Search space analysis of combinatorial problems: a case study

Nicholas Vergeylen, Kenneth Sörensen, Daniel Palhazi Cuervo
Engineering management - ANTOR Research Group,
University of Antwerp, Belgium
Email: Nicholas.Vergeylen@uantwerpen.be

September 14, 2016

Abstract

It is common in the literature on combinatorial problems for algorithm designers to state the selection of heuristic components without motivating them. Providing such a motivation is not an easy task. Sometimes motivation is based on historical success of a heuristic for "similar" problems, assuming that its effectiveness remains constant for the observed changes in problem structure. We provide arguments for approaching the problem of motivating algorithmic component selection by studying the combinatorial properties of the optimization problem at hand. This analysis will make the design process more rational and therefore provide the designer with a better level of support for his or her decision as to which components to introduce. Without making any claim on completeness or resulting argument strength, we believe that following this process will generally be beneficial for heuristic quality, as it can identify (in-)effective solution strategies early in the algorithm design phase, based on the problem characteristics. In this contribution we illustrate such an analysis by applying it on the search space of the city Bike Request Scheduling Problem (BRSP), which is a NP hard problem in the context of bicycle repositioning.

Keywords: Search Space Analysis, Bike Request Scheduling Problem, Bicycle Repositioning, Multi Dimensional Scaling

1 Introduction

1.1 Search space analysis

Search space analysis for combinatorial problems is not new. Reference works such as [**?**] and to a lesser degree [Michalewicz and Fogel, 2000] describe and/or use it to provide more insight into the described optimization techniques. In these reference works, however, one does not explicitly find a search space analysis phase in the design process, perhaps leading to an underutilized area of study for the fast growing body of new combinatorial problems, among which many variants of the VRP ([Eksioglu et al., 2009]). Moreover, since, in theory, no algorithm can be better than any other for all optimization problems according to the NFL theorem in search and optimization ([Wolpert and Macready, 1997]), it can pay off to introduce problem specific knowledge on the problem class of interest into the solution algorithm. It is the main reason a solution algorithm can become more effective for this class, in comparison to other algorithms. Analyzing the search space can provide

that kind of information in terms of solution distance properties, given the selection of an operator linking solutions and its related distance measure.

Visualizing the search space is a complementary analysis step with respect to deriving formal properties. In the algorithm design phase, which is explorative by nature, iterating between both analysis steps can channel the search for good heuristic components. There are, however, some technical difficulties in doing this. Given a certain distance metric defined between solutions, the solutions reside in, say, ρ dimensions, where ρ is unknown: there are no explicit coördinates, only a pairwise distance matrix. A multivariate technique called multi-dimensional scaling ([Buja and Swayne, 2002, Wickelmaier, 2003]), or MDS in short, can be used to provide dimensionality reduction and a low-dimensional representation of a potentially high dimensional space. In this paper we utilize a method similar to the one proposed in [Rasku et al., 2013], who tries to visualize search spaces of VRP problems. In this method 5 steps are followed: (1)1. generate the entire solution space, 2. build the neighborhood structure, 3. build a distance matrix, 4. perform a type of MDS to generate 2D coördinates, and finally 5. optionally embellish the points with problem specific data such as objective value, feasibility and/or evaluation function value.

1.2 Bicycle repositioning

Bicycle Sharing Systems (BSSs) are examples of pooling systems, where resources are shared among users in order to deliver higher service to cost ratios on community level, when compared to individual acquisition policies. In short, in a BSS, users can pick up a bicycle at one of the many stations distributed in an urban environment and leave it at one of the stations once the trip is done. BSSs give rise to an interesting area of research in operations research. This can be explained because of its booming popularity. In 2014 The Institute for Transportation and Development Policy (ITDP) reported over 600 implemented BSSs and this number still increases every year. The spectrum of topics that are under study range from network design, such as Lin et al. [2013] and García-Palomares et al. [2012], to measuring urban mobility patterns, like Vogel et al. [2011] and Kaltenbrunner et al. [2010].

Demand and supply at specific rental stations are rarely balanced, and differences in the long or the short term might cause stations to fill up or deplete, preventing users from collecting or returning bikes. BSSs therefore use a fleet of light vehicles to transfer bikes between stations, attempting to rebalance the system. The vehicles are in constant contact with the dispatching station, where the current inventory of each station is monitored, and from where the repositioning activities are directed. Repositioning, together with maintenance, are the most important operational activities according to a report by the OBIS project, published in 2011 (OBIS [2011]). It is not surprising, therefore, that the most studied topic is the routing of the repositioning vehicles. It is due to the growing domain popularity and novelty of its proposed combinatorial problems that we opted to select one of these problems for illustrating the application of search space analysis.

