as the vehicles are assumed to be able to reach the first request in their route at its earliest allowed service time, and start serving the last request in their route up to its latest service time. The initial vehicle load and the capacity to store the final vehicle load are assumed to be available in a depot. Note that the travel time matrix provides travel times between stations, rather than between requests. However, travel times between requests can easily be retrieved.

A mathematical formulation of the problem is given below. The meaning of each symbol is described in Table 2, while the decision variables are explained in Table 3.

Table 2: Parameters of the BRSP

R	set of requests
N	set of positions within a vehicle tour
K	set of vehicles
b_r	number of bikes to (un)load for request r ($< 0 \rightarrow$ delivery, $> 0 \rightarrow$ pick-up)
p_r	priority of request r
o_r	working time for request r
a_r^e	earliest service time for request r
a_r^l	latest service time for request r
t_{ij}	travel time from request i to request j
C	capacity of the vehicles

Table 3: Decision variables of the BRSP

x_{rn}^k	1 if request r is served as the n-th request of vehicle k, 0 otherwise
y_r	1 if request r is not served, 0 otherwise
a_r	service time for request r
z_{ij}	1 if request j is visited immediately after request i by the same vehicle
l_k	the number of initial bikes loaded on vehicle k

Using this notation, the problem can be formulated as follows.

$$\min \sum_r y_r p_r \tag{1}$$

s.t.

$$\sum_k \sum_n x_{rn}^k + y_r = 1 \qquad \forall r \in R \tag{2}$$

$$\sum_r x_{r(n+1)}^k \leq \sum_r x_{rn}^k \qquad \forall k \in K, n \in N \tag{3}$$

$$\sum_r x_{rn}^k \leq 1 \qquad \forall n \in N, k \in K \tag{4}$$

$$0 \leq l_k + \sum_r \sum_{n' \leq n} x_{rn'}^k b_r \leq C \qquad \forall n \in N, k \in K \tag{5}$$

$$z_{ij} \geq x^k_{jn+1} + x^k_{in} - 1 \qquad \forall i,j \in R, k \in K, n \in N \tag{6}$$
$$a_j \geq a_i + o_i + t_{ij} - (1 - z_{ij})M \qquad \forall i,j \in R \tag{7}$$
$$a^e_r \leq a_r \leq a^l_r \qquad \forall r \in R \tag{8}$$
$$a_r, a^e_r, a^l_r \in \mathbb{R}^+ \qquad \forall r \in R \tag{9}$$
$$x^k_{rn}, z_{rj}, y_r \in \{0,1\} \qquad \forall r,j \in R, k \in K, n \in N \tag{10}$$
$$l_k \in [0, C] \subset \mathbb{N} \qquad \forall k \in K \tag{11}$$

The objective, given by expression (1), is to minimize the number of unscheduled requests, weighted by their priority. Constraints (2) ensure a request is scheduled at most once. Constraints (3) and (4), respectively, make sure a sequence of vehicle requests contains no empty positions and does not contain more than one request per sequence position. Constraints (5) enforce the vehicle capacities. Note that the formulation given here is more general than the one proposed in Sörensen and Vergeylen (2015), in that the l_k variables are allowed to be larger than zero (i.e., the vehicle start with a certain number of bikes loaded). Constraints (6), (7), and (8) enforce the request time windows. M is an arbitrarily large number, a reasonable choice for which is

$$M \geq \max_{r\in R}(a^l_r) + \max_{r\in R}(o_r) + \max_{i,j\in R}(t_{ij}) + 1. \tag{12}$$

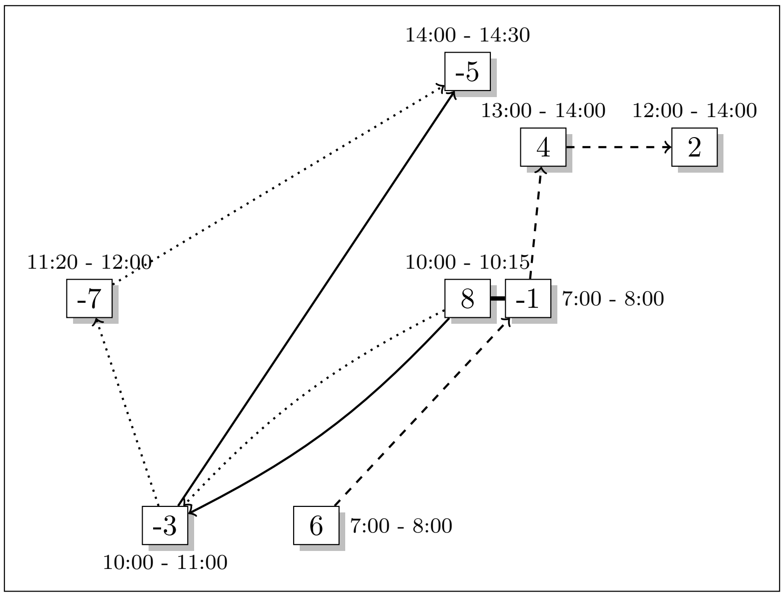


Figure 1: An instance and two 2-vehicle solutions; requests are depicted as squares, with quantities and time windows (priorities are not shown); a suboptimal solution (solid and dashed lines) and optimal solution (dotted and dashed lines) are also given

A schematical instance, including two solutions, is provided in Figure 1. Requests are represented as squares and all priorities are equal, so only quantities and time windows are relevant. There is one station that has 2 requests (with quantities of 8 and -1 bikes), all other stations have a single request. A solution for the instance, in a situation with two vehicles, is represented by black solid and dashed arrows in Figure 1, each depicting one vehicle route. No capacity violations are seen and the initial bicycle load can be 0. In this solution, the request with a quantity of -7 bikes is not executed. The priority of this request therefore determines the value of the objective function, since it represents the only term in the sum of priorities of unscheduled requests. The optimal solution, with an objective value of 0, is obtained by inserting the last unscheduled request in the route of the first vehicle. The improved situation is represented by changing the first vehicle route from the solid to the dotted line. The initial load for this vehicle should now be at least 7 bicycles.

3 Solution space cardinality analysis

Even though the BRSP is NP-hard (Sörensen and Vergeylen, 2015), it is not known how quickly the solution space grows as a function of its instance parameters. To this end, we derive a closed form expression of the solution space cardinality. Based on this analysis, we propose an improvement to the MILP model that adds symmetry-breaking constraints.

3.1 Identifying solutions

A BRSP instance $\Im$ specifies all request attributes (for all n requests), the number of vehicles (v), their capacity and the $m \times m$-matrix of travel times ($\bar{\bar{T}}$) where n requests are defined over m stations ($m \leq n$). The set $S_{\Im}$ of all possible solutions s, is called the solution space of $\Im$. Since no confusion is possible, from now on the subscript $\Im$ will be omitted.

A solution is defined by the assignment of a specific sequence, or list of requests to each vehicle. The specification of these lists also implies the set of unassigned requests. Two solutions are considered equal when the sequences of requests are equal for each vehicle. Since no distinction can be made between vehicles in the case of a homogeneous fleet, permuting the sequences does not change the solution. The MILP formulation presented in Section 2, however, does not take into account this symmetry.

3.2 Symmetry-breaking constraints

As noted by Walsh (2006), symmetry in the solution space can cause a search to be wasteful, visiting many identical solutions and leading to longer computing times. In order to break the symmetry caused by vehicle permutations, constraints (13), (14), and

Table 4: Extra decision variables of the restricted static BRSP formulation

q_k	0 if vehicle k serves as many requests as vehicle $k+1$
κ_r	ID of request r (unique within the instance context)

(15) are added to the original formulation. The new decision variables are presented in Table 4.

$$\sum_{r,j} x_{rj}^{k} \geq \sum_{r,j} x_{rj}^{k+1} \qquad \forall r \in R, j \in N, k \in K \setminus |K| \tag{13}$$

$$\sum_{r,j} x_{rj}^{k} - \sum_{r,j} x_{rj}^{k+1} \geq q_k \qquad \forall r \in R, j \in N, k \in K \setminus |K| \tag{14}$$

$$\sum_{r} x_{r0}^{k+1} \kappa_r + q_k M \geq \sum_{r} x_{r0}^{k} \kappa_r \qquad \forall r \in R, j \in N, k \in K \setminus |K| \tag{15}$$

$$q_k \in \{0,1\} \qquad \forall k \in K \setminus |K| \tag{16}$$

Constraints (13) make sure that the vehicle sequences are ordered in a non-increasing order with respect to their size (i.e., vehicle k serves at least as many requests as vehicle $k+1$). Constraints (14) make sure that q_k, a binary variable introduced in (16), is forced to zero when two subsequent vehicle sequences have equal size. If this is the case, constraints (15) allow only the solution where the first request in the first route has a smaller index (ID) than the first request of the second route. This extension of the constraint structure has no impact on the NP-hardness of the BRSP, as only single vehicle instances were used to construct the proof in Sörensen and Vergeylen (2015).