1.3 The BRSP

In Sörensen and Vergeylen [2015] a novel approach was propopsed that tackles the problem of bicycle repositioning by decomposing it into two distinct subproblems: (1) 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) the assignment of these requests to vehicles and the scheduling of requests within each vehicle. The latter problem, which is called the *bike*

Table 1: Request attributes

Property		Description
issue time	IT	The time at which a request is created (0 in the static problem)
id	ID	The unique identifier of a request within the instance
station id	SID	The unique identifier of a station within the instance
quantity	Q	The number of bikes to (un)load
early arrival time	EAT	The earliest time for serving a request
late arrival time	LAT	The latest time for serving a request
droptime	DT	The time it takes for serving a request
priority	P	A number specifying the relative importance of a request

request scheduling problem (BRSP), assumes that the instructions for loading and unloading, called requests, are known and considered fixed while scheduling vehicle routes. Due to the lack of information on the routing constraints while generating this list, not all requests will necessarily be executable without introducing constraint violations, and therefore the aim is to minimize the priority-weighted sum of unscheduled requests when generating a feasible solution to the routing problem.

The BRSP is an elegant and simple yet novel vehicle routing variant, that combines aspects of the one-commodity pickup and delivery problem (1-PDP) [Hernández-Pérez and Salazar-González, 2004] and the orienteering problem (Golden et al. [1987], Gunawan et al. [2016]). The lack of existing heuristics, as well as having a non-straightforward evaluation function renders this problem as an interesting candidate to illustrate the analysis. Furthermore, the concrete results can be useful for designers of heuristics utilizing the distance metric we analyze.

The remainder of this contribution illustrates the findings of the search space analysis. This analysis consisted of a cardinality analysis, which leads to a sharper formulation of the existing model by introducing symmetry-breaking constraints, presented in section 2, as well as a formula for calculating search space cardinality and results on complexity indicating that the search space grows in a super exponential rate, which confirm prior experimental findings (Sörensen and Vergeylen [2015]). The latter results are presented in section 3. The selection of a distance metric and corresponding properties are presented in section 4 and the search space visualizations are presented in section 5. Finally a conclusion is presented in 6.

2 The BRSP MIP model

2.1 Original formulation

In Sörensen and Vergeylen [2015] the BRSP was introduced. In this problem the existence of a list of vehicle instructions or "requests" is assumed to be given and to remain fixed throughout time. These requests define what needs to happen at what time at what place, and they are executed by a fleet of vehicles of the same capacity. The characteristics of such a request are shown in table 1. This list can be generated by a machine, or by a human. It is the aim of the BRSP to determine at most one sequence of such requests for each vehicle in the repositioning fleet such that they can be physically executed and the number of unserved requests, weighted by their priority, is minimal.

The mathematical formulation from Sörensen and Vergeylen [2015] is repeated below. In table 2, the meaning of each symbol is given and in table 3, the decision variables are given.

Table 2: Parameters of the static BRSP formulation

I	set of requests
N	set of positions within a vehicle tour
K	set of vehicles
b_i	number of bikes picked up or delivered for request i ($< 0 \rightarrow$ delivery, $> 0 \rightarrow$ pick-up)
p_i	priority of request i
o_i	time consumption for executing request i
a_i^e	earliest arrival time for request i
a_i^l	latest arrival time for request i
t_{ij}	travel time from request i to request j
C	capacity of the vehicles

Table 3: Decision variables of the static BRSP formulation

x_{in}^k	1 if request i is served as the n-th request of vehicle k, 0 otherwise
y_i	1 if request i is not served, 0 otherwise
a_i	arrival time at request i
l_k	initial load of vehicle k
z_{ij}	1 if request j is visited immediately after request i by the same vehicle

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

$$\min \sum_i y_i p_i \tag{1}$$

s.t.

$$\sum_k \sum_n x_{in}^k + y_i = 1 \qquad \forall i \in I \tag{2}$$

$$z_{ij} \geq x_{j(n+1)}^k + x_{in}^k - 1 \qquad \forall i, j \in I, k \in K, n \in N \tag{3}$$

$$\sum_i x_{i(n+1)}^k \leq \sum_i x_{in}^k \qquad \forall k \in K, n \in N \tag{4}$$

$$\sum_i x_{in}^k \leq 1 \qquad \forall n \in N, k \in K \tag{5}$$

$$l_k + \sum_i \sum_{n' \leq n} x_{in'}^k b_i \leq C \qquad \forall n \in N, k \in K \tag{6}$$

$$l_k + \sum_i \sum_{n' \leq n} x_{in'}^k b_i \geq 0 \qquad \forall n \in N, k \in K \tag{7}$$

$$a_j \geq a_i + o_i + t_{ij} - (1 - z_{ij})M \qquad \forall i, j \in I \tag{8}$$

$$a_i^e \leq a_i \leq a_i^l \qquad \forall i \in I \tag{9}$$

$$x_{in}^k, z_{ij}, y_i \in (0, 1) \qquad \forall i \in I, k \in K, n \in N \tag{10}$$

$$l_k \in [0, C], \forall k \in K \tag{11}$$
$$a_i \geq 0 \qquad \forall i \in I \tag{12}$$

The objective is to minimize the number of unscheduled requests, weighted by their priority. Constraints 2 make sure a request is scheduled at most once. Constraints 4 and 5 respectively make sure a sequence of vehicle requests contains no empty positions and does not contain more than one request per sequence position. Constraints 6, 7 and 11 make sure the vehicle capacities are respected. The form of constraints 6 and 7 is different from the original formulation in that l_k is now introduced to allow for nonzero initial loads. This modification makes the model more realistic, less constrained but does not change NP hardness, since the original formulation is a special case of the presented model. Constraints 3, 8 and 9 make sure the request time windows are respected. M is an arbitrarily large number.

Note that the travel time matrix provides travel times between stations, rather than between requests, since they can be deduced from the smaller matrix. From this formulation it is clear that a solution allows for a (possibly empty) sequence of requests for each vehicle. A feasible solution makes sure that every vehicle can serve every request in its sequence, respecting the vehicle capacities and the request time windows. A solution is optimal when no other feasible solution has a lower amount of requests, weighted by their priority, left unscheduled.

2.2 Symmetry-breaking constraints

The presented formulation allows for two solutions to be different when only the vehicles are permuted. In the case of a homogeneous fleet, however, these solutions should be considered identical. Therefore symmetry-breaking constraints 13, 14 and 15 are added to make the model search space not larger than strictly required. This enhancement can enhance efficiency of exact methods. The new decision variables, are presented in table 4.

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_i	ID of request i (unique within the instance context)

$$\sum_{i,j} x_{ij}^k \geq \sum_{i,j} x_{ij}^{k+1} \qquad \forall i \in I, j \in N, k \in K \setminus |K| \tag{13}$$
$$\sum_{i,j} x_{ij}^k - \sum_{i,j} x_{ij}^{k+1} \geq q_k \qquad \forall i \in I, j \in N, k \in K \setminus |K| \tag{14}$$
$$\sum_{i} x_{i0}^{k+1} \kappa_i + q_k M \geq \sum_{i} x_{i0}^k \kappa_i \qquad \forall i \in I, j \in N, k \in K \setminus |K| \tag{15}$$
$$q_k \in \{0, 1\} \qquad \forall k \in K \setminus |K| \tag{16}$$

Constraints 13 indicate that the vehicle sequences are ordered in a non-increasing sequence. 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. Introducing constraints 15 then implies allowing only the permuted solution where the IDs of the first request in one of the equal sized sequences is in

decreasing order. This constraint structure extension 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 Cardinality of the search space

We know the BRSP is NP-hard. But how fast does the search space grow with instance parameters? It is this question that is addressed here. In answering this question, an interesting structure is discovered that is discussed later in section 4

A BRSP instance $\Im$ specifies all request attributes (in total, 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 search space or solution space of $\Im$. As no confusion is possible, from now on the subscript $\Im$ will be omitted.

A solution is defined by the assignment of a specific sequence of requests to each vehicle. Therefore, the size of the search space, given by $|S|$, is determined only by n and v. Assume $v \leq n$, without loss of generality. In section 2 some effort was spent in restricting the search space so that two solutions are considered equal when the sequences of requests are equal, up to a permutation of vehicles. Now that it is clear when a solution is considered equal and what it is that constitutes a solution, the entire set of solutions can be counted.