3.3 Solution space cardinality

A solution of the BRSP is determined by the order of the scheduled requests in each of the vehicles. It consists of up to v lists of up to n requests, which implies that the size of the solution space, denoted as $|S|$, is determined only by n and v. In the following, we will assume that $v \leq n$, i.e., an instance contains more requests than vehicles.

When counting solutions, the following divide and conquer strategy emerges naturally: 1) select a set of, say, k requests to be scheduled, 2) permute these k requests, and 3) partition each permuted sequence in up to v parts. This results in expression (17).

$$\sum_{k=0}^{n} C_n^k k! \sum_{j=1}^{\min(k,v)} \mathbb{P}(k,j) \tag{17}$$

Here, C_n^k is the binomial coefficient that counts the number of selections of k requests out of a set of n requests, $k!$ counts the number of permutations of this selection, $\mathbb{P}(k,j)$

is the partition function that counts the possible partitions of a sequence of k requests into j parts (or, equivalently, the number k as a sum of j terms), which is equal to the number of ways k requests can be divided in j vehicles. For more information on the partition function, the reader is referred to Andrews (1984). The min(.) function in the second sum operator ensures that no more than v vehicles are used to partition the selected number of requests.

For the same reason the original MILP formulation's solution space was excessively large, the proposed process will count some solutions multiple times. As an example, consider the sequence "$1, 2, 3, 4, 5$" of 5 request IDs. Partitioning this sequence in 3 vehicles will produce, among other partitions, the partition represented by $[1, 2][3, 4][5]$. So vehicle 1 executes request 1 and then request 2; vehicle 2 executes request 3 and then request 4; finally, vehicle 3 executes request 5. A permutation of this sequence, say $3, 4, 1, 2, 5$, will have a partition $[3, 4][1, 2][5]$, which is equal to the first in terms of the BRSP solution it represents: the first two parts, or vehicle routes, have been permuted. Now vehicle 1 executes the sequence $3, 4$ and vehicle 2 executes the sequence $1, 2$.

In order to account for these multiple counts, we need to keep track of how many partitions have equal summands (corresponding to two or more vehicles having the same number of requests in their route). When we encounter this scenario, and we partition a permuted subset of requests into the vehicles, at least one other distinct permutation can be found that has permuted the part of the first equal summand with the second one. How many distinct permutations one can find depends on how many equal summands the partition has. If the number of equal summands in the partition equals p, then exactly $p!$ permutations exist, only one of which should be counted. Therefore, the counts of equal summand solutions need to be divided by the number of equal summand permutations to obtain a correct expression for the number of solutions.

An illustration of this process is given in Table 5, and presents the process of counting all possible solutions, starting from a selection of 3 requests in 3 vehicles. Partitions are represented in a Ferrers diagram. The first row counts all permutations of 3 requests, assigned to one vehicle in order (i.e.: partition $3 = 3$). The second row counts all permutations as well, but each permuted sequence represents a solution where the first two requests are assigned -in order- to the first vehicle and the last request is assigned to the second vehicle (i.e.: partition $3 = 2+1$). The third row demonstrates the redundancy since there are three equal summands (i.e.: partition $3 = 1+1+1$). The first, second and third vehicle get the first, second and third request respectively. Other permutations of the same 3 requests will result in a solution equal to the first, but with permuted vehicles. Hence, the correction factor of $\frac{1}{3!}$ (i.e., the inverse of the number of equal summand permutations) is used to correct the total count of these equal sequences, and set it to one. Note that, in general, partitions containing different groups of equal summands will need to be corrected for each of the groups in a multiplicative fashion (e.g.: partitioning 6 requests into 4 vehicles will produce, among others, partition $6 = 2 + 2 + 1 + 1$, which will need to be corrected by the correction factor $\frac{1}{2!2!}$).

Table 5: Counting all solutions where exactly 3 requests are scheduled in up to 3 vehicles is done by counting all permutations of 3 requests, multiplying with all possible ways to split such a sequence in 3 vehicles and correcting for possible vehicle permutations

Partition	Permutation	Equal summand permutations	Correction factor	Total
$\bullet\bullet\bullet$	3!	1	1	3!
$\begin{matrix}\bullet\bullet\\ \bullet\end{matrix}$	3!	1	1	3!
$\begin{matrix}\bullet\\ \bullet\\ \bullet\end{matrix}$	3!	3!	$\frac{1}{3!}$	1
			Σ	13

The function that counts partitions and then corrects for permutations of equal summands will be denoted by $\mathbb{P}_\epsilon$. Using this notation, the solution space cardinality can be expressed as:

$$|S| = \sum_{k=0}^{n} C_n^k k! \sum_{j=1}^{\min(k,v)} \mathbb{P}_\epsilon(k,j) \tag{18}$$

3.4 Bounds and complexity

To show that naive search algorithms such as enumeration and random search cannot be applied to the BRSP, elementary theoretical lower bounds are proposed based on the expression of the search space cardinality (18). The two parameters determining the search space cardinality are the total number of requests, n, and the fleet size v. In this section, the impact of each parameter is assessed.

Note, first of all, that assessing the exact effect of $\mathbb{P}_\epsilon(k,j)$ on the search space cardinality is not straightforward, as this function has — to the best of our knowledge — no known closed-form expression. A lower and upper bound on $\mathbb{P}_\epsilon(k,j)$ is

$$\mathbb{P}(k,j) \geq \mathbb{P}_\epsilon(k,j) \geq \mathbb{Q}(k,j), \tag{19}$$

where $\mathbb{Q}(k,j)$ is the partition function that counts the number of ways to partition k in j distinct parts. Since all partitions counted by $\mathbb{Q}(k,j)$ are represented in $\mathbb{P}_\epsilon(k,j)$, which also counts permutation-corrected equal summand solutions, the second inequality follows. Because all partitions counted by $\mathbb{P}_\epsilon(k,j)$ are represented in $\mathbb{P}(k,j)$, which does not correct for permutations of equal summand terms, the first inequality follows. An upper bound for the unrestricted partition function and a lower bound for the partition function with distinct parts will therefore provide bounds for the partition function used here.

To assess the effect of increasing n, given a certain v, a loose lower bound can be obtained by substituting a constant for $\mathbb{Q}(k,j)$. In that case, the cardinality formula leads to

$$|S| > \sum_{k=0}^{n} C_n^k k! = n! \sum_{k=0}^{n} \frac{1}{(n-k)!} = n! \sum_{i=0}^{n} \frac{1}{i!} \tag{20}$$

where $i = n - k$, and the order of terms has been reversed. The sum monotonically converges from 1 to $\lim_{n\to\infty} \sum_{i=0}^{n} \frac{1}{i!} = e$. Therefore, we see that the solution space asymptotically grows in a rate at least as large as $\mathcal{O}(n!)$, which is super-exponential. The expression shows that deciding on request sequences (i.e., permuting) is a major source of combinatorial growth.

To assess the effect of increasing fleet size, given a certain n, note that, in expression (18), we can focus on the terms $\sum_{k=v}^{n} C_n^k k! \mathbb{P}_\epsilon(k,v)$, the number of additional solutions when incrementing the fleet size from $v-1$ to v. This expression shows us two sources of combinatorial growth due to the fleet size: more partitions exist, due to $\mathbb{P}_\epsilon(k,v)$, which is the first source. These extra partitions then impact each permutation of more than v requests, i.e., all factors $C_n^k k!$ where $v \le k \le n$. This is the second source of combinatorial growth and shows interaction between the fleet size and the number of requests. Retaining only the term for $k = v$ yields a conservative lower bound on growth: $C_n^v v! \mathbb{P}_\epsilon(v,v) = \frac{n!}{(n-v)!} \mathbb{P}_\epsilon(v,v)$. And, since $\mathbb{P}_\epsilon(v,v) > \mathbb{Q}(v)$, the following inequality holds

$$|S| > \frac{n!}{(n-v)!} \mathbb{Q}(v) \tag{21}$$

Expanding the factorials yields

$$\frac{n!}{(n-v)!} = n(n-1)(n-2)\cdots(n-v+1) > (n-v+1)^v \tag{22}$$

Assuming $n \gg v$, this expansion grows in a rate at least $\mathcal{O}(n^v)$. We find that the search space cardinality grows, at least, in an exponential rate with respect to v, and there is an interaction with the number of requests.

To take into account the effect of partitioning as well, note that Maróti (2003) and Knessl and Keller (1990) provide lower bounds and asymptotic expansions for various partition functions. One such asymptotic series expansion yields $\mathbb{Q}(v) \sim \frac{e^{\pi\sqrt{v/3}}}{4 \cdot 3^{1/4} v^{3/4}}$. An asymptotic formula for complexity growth of the BRSP in terms of fleet size is given by

$$\mathcal{O}(\frac{n^v e^{\pi\sqrt{v/3}}}{v^{3/4}}) \tag{23}$$

To conclude, we find that both fleet size and number of requests have a major impact on the cardinality, both as an individual effect and by interacting. These results (unsurprisingly) demonstrate the intractability of algorithms based on complete enumeration and random search. Moreover, these theoretical results confirm prior experimental results in Sörensen and Vergeylen (2015).