Naturally a divide and conquer strategy emerges in doing this: (1) select a set of, say, k requests to be scheduled, (2) permute these k requests, and (3) partition each permuted sequence up to v parts. Each of these partitions will be counted. The resulting number of solutions by this process is given by expression 17.

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

Here, C_n^k is the binomial coëfficient that counts the number of selections of k requests out of n, $k!$ counts the number of permutations of this selection, $\mathbb{P}(j,k)$ is the partition function that counts the possible partitions of a sequence of k requests into j parts. For more information on the partition function, the reader is refered to Andrews [1984]. The partitions represent the assignment of a selection of requests to one or more vehicles. Note that the two sums in 17 are very alike, but differ in the input of the partition function. The terms in the left sum represent solutions where k, the number of scheduled requests, is lower than v. Therefore at most k vehicles can be used, which means at most k partitions can occur. The terms in the right sum represent selections of requests that exceed v. The fleet has at most v vehicles to route, therefore the maximum number of vehicles that can be used is fixed to v.

The proposed process will count some solutions multiple times, however, since partition parts (read: vehicle sequence lengths) of equal size are permuted. 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 $[1, 2][3, 4][5]$. A permutation of this sequence, say $3, 4, 1, 2, 5$, will have one partition $[3, 4][1, 2][5]$, which is equal to the first in terms of BRSP solution. In order to correct this situation, every numerical partition that contains equal terms should correct its count by the number of equal summand permutations. 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. 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. The third row demonstrates the redundancy. The first, second and third vehicle get the first, second and third request respectivelly, after permuting them. In reality this permutation is completely redundant, since no distinction can be made between the vehicles. Hence the factor of $\frac{1}{3!}$ is used to correct the count of these equal sequences, and set it to one.

Table 5: Counting all solutions where 3 requests are scheduled in 3 vehicles

Permutation	Partition representation	Partition count	Total
3!	•••	1	3!
3!	•• •	1	3!
3!	• • •	$\frac{1}{3!}$	1
		Σ	13

The function of counting partitions, just like $\mathbb{P}$, and then correcting for permutation of equal parts or summands will be denoted by $\mathbb{P}_\epsilon$. Using this notation, the correct form for the search space cardinality becomes

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

Increasing n, ceteris paribus, the form 18 has 19 as a lower bound. Here the corrected partition function is neglected entirely, so sharper bounds are conceivable. Even so, one can clearly see superexponential growth of the search space in n which confirms our previous empirical results of explosive behavior.

$$\frac{\sum_{k=0}^{n} k!}{(n-v)v!} \sim \mathcal{O}(n!) \tag{19}$$

4 Search space structure

In calculating the search space cardinality in the previous section, we arrive at an expression consisting of a sum of terms. These terms represent cardinalities of special subsets of the search space. We chose to elaborate on this particular partition by introducing an operator that links solutions between partitions: request insertion (RI). This operator is comparable to node insertion in the TSP context, but it differs from it since it considers removal and addition of a request as two operations, instead of one. The characteristics of this operator will now be formalized in mathematical terms.

Consider the indexed family of subsets $\{S_\alpha\}_{\alpha \in I}$ of S, where $I = [0, n]$. Membership of a solution s to such a subset is defined by the function $S \times I \to \{0, 1\} : \sigma(s, \alpha)$. This function returns 1 if the number of requests that is scheduled equates 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 $\cup_{\forall \alpha \in [0,n]} S_\alpha = S$.

This definition of the search space partition exhibits the following three properties.

Property 1: expanding partition size

$$(\forall \alpha, \beta)(\alpha \in [0, N-1] \wedge (\beta \in [0, N-1] \vee (\beta = n \wedge v > 1)) \Rightarrow (\alpha < \beta \Leftrightarrow |S_\alpha| < |S_\beta|)) \tag{20}$$

$$|S_{N-1}| = |S_N| \Leftrightarrow v = 1 \tag{21}$$

Simply put, the first property formalizes the observation that in general a lower order partition contains less elements than a higher order partition. The only case when this is not true occurs when only one vehicle is present and one is looking at the two highest partitions. Since one can find a one-to-one mapping between solutions in these two partitions (i.e.: by appending the only non-scheduled request to a solution in the second last partion.