4 RI–connected solution space

One of the most natural and elementary operations in any (local-search based or constructive) heuristic search algorithm for the BRSP is the insertion of a single unscheduled request into a route (and thus its removal from the set of unscheduled requests), or the removal of a single scheduled request from a route (and thus its addition to the set of unscheduled requests). We call this the Request Insertion (RI) operator.

The RI operator defines a distance metric between solutions, called the RI distance. In this section we aim to provide insights into the functioning of the RI operator through the RI distance. To this end, we formulate and prove three propositions. Furthermore, a visualization method based on Multi Dimensional Scaling (MDS) is used to visualize the search space, which in turn leads to insights into alternative search strategies and components.

4.1 RI propositions

4.1.1 Notation and terminology: solution space partitioning

In the rest of this section, we will use a partitioning of the solution space in which the number of scheduled requests in a solution defines that solution's membership of a partition. Consider the indexed family of subsets $\{S_\alpha\}_{\alpha \in R}$ of S, where $R = [0, n]$. Membership of a solution s of such a subset is defined by the function $S \times R \to \{0, 1\}$: $\sigma(s, \alpha)$. This function returns 1 if the number of requests that is scheduled equals α and 0 otherwise. By construction, this is a partition on S, as $S_i \cap S_j = \emptyset, \forall i, j \in [0, n], i \neq j$ and $\cup_{\alpha=0}^{n} S_\alpha = S$. From now on, α will be called the order of the partition.

In the previous section, the cardinality of the BRSP solution space was expressed as a sum of terms (see expression (18)). These terms were defined as the number of solutions that have k requests planned. Note that this definition coincides with the partition defined here, hence the following holds

$$|S| = \sum_{k=0}^{n} C_n^k k! \sum_{j=1}^{\max(k,v)} \mathbb{P}_\epsilon(k, j) = \sum_{k=0}^{n} |S_k| \tag{24}$$

Every term in expression (24) counts the number of solutions that have k requests scheduled, i.e. the number of solutions for which $\sigma(s, k) = 1$. Therefore, termwise identification yields the cardinalities of the partition subsets:

$$|S_k| = C_n^k k! \sum_{j=1}^{\max(k,v)} \mathbb{P}_\epsilon(k, j) \tag{25}$$

4.1.2 Notation and terminology: RI distance metric

A mapping is defined on the solution space S, based on the RI operator, which implies a distance measure between any two solutions. More formally, consider the mapping $m : S \to 2^S$, where, for each $s_k \in S$, m returns the set of solutions that can be derived by a single RI operation, i.e., by either removing a request or adding a request. This set is denoted $N_{RI}(s_k)$, and is called the RI-neighborhood of s_k. By construction, the relation "is neighbor" is symmetric. Neighbors are said to be at distance 1 from one another. We say that a path between two solutions exists iff a sequence of solutions exists, starting with the first and ending with the second solution, such that each subsequent pair of solutions in the sequence are neighbors. The length of a path containing n solutions is $n - 1$. The distance between any two given solutions s_a and s_b is the minimum path length between them, and is represented by $d_{RI}(s_a, s_b)$. In other words, the RI distance between a pair of solutions is the minimum number of RI operations required to transform the first solution into the second (or vice versa).

4.1.3 Proposition 1: expanding partition size

The first proposition captures the observation that the larger the order of the partition (i.e., the larger the number requests scheduled in the solution), the larger the size of the partition. The only exception is when the instance contains a single vehicle and when there is only one request left unscheduled. In this case, the number of solutions obtained by allowing all requests to be scheduled is equal to the number of solutions where a single request is left unscheduled.

Formally, the proposition can be written as:

$$(\forall \alpha, \beta)(\alpha \in [0, N-1] \wedge (\beta \in [0, N-1] \vee (\beta = n \wedge v > 1)) \Rightarrow (\alpha < \beta \Leftrightarrow |S_\alpha| < |S_\beta|)) \tag{26}$$

$$|S_{N-1}| = |S_N| \Leftrightarrow v = 1 \tag{27}$$

proof. More formally, note that for every solution in S_α we can generate a solution in $S_{\alpha+1}$ by appending the unscheduled request with the lowest ID to the first vehicle sequence. This process implies an injective relation as every solution in S_α has a corresponding solution in $S_{\alpha+1}$; but not a bijective relation, since using any other unscheduled request will render a solution in $S_{\alpha+1}$ that is by construction not part of this relation. Therefore, $|S_\alpha| > |S_{\alpha-1}|$ if $0 < \alpha < n$. For $\alpha = n$ and $v = 1$, by construction, this relation does become bijective as the list of unscheduled requests is exhausted up to the final request. So there are no alternative solutions that can be constructed by appending another request. When $v > 1$, however, this is not the case. There are always solutions containing sequences of equal size (e.g. the partition $w + 1 + 1$, with $w \geq 0$, for any partition of $p > 1$). Due to the symmetry-breaking constraints, one can create a solution not part of the relation by appending the unscheduled request to the last of the equal sized vehicles and permuting it such that it respects the symmetry-breaking constraints again. In the case of $p = 1$, solutions spread over 2 vehicles are not part of the relation.

Similarly, when $|S_\alpha| = |S_\beta|, \alpha < \beta$, a bijection on $S_\alpha \times S_\beta$ is possible, which excludes the possibility of constructing an injection that is not a bijection. The only case where this could hold is the case $\alpha = n - 1, \beta = n, v = 1$, since for any other case a non-bijective injection has been constructed. For this case, a bijection was constructed. Hence the proposition has been proven by construction.

4.1.4 Proposition 2: Transcendental infeasibility

Proposition 2 states that an infeasible solution cannot be rendered feasible by appending requests to (the end of) a vehicle route. An important consequence is that all feasible solutions are connected to the empty solution and, therefore, to each other.

Consider the set of feasible solutions $\mathscr{F} \subseteq S$. We can define the indexed family of subsets $\{\mathscr{F}_\alpha\}_{\alpha\in[0,n]}$ of S via $\mathscr{F}_\alpha = S_\alpha \cap \mathscr{F}, \forall\alpha$. Using this formalism, the second proposition can be written as:

$$(\forall\alpha)(\alpha \in [1, n] \Rightarrow (\mathscr{F} \cap S_{\alpha-1} = \emptyset \Rightarrow \mathscr{F} \cap S_\alpha = \emptyset)) \tag{28}$$

proof. To see this, note that any feasible solution in $S_\alpha, \alpha \in [1, n]$, has neither request time window violations nor capacity violations. In this case, we can always unschedule the last request of any non-empty vehicle, without introducing any time or capacity violations. In performing this operation, one obtains a solution in $S_{\alpha-1}$. This is formulated by the set of implications

$$\mathscr{F} \cap S_\alpha \neq \emptyset \Rightarrow \mathscr{F} \cap S_{\alpha-1} \neq \emptyset, \forall\alpha \in [1, n] \tag{29}$$

which is equivalent to the set of implications in (28).

4.1.5 Proposition 3: Nearness of feasible solutions

Proposition 3 captures the observation that, when the number of requests is incremented by one (meaning the number of solutions is at least doubled), the maximum distance between any pair of solutions increases at most by two.

More formally, the proposition can be written as:

$$(\forall s_a, s_b)((s_a \in S_\alpha \wedge s_b \in S_\beta) \Rightarrow d_{RI}(s_a, s_b) \leq \alpha + \beta) \tag{30}$$

To see this, consider a trivial case, valid for all solutions s_a:

$$(\forall s_a, \alpha)((s_a \in S_\alpha \wedge \alpha \in [0, n]) \Rightarrow d_{RI}(s_0, s_a) \leq \alpha) \tag{31}$$

proof. Using the constructional proof for Proposition 2, one can construct the path between s_0 and s_k by removing iteratively the last request of any vehicle sequence α times, starting from s_k. It is trivial to see that there can be no shorter paths to s_0 using only RI-operations, as there are at least α moves required to construct the solution from s_0.

From inequality (31), expression (30) immediately follows by using the triangular inequality: Clearly, inequality (31) can be used to yield the triangular inequality between any two solutions s_a, s_b. It is sufficient to note that one can remove all requests from solution s_a and insert all requests in solution s_b. This yields a distance

$$d_{RI}(s_a, s_b) \leq d_{RI}(s_a, s_0) + d_{RI}(s_0, s_b) = \alpha + \beta. \tag{32}$$

Which proves expression (30).

We know from the cardinality analysis that the number of possible solutions increases at a super-exponential rate. Combined with Proposition 3, this implies that the solution space will be concentrated in the highest-order partitions. In other words, when considering solutions in the higher-order partitions where many requests are scheduled in vehicle routes, the number of alternative solutions becomes unmanageable, even for small distances. However, since the distances remain small, and taking into account Proposition 2, the feasible solutions in these higher-order partitions are close to feasible solutions in the partitions with lower orders. In the worst case, the closest feasible solution is the empty solution. We therefore obtain a way to deal with such a large solution space: from a few solutions with a low number of requests scheduled, we obtain feasible solutions with many requests scheduled while maintaining feasibility by appending requests.