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 injection relation as every solution in S_α has a corresponding solution in $S_{\alpha+1}$, and 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 constructable 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 property has been proven by construction.

Property 2: Transcendental infeasibility

Consider the feasibility function $S \to \{0, 1\} : feas(s)$, which defines the membership of 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 property can be formulated.

$$(\forall \alpha)(\alpha \in [1, n] \Rightarrow (\mathscr{F} \cap S_{\alpha-1} = \emptyset \Rightarrow \mathscr{F} \cap S_\alpha = \emptyset)) \tag{22}$$

To see this, note that any feasible solution in $S_\alpha, \alpha \in [1, n]$, neither has 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

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

which is equivalent to the set of implications in 22.

The property formulated in 22 could be used to develop an efficiënt branch and bound search: start a search by evaluating lower order partition solutions and branch when an infeasible solution is found, effectively cutting larger parts of the higher order partitions.

Adding or removing a request from a solution is a straightforward and useful operation. It influences the objective value immediately (unlike swapping requests for example). Moreover, unlike swapping it connects the entire search space, meaning a local search can use it and have the guarrantee that every solution is reachable. To formulate the following property a mapping on the solution set S is defined, based on the RI operator, which implies a measure of nearness.

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 either removing a request or adding a request. This set is represented by $N_{RI}(s_k)$, which 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. There is a path between two solutions iff there is a sequence of solutions, starting with the first and ending with the second solution, such that each subsequent pair of solutions in the sequence are neighbors. Consider a path of n solutions, then its length 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 $d_{RI}(s_a, s_b)$.

Using this definition of distance, property 2 immediately leads to the third property.

Property 3: Nearness of feasible solutions

$$(\forall s_a, s_b)((s_a \in S_\alpha \wedge s_b \in S_\beta) \Rightarrow d_{RI}(s_a, s_b) \leq \alpha + \beta) \tag{23}$$

To see this, consider a special case:

$$(\forall s_a, \alpha)((s_a \in S_\alpha \wedge \alpha \in [0, n]) \Rightarrow d_{RI}(s_0, s_a) \leq \alpha) \tag{24}$$

Using the constructional proof for property 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-moves, as there are at least α moves required to construct the solution from s_0.

From inequality 24, 23 immediately follows by using the triangular inequality: $d_{RI}(s_a, s_b) \leq d_{RI}(s_a, s_0) + d_{RI}(s_0, s_b) = \alpha + \beta$.

This property is stronger than property 1, and states that a super exponential growing search space in the number of requests is embedded in a compact high dimensional space. Distances between feasible and infeasible solutions will remain fairly small, though their number increase rapidly. This observation suggests that searching small partitions and restricting the higher partitions by pruning infeasible solutions is an efficient approach in solving the BRSP.

5 Visualization

5.1 Dimensionality reduction

Using the distance measure that was introduced in the previous section, we try to visualize the search space to see if we can gain more insight into what type of search could or could not be useful for the BRSP. Large instances cannot be solved exhaustively, so they cannot be visualized, therefore we are limited to small instances. Instances with 5 requests and 3 vehicles were used to visualize the search space. This search space shall be denoted as (5,3). The choice for these values was to allow the search space to be complex enough to convey information, while still having a manageable search space size. Using formula 18, we calculate the size of the search space to be 1006.The generated solution space is structured by generating the neighborhood structure. The way this can be attained is by calculating first, for each solution, every other solution that can be generated by adding any unscheduled request or removing any scheduled request. By adding a link between these solutions, the neighborhood structure is created. From this structure, one can develop the distance matrix by utilizing the Floyd-Warshall algorithm, which calculates the all-pairs shortest path in $\emptyset(n^3)$ time. The Floyd-Warshall algorithm is a standard algorithm in computer science and the reader is refered to a reference work, like Cormen et al. [2001], for more information. The resulting distance matrix is instance independent, since the search space and the RI distance metric are instance independent.

The distance table is the input for a MDS analysis. An iterative procedure tries to find coördinates in the requested 2 dimensions such 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 an ordinal MDS, only the order of the distance table entries are respected, not their relative sizes. In the case of an absolute MDS, the relative sizes are respected.