4.2 Visualization

4.2.1 Dimensionality reduction

Using the RI-distance measure introduced in the previous section, we visualize the solution space to gain more insights into the best way to solve the BRSP heuristically. In doing so, we follow the example of Rasku et al. (2013), who suggest to use solution space visualization as a tool for the development of routing algorithms.

The solutions of a given instance, along with their pairwise RI-distances, yield a space with multiple dimensions. Therefore, any sort of visualization (in two or three dimensions) must use some type of dimensionality reduction. An introduction to this topic can be found in Sharma (1996). We make use of Multi Dimensional Scaling (MDS) in order to create a two-dimensional representation of the original solution space as accurately as possible. The reduction performed by this technique is explained in detail by Buja and Swayne (2002) and Wickelmaier (2003).

Since large instances cannot be enumerated exhaustively, we limit our efforts to small instances. In this section, we illustrate how instances with 5 requests and 3 vehicles can be used to visualize the solution space. This solution space is labeled as (5,3). The number of solutions for this instance can be computed using expression (18), and it is equal to 1006.

The solution space is generated exhaustively by using the RI operator, starting from the empty solution (all requests unscheduled) and recursively scheduling every possible request. From this structure, the distance matrix is calculated using the Floyd-Warshall algorithm (FWA), which computes all-pairs shortest paths in $\mathcal{O}(n^3)$ operations (see, e.g., Cormen et al., 2001, for more information about this algorithm). The resulting distance matrix is instance independent, since the solution space and the RI distance metric are instance independent. Of course, the feasibility of the different solutions in the solution space does depend on the instance.

The distance matrix is the input for the MDS analysis. An iterative procedure tries to find coordinates in a two-dimensional space in such a way that a stress criterion is minimized. Both, an ordinal or non-metric MDS, and an absolute or metric MDS, were performed using SAS. In the case of the ordinal MDS, only the order of the elements in the distance matrix are respected, not their relative sizes. For the absolute MDS, the relative sizes are respected. The quality of the analysis is represented by the badness-of-fit (BOF) criterion, which is displayed in Table 6.

Table 6: MDS results

MDS type	BOF	Iterations to convergence
Metric	0.3612	11
Ordinal	0.3588	19

The MDS analysis provides results for which both BOF criterion values exceed a (generally accepted) threshold of 0.20 (Wickelmaier (2003)). This means that the results of the visualization should be treated with caution. More specifically, the relatively large values mean that neighboring solutions can sometimes be located farther from or closer to each other in the plane, than in the original solution space. Still, the visualization does help to provide useful insights. Since the partitions of the solution space (along with the corresponding neighborhood structure) are always marked on the figures, the output of the MDS analysis is used as a starting point for the visualization, rather than as an end result. In what follows, the ordinal MDS was used for further analysis because of the (slightly) smaller BOF criterion value.

Figure 2 shows the visualization of the solution space (5,3). It can be seen that, from the empty initial solution, s_0, the partitions are organized in outward layers, with the outer layers containing solutions with more requests scheduled. The relatively large BOF can be perceived by noting that a part of S_2 is closer to s_0 than all the solutions in S_1. Moreover, the solutions in S_1 appear very close to each other, almost entirely overlapped. Considering that the MDS technique aims to minimize the global deviation from the original RI-distance matrix, these inconsistencies can be explained by Proposition 1 of the search space: the impact of the first three partitions on the overall BOF is limited due to their relatively small cardinality.

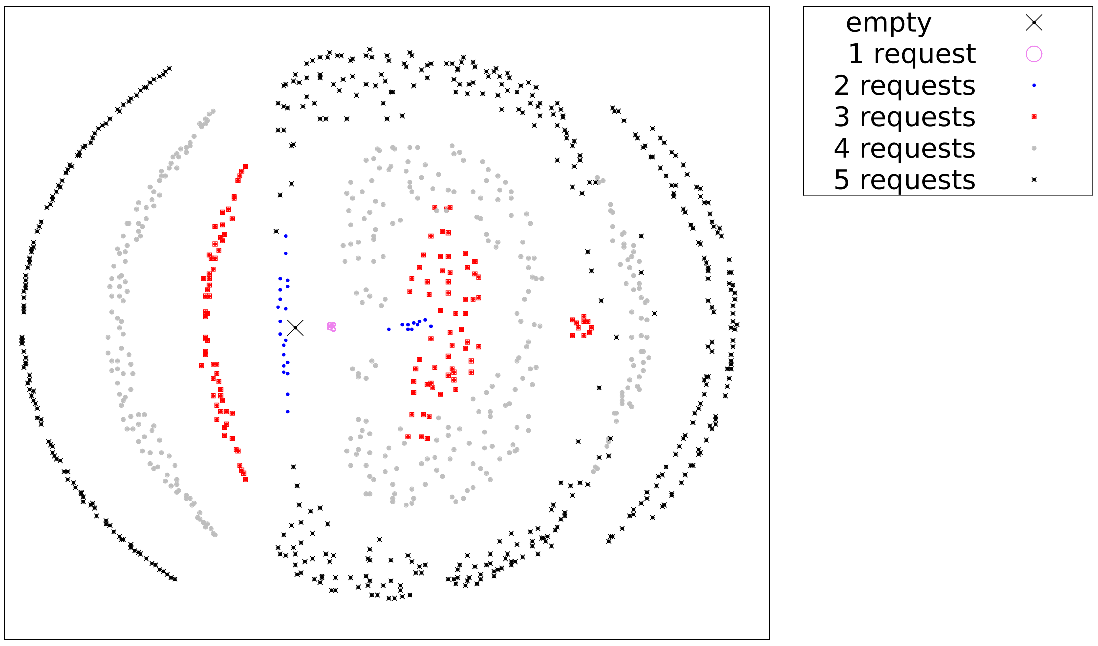


Figure 2: The solutions in the RI-connected solution space are organized in layers around the empty solution

4.2.2 Visualizing search strategies and evaluation functions

Local search algorithms require an evaluation function that, given an incumbent solution, defines which of the solution's neighbors are the most promising ones to explore. In other words, the evaluation function helps to select the neighboring solution that will become the next incumbent solution, in the following iteration of the algorithm (Michalewicz and Fogel, 2000). For many optimization problems, the objective function is a natural choice for the evaluation function. In this case, the execution of the local search is automatically directed towards the exploration of good solutions (with good values of the objective function). For the BRSP, however, the objective function may not be the best evaluation function for a local search algorithm. The main reason for this is that it does not provide a direction of optimization when starting from an infeasible solution that only has infeasible neighbors. The objective function only provides information about how valuable certain (un)scheduled requests are. Another reason is that, any solutions with the same requests scheduled, has the same objective function value, regardless of the way in which these requests are ordered in the routes. Two solutions might therefore have the same objective function value, even though the routes of the first solution are considerably more efficient than those of the second. In the remainder of this section, we focus on the lack of guidance in the objective function to move towards the feasible region of the search space.

Two possible search strategies appear naturally: 1) restricting the search to the feasible area $\mathscr{F}$ or 2) extending the objective function in a meaningful way such that solutions in the infeasible area $\bar{\mathscr{F}}$ can be evaluated.

To illustrate both approaches, two different instances in (5,3) are presented in Table 7. The first instance is very restricted in that it has very tight time windows. To the contrary, time windows in the second instance are more relaxed. It is important to notice that, for each instance, vehicle capacities are set to 30 units.

Table 7: Request properties of a restricted instance [left] and of a more relaxed instance [right], as well as the travel time matrix [bottom]

EAT	LAT	DT	Q	P	ID	SID	EAT	LAT	DT	Q	P	ID	SID
500	700	220	10	5	0	0	0	2000	220	10	5	0	0
500	700	270	15	3	1	1	1500	3600	270	15	3	1	1
500	700	170	-10	5	2	2	300	900	170	-10	5	2	2
500	700	320	-10	3	3	3	2300	3200	320	-10	3	3	3
500	700	270	-15	4	4	4	0	3600	270	-15	4	4	4

	1	2	3	4	5
1	0	1201.3	1489.46	2099.87	391.24
2	1201.3	0	1439.96	2632.09	1382.99
3	1489.46	1439.96	0	1320.16	1249.9
4	2099.87	2632.09	1320.16	0	1712.6
5	391.24	1382.99	1249.9	1712.6	0

Restricting the search to $\mathscr{F}$ The reason why the objective function is ineffective in $\bar{\mathscr{F}}$ is that it does not penalize constraint violations. For this reason, a natural strategy is to restrict the search to the feasible region $\mathscr{F}$. This approach requires that the initial solution is feasible as well. The empty solution s_0 can be chosen for this purpose, as well as all solutions in S_1 (because the initial load of the vehicle can be freely chosen).

Visualizing the instances and the feasible solution paths shows that all feasible paths contain the empty solution (Figure 3a and 4a). This is, of course, always the case due to Proposition 2.

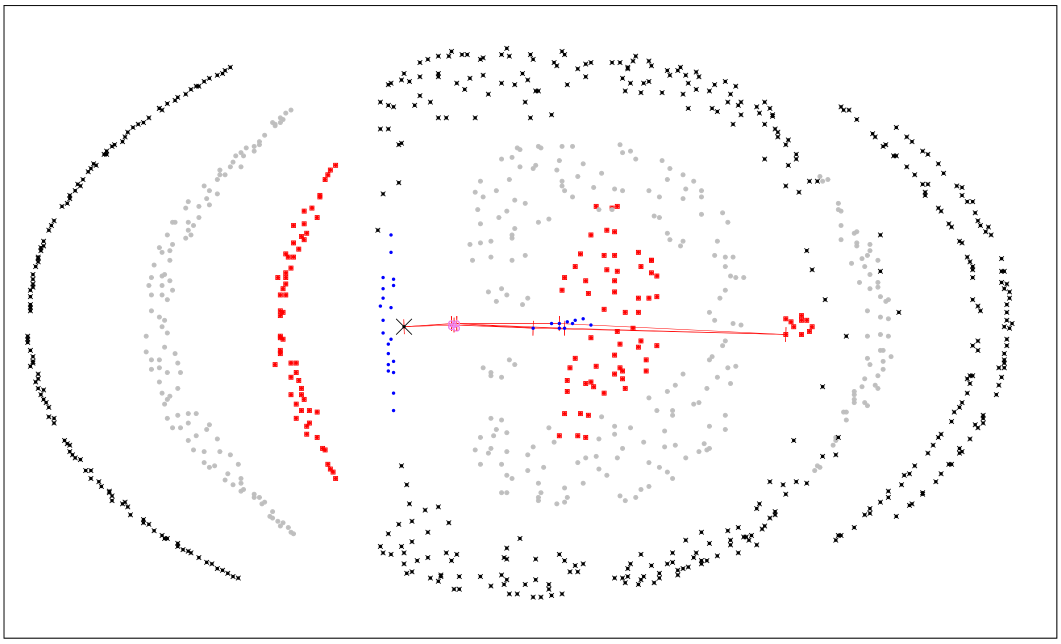
A search that starts from the empty solution and exhaustively searches the entire feasible space is visualized for the instances in Figures 3b and 4b. From these figures, it is evident that only a small subset of the solution space is feasible and needs to be evaluated in the restricted instance. The search can therefore be considered efficient. For the relaxed instance, the efficiency of the search is obviously less, but still significant.

More importantly, the gain of restricting the search to the feasible region will usually be the highest in the higher order partitions, where most solutions of the instance reside. The reason for this is that scheduling more requests means increasing the risk of violating constraints (see Propositions 2 and 3). The gain of restricting the search to the feasible region is therefore twofold. First, the need to evaluate infeasible solutions is limited, and hence also the waste of time. Secondly, the objective function can now be used as an evaluation function, since it can rank all solutions under consideration, provided that an infeasible solution will never be an incumbent solution and is therefore always outranked by a feasible solution.

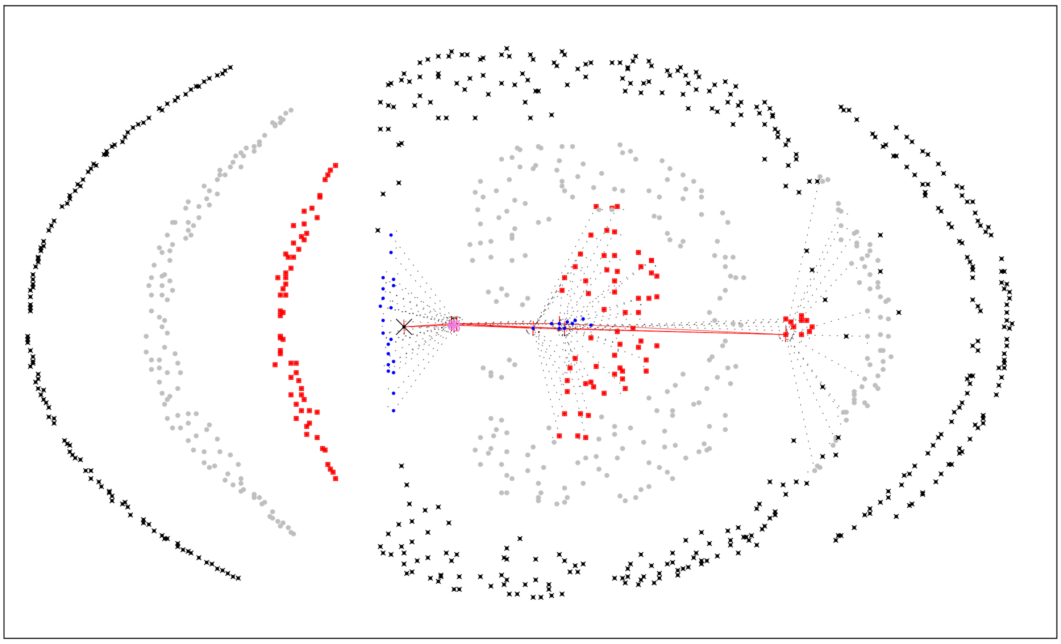
To harness these ideas and use them to increase the efficiency of a heuristic search algorithm, one can always start from the empty solution s_0 and move iteratively from S_k to S_{k+1}, pruning the branches where no feasible solutions can ever reside. This is the branch and bound principle. The observations also suggest that dynamic programming and constraint programming are viable solution methods for the BRSP. A possible strategy, for example, can be to start with a request and prepend other requests, ignoring requests that have an EAT later than the first request's LAT. In other words, the search keeps track of the feasible solutions. By taking into account that the not-yet-considered requests can be scheduled without violating constraints, the hypothetical objective allows to compute a lower bound on the solution, and so provides a way to further prune suboptimal branches.

In conclusion, we find that intelligent use of the constraint structure and iteratively moving through the feasible part of the partition structure exhibits potential in designing both heuristics and exact methods, a fortiori for more constrained cases.

Extending the objective function to $\bar{\mathscr{F}}$ Another approach to improve search efficiency is to remedy the lack of guidance the objective function provides in $\bar{\mathscr{F}}$, i.e., in the space

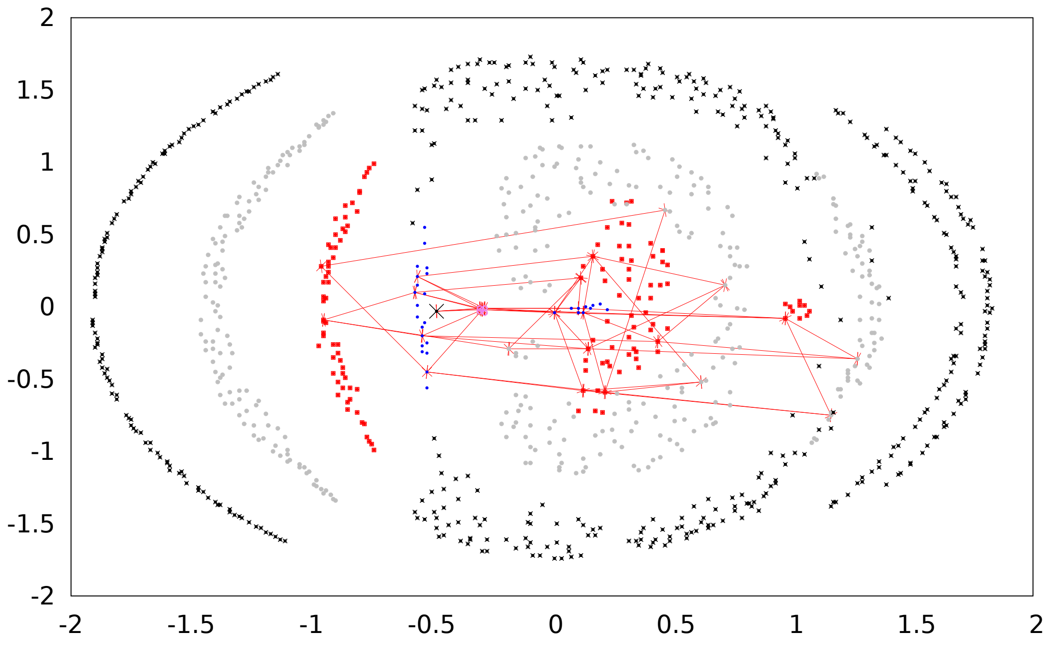


(a) Feasible solutions are linked in "paths"

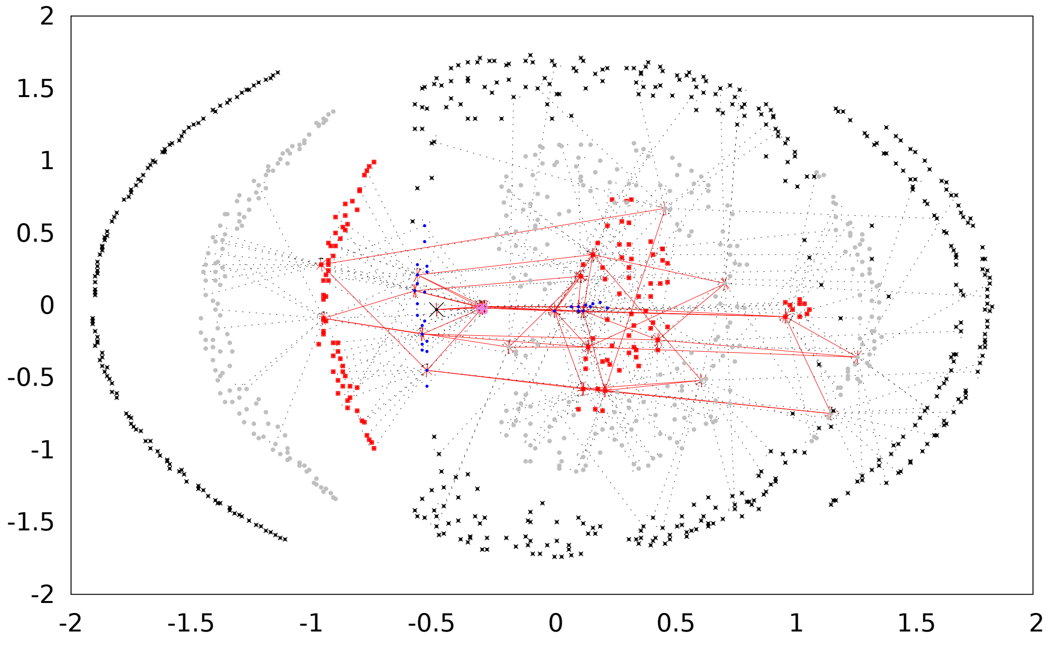


(b) Feasible solutions and their neighbors

Figure 3: A version of the solution space (Figure 2) for a restricted instance, where feasible solutions and their neighbors are connected, showing that feasible solutions are more common in the inner layers than in the outer layers



(a) Feasible solutions are linked in “paths”



(b) Feasible solutions and their neighbors

Figure 4: A version of the solution space (Figure 2) for a more relaxed instance, where feasible solutions and their neighbors are connected, showing that feasible solutions are still concentrated in the inner layers, though somewhat less pronounced as was the case in the restricted instance

of infeasible solutions. Developing an evaluation function that allows the search to continue even though the incumbent solution is infeasible, is frequently done by penalizing constraint violations.

Consider a first extension of the objective function where capacity constraints are penalized. This can be done in many ways, but one possibility is presented below:

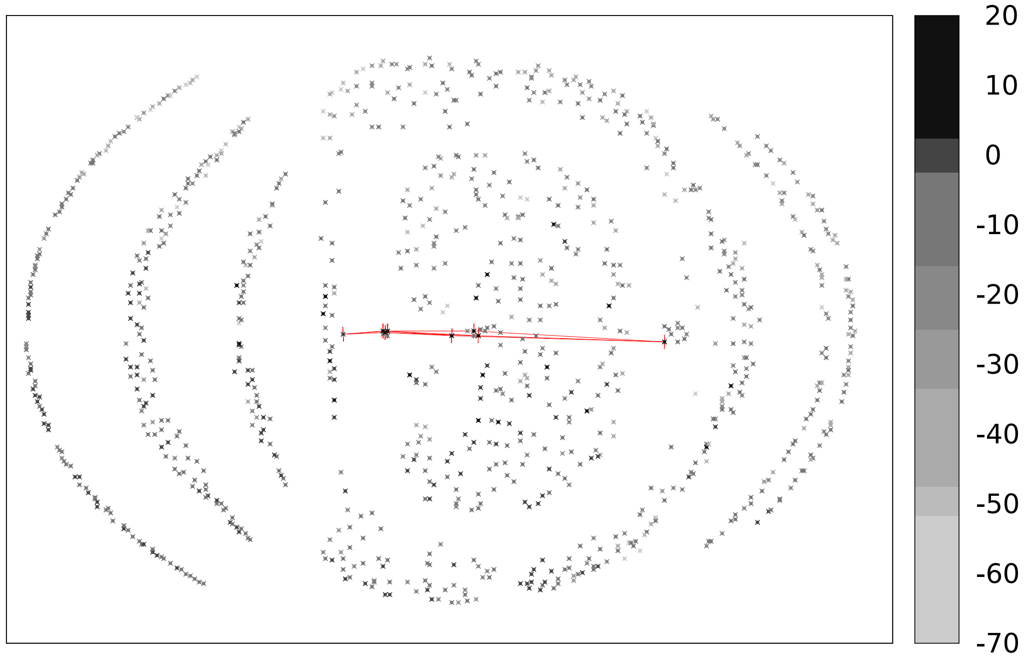
$$eval(s_a, s_b) = \begin{cases} \mathscr{O}(s_a) - \mathscr{O}(s_b), & s_a, s_b \in \mathscr{F} \cup (s_a, s_b \in \bar{\mathscr{F}} \cap (\mathscr{V}^c(s_b) = \mathscr{V}^c(s_a))) \\ \mathscr{O}(s_a), & s_a \in \mathscr{F} \cap s_b \in \bar{\mathscr{F}} \\ -\mathscr{O}(s_b), & s_a \in \bar{\mathscr{F}} \cap s_b \in \mathscr{F} \\ \mathscr{V}^c(s_b) - \mathscr{V}^c(s_a), & s_a, s_b \in \bar{\mathscr{F}} \cap (\mathscr{V}^c(s_b) \neq \mathscr{V}^c(s_a)) \end{cases}$$

Here $\mathscr{O}(s)$ stands for the objective value of solution s. $\mathscr{V}^c$ represents the total capacity violation encountered in the solution. The capacity violation of a solution is calculated as the sum of total excess (executing a scheduled request results in more bikes being carried than the vehicle capacity) and (the absolute value of) negative capacity (executing a scheduled request results in the number of bikes carried to fall below zero) in the vehicle routes. The function ranks solution s_a better than s_b when its objective value is higher and both solutions are either feasible or have an equal capacity violation. If one of both solutions is infeasible and the other is not, the feasible solution outranks the infeasible one. When they are both infeasible but have a different capacity violation, the solution with the lowest violation is ranked best.

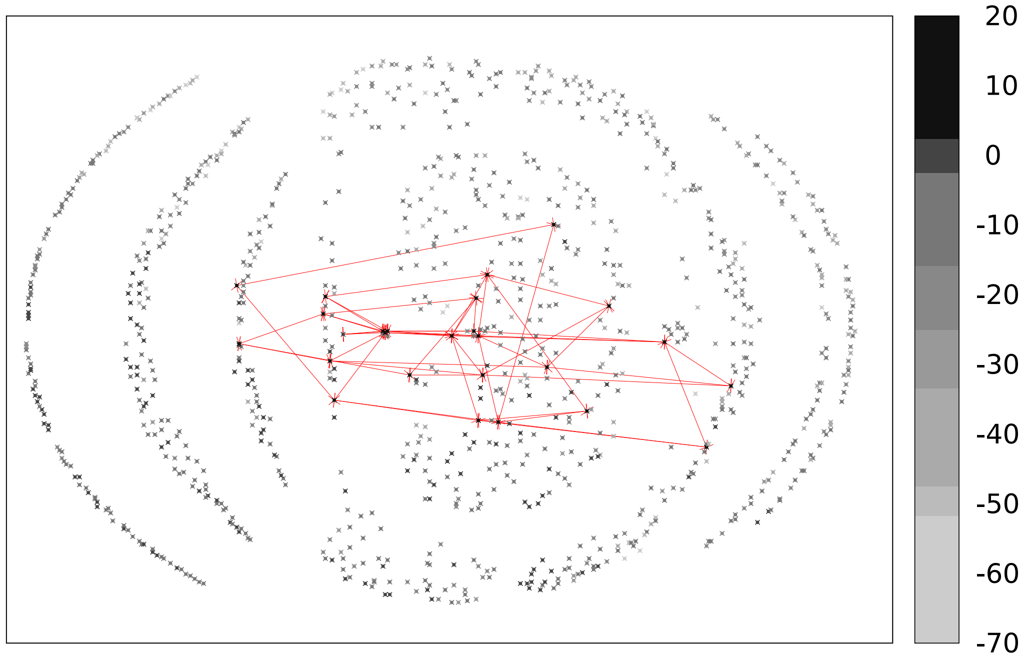
Ranking infeasible solutions in order of capacity violation steers the search towards the feasible region, or, when no discrimination is possible with respect to capacity violations, towards solutions with better quality. Infeasible solutions will always be ranked lower than feasible ones. In this way, the objective function's effective region is extended to the entire solution space. This ranking strategy is visualized by using the evaluation function value as a grayscale value (Figure 5).

For the restricted instance, the evaluation function does not seem to be highly effective in guiding any search towards the feasible/optimal solution(s), as many local optima are located in the leftmost (infeasible) part of the search space (Figure 5a). The results seem slightly better for the relaxed instance, as there are less dark solutions in the infeasible areas (Figure 5b). The local optima in the evaluation landscape for the restricted instance can be explained by the fact that time constraints are very important compared to capacity constraints, but this is not reflected in the evaluation function. In other words, the evaluation function does not understand the major source of infeasibility for this instance, i.e., the time window violations.

A second, improved evaluation function copes with this problem. The function extends the previous evaluation function by also penalizing time constraint violations. Again, many alternatives are possible for the penalty function, and additionally, a decision needs to be made on the trade-off between capacity and time constraint violation. A first possibility is to use priority rules, e.g., first remove time constraints, then capacity constraints. However, unless one has a priori information on which constraints are more



(a) The restricted instance



(b) The more relaxed instance

Figure 5: Evaluating solutions by penalizing capacity constraint violations show that the feasible region is at the top of the evaluation surface, but infeasible local optima can still occur

important, a second possibility, i.e., a linearization of both constraints, seems a more logical choice to combine both types of violations. In that case, the weights need to be chosen. A third way of comparing both types of constraint violations is to use sums of piecewise functions to assess violations. Because of the non-linear nature of these functions, they can combine both linearization and priority rules. This last method will be illustrated.

As an example, a piecewise function approach is proposed here. Keep in mind that capacity violations are computed as before. The time window violation for a single request is computed by taking the difference between the vehicle arrival time and the LAT, $a_r - a_r^l$. Only positive values should be counted as violations. Because there is no obvious way to compare a violation of time windows (measured in time units) with a violation of capacity constraints (measured in bicycles), the penalty function for time window violations is chosen such that common violations of time windows are evaluated on the same scale as common violations of capacity, and just so, large violations should be penalized on roughly the same scale. Calibrating these penalty functions in this way allows them to be summed in order to form a single violation penalty. Moreover, we will use an exponential curve to penalize time window violations such that the penalty is less sensitive for time window violations when they are low than when they are high. In that case, they should become a priority to solve. To fit this exponential curve, the two orders of violation, i.e. common violations and large violations, are defined for both capacity and time window violations, and the penalties are set to the same order of magnitude. The fit will be approximate, as the example is for illustration purposes only, and elaborate tuning can and should be done in practice for a set of relevant instances.

An estimate for large capacity constraints can be obtained by summing the absolute values of the quantity attributes, which amounts to a penalty of 65 for both instances (13 per request). An estimate for lower capacity violations can be obtained by taking the absolute value of the sum of the quantity attributes, which amounts to 10 (2 per request) for both instances. We propose that an average violation of 6 minutes is considered a common violation. A violation of 25 minutes is considered large. Using the following function respects the discussed order of magnitudes and follows the exponential curve:

$$\mathscr{T}_i(s) = \begin{cases} \mathrm{e}^{\frac{a_r - a_r^l}{600}}, & a_r - a_r^l < 2400, \forall r \in R \\ \mathrm{e}^4, & a_r - a_r^l \geq 2400, \forall r \in R \\ 0, & a_r \leq a_r^l, \forall r \in R \end{cases}$$

Where time units are measured in seconds. A time window violation of 6 minutes will produce a penalty of 1.82, which is close to the penalty of 2 per request given by the capacity violation function. A time window violation of 25 minutes produces a penalty of 12.18, which is close to the penalty of 13 per request. The function is constant after 40 minutes, since then the penalty is $e^4 \approx 55$, which is close to the maximum quantity penalty conceivable. Doing this will limit the sensitivity of a single violation, so that not only the time violation but also the amount of requests in time window violation is still considered important.

Combination of capacity violation and time window violations penalty functions yields:

$$\mathscr{T}(s) = \sum_{\forall i} \mathscr{T}_i(s) \tag{33}$$

$$\mathscr{V}(s) = \mathscr{T}(s) + \mathscr{V}^c(s) \tag{34}$$

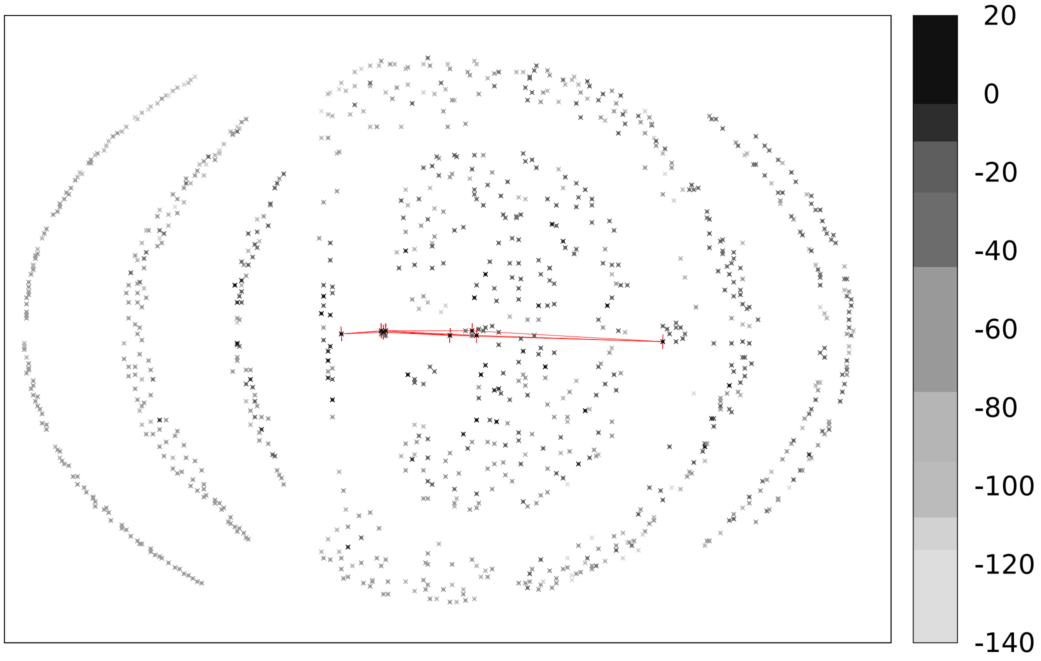
Figure 6 shows the effect of using $\mathscr{V}(s)$ instead of $\mathscr{V}^c(s)$ in the evaluation function. This function shows a smoother optimization landscape. A search relying on this evaluation function is guided more effectively towards the feasible solutions. The further the search departures from the feasible area (in terms of both capacity violations or time windows violations, or any combination thereof), the more negative the evaluation function becomes.

5 Conclusion

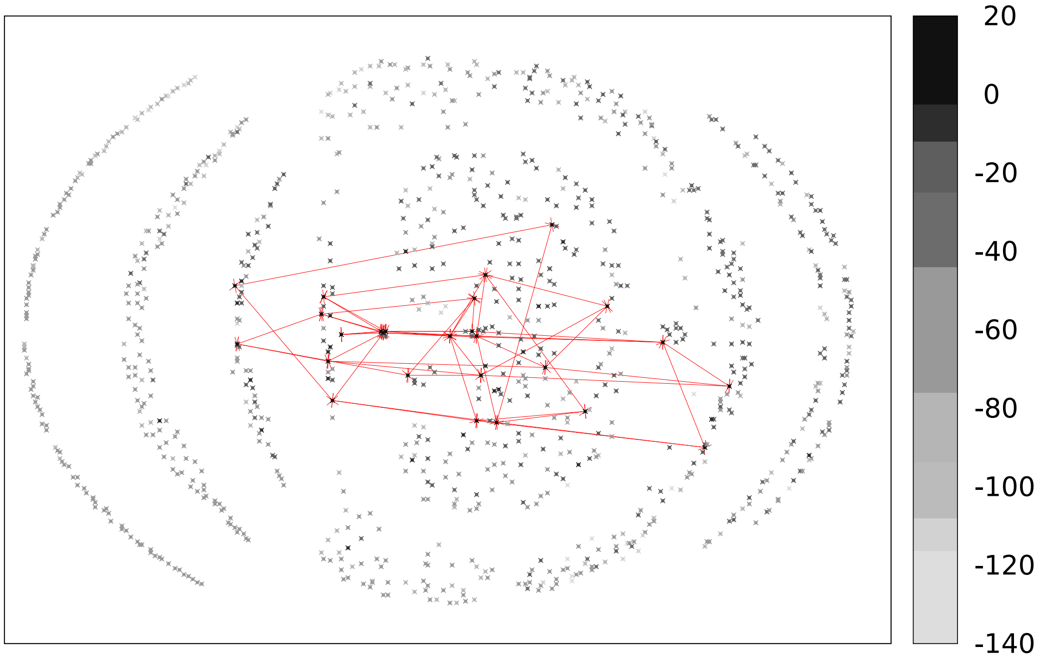
In this paper we have studied the BRSP solution space by considering its cardinality and a few interesting properties of the request insertion (RI) operator as a distance metric.

We developed an expression to compute the cardinality of the solution space. Furthermore, based on this expression, we derived lower bounds on the growth of the solution space. It turns out that the solution space grows at least as fast as $\mathcal{O}(n!)$, i.e., super-exponential, with increasing number of requests. With increasing fleet size, growth is bounded from below by $\mathcal{O}(n^v)$, which is exponential and the more requests are available, the more steep growth will be. Clearly, the problem quickly becomes untractable, which theoretically confirms experimental results reported in Sörensen and Vergeylen (2015). The cardinality implied by the original BRSP MILP model formulation was observed to be too large, and formulation improvements have been made by adding symmetry-breaking constraints, which can help to speed up exact solvers.

With respect to the RI distance metric, we formulated and proved three propositions that can be used by heuristic designers to construct solution algorithms. Furthermore, we used MDS to visualize the search space and, in combination with the propositions, managed to find alternative search strategies that boost efficiency. Starting from the empty solution and using a specific pruning rule, a branch and bound avoids evaluating large parts of the solution space. The more constrained the instance, the better this strategy will work. Here the objective function is the evaluation function. A second approach is to use an evaluation function that penalizes capacity and time constraint violations. In contrast to the plain objective function, it is usable in the infeasible zone of the solution space. For relaxed instances, this approach becomes more attractive than the fist alternative, since it can be combined with different search strategies. Finally, because RI can serve as a benchmark to assess the impact of more complex operators, showing solution connections due to these complex operators within the visualized RI-connected solution space can help to improve understanding of those operators.



(a) The restricted instance



(b) The more relaxed instance

Figure 6: Evaluating solutions by penalizing both capacity and time window constraint violations shows that the evaluation surface is more steep than only penalizing capacity constraints, and a search is guided more effectively

In summary, we improved the understanding of the BRSP by studying its solution space and by studying the properties of RI, the most elementary operator to improve or worsen solutions. And, in addition to that, we formulated some alternative search strategies and possible evaluation functions that can be used in the future to solve the BRSP.

The results of this study can be improved by conceiving new search strategies or new evaluation functions, such that the number of required evaluations is reduced and/or the optimization surface is observed to be more smooth. A second avenue for future research is to compare other heuristic operators to the RI. They should be able to demonstrate their merit, both from a theoretical and a visualization point of view. Future research can also use the knowledge obtained here to develop an effective heuristic for the BRSP. A final avenue for future research is to extend the methodology followed in this study to other problems, which might shed some light on the way different optimization operators work for different problems.

References

G.E. Andrews. The Theory of Partitions. Cambridge University Press, 1984. Cambridge Books Online.

M. Benchimol, P. Benchimol, B. Chappert, A.D.L. Taille, F. Laroche, F. Meunier, and L. Robinet. Balancing the stations of a self service bike hire system. RAIRO - Operations Research, 45(1):37–61, 2011.

A. Buja and D.F. Swayne. Visualization methodology for multidimensional scaling. Journal of Classification, 19(1):7–43, 2002.

L. Caggiani and M. Ottomanelli. A modular soft computing based method for vehicles repositioning in bike-sharing systems. Procedia - Social and Behavioral Sciences, 54: 675 – 684, 2012.

L. Caggiani and M. Ottomanelli. A dynamic simulation based model for optimal fleet repositioning in bike-sharing systems. Procedia - Social and Behavioral Sciences, 87: 203 – 210, 2013.

D. Chemla, F. Meunier, and R. Wolfler Calvo. Bike sharing systems: Solving the static rebalancing problem. Discrete Optimization, 10(2):120 – 146, 2013.

C. Contardo, C. Morency, and L.-M. Rousseau. Balancing a dynamic public bike-sharing system. Centre Interuniversitaire de Recherche sur les Réseaux d'entreprise, la Logistique et le Transport, 2012.

T.H. Cormen, C. Stein, R.L. Rivest, and C.E. Leiserson. Introduction to Algorithms. McGraw-Hill Higher Education, 2nd edition, 2001.

F. Cruz, A. Subramanian, B.P. Bruck, and M. Iori. A heuristic algorithm for a single vehicle static bike sharing rebalancing problem. Computers & Operations Research, 79:19 – 33, 2017.

P. DeMaio. Bike-sharing: History, impacts, models of provision, and future. Journal of Public Transportation, 12(4):41–56, 2009.

I. A. Forma, T. Raviv, and M. Tzur. A 3-step math heuristic for the static repositioning problem in bike-sharing systems. Transportation Research Part B: Methodological, 71:230 – 247, 2015.

S.C. Ho and W.Y. Szeto. A hybrid large neighborhood search for the static multi-vehicle bike-repositioning problem. Transportation Research Part B: Methodological, 95:340 – 363, 2017.

ITDP. The bike-share planning guide. Report, Institute for Transportation & Development Policy, 2014.

C. Kloimüllner, P. Papazek, B. Hu, and G.R. Raidl. Balancing bicycle sharing systems: An approach for the dynamic case. In C. Blum and G. Ochoa, editors, Evolutionary Computation in Combinatorial Optimization, volume 8600 of Lecture Notes in Computer Science, pages 73–84, Berlin Heidelberg, 2014. Springer.

C. Knessl and J. B. Keller. Partition asymptotics from recursion equations. SIAM J. Appl. Math., 50:323 – 338, 1990.

Y. Li, W.Y. Szeto, J. Long, and C.S. Shui. A multiple type bike repositioning problem. Transportation Research Part B: Methodological, 90:263 – 278, 2016.

A. Maróti. On elementary lower bounds for the partition function. Integers: Electronic Journal of Combinatorial Number Theory, 3(A10):2, 2003.

Z. Michalewicz and D.B. Fogel. How to Solve It: Modern Heuristics. Springer-Verlag New York, Inc., New York, NY, USA, 2000.

P. Midgley. Bicycle sharing schemes: enhancing sustainable mobility in urban areas. Background Paper CSD19/2011/BP 8, United Nations Department of Economic and Social Affairs, Commission on Sustainable Development, 2011.

OBIS. Optimizing bike sharing in european cities: A handbook. Technical report, 2011.

J. Pfrommer, J. Warrington, G. Schildbach, and M. Morari. Dynamic vehicle redistribution and online price incentives in shared mobility systems. IEEE Transactions On Intelligent Transportation Systems, 15(4):1567 – 1578, 2014.

M. Rainer-Harbach, P. Papazek, G.R. Raidl, B. Hu, and C. Kloimllner. Pilot, grasp, and vns approaches for the static balancing of bicycle sharing systems. Journal of Global Optimization, pages 1–33, 2014.

J. Rasku, T. Kärkkäinen, and P. Hotokka. Solution space visualization as a tool for vehicle routing algorithm development. In M. Collan, J. Hämäläinen, and P. Luukka, editors, LUT Scientific and Expertise Publications - Research Reports, volume 13 of Proceedings of the Finnish Operations Research Society 40th Anniversary Workshop FORS40, pages 9–12, 2013.

T. Raviv and O. Kolka. Optimal inventory management of a bike-sharing station. IIE Transactions, 45(10):1077–1093, 2013.

T. Raviv, M. Tzur, and I.A. Forma. Static repositioning in a bike-sharing system: models and solution approaches. EURO Journal on Transportation and Logistics 2.3, pages 187–229, 2013.

R. Regue and W. Recker. Proactive vehicle routing with inferred demand to solve the bikesharing rebalancing problem. Transportation Research Part E: Logistics and Transportation Review, 72:192 – 209, 2014.

J. Schuijbroek, R.C. Hampshire, and W.-J. van Hoeve. Inventory rebalancing and vehicle routing in bike sharing systems. European Journal of Operational Research, 257(3): 992 – 1004, 2017.

S. Sharma. Applied Multivariate Techniques. John Wiley & Sons, Inc., New York, NY, USA, 1996.

K. Sörensen and N. Vergeylen. Computer Aided Systems Theory – EUROCAST 2015: 15th International Conference, Las Palmas de Gran Canaria, Spain, February 8-13, 2015, Revised Selected Papers, chapter The Bike Request Scheduling Problem, pages 294–301. Springer International Publishing, Cham, 2015.

T. Walsh. General symmetry breaking constraints. In International Conference on Principles and Practice of Constraint Programming, pages 650–664. Springer, 2006.

F. Wickelmaier. An introduction to mds. Technical report, Sound Quality Research Unit, Aalborg University, Denmark, 2003.

D. Zhang, C. Yu, J. Desai, H.Y.K. Lau, and S. Srivathsan. A time-space network flow approach to dynamic repositioning in bicycle sharing systems. Transportation Research Part B: Methodological, 2016.