After performing MDS, the estimates in 2 dimensions are not uniquely defined, so one needs to be careful in interpreting the results. The quality of the analysis is represented by the badness-of-fit (BOF) criterion, which is displayed in table 6. The MDS analysis provides us in both cases

Table 6: MDS results

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

with relatively poor results, as both BOF metrics exceed 20 %, a general cut-off value. Thus, one cannot use this two-dimensional representation as such to accurately visualize the neighborhood structure. This result indicates that, using RI as a distance measure induces a high dimensional metric space, which is explained by property 3 of the search space. In layman's terms, a super exponential growing search space with maximum integer pairwise distances growing in a linear rate requires a spatial freedom to satisfy the distance matrix that only extra dimensions can give. The ordinal MDS result was used for further analysis because of the slightly better BOF.

This poor representation does not prohibit the possibility for using the two dimensional configuration as a visualization tool. The important caveat to make is that neighbors can sometimes be located farther from or closer to each other in the plane than one would expect. Since the relevant neighbor structure is always marked on the plots, the MDS result is used as a starting point for

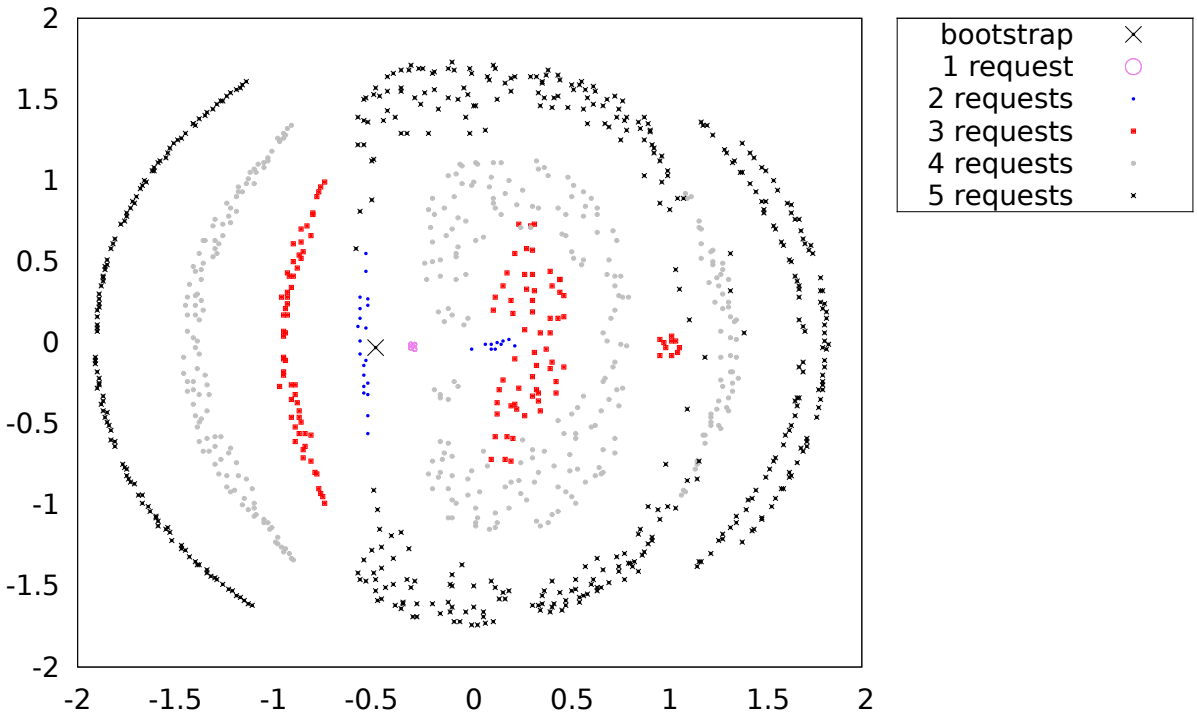


Figure 1: The 2D representation of the (5,3) search space partitions (relative distances).

visualization, rather than a result. Figure 1 shows the (5,3) search space partitions. What strikes is that from the empty inital solution, s_0, which will be called the bootstrap solution, one finds the partitions to be organized in layers outward, with the outer layers containing solutions with more requests scheduled.

The high BOF can be perceived by noting that a part of S_2 is closer to s_0 than any solution in S_1. Also, the solutions in S_1 seem to be compressed. Knowing that the MDS technique aims to minimize the global deviation from the original distance matrix, these aberrations can be explained by property 1 of the search space: table **??** shows the relative sizes of each partition, and we notice that the impact of the first three partitions is limited due to the relatively low cardinality.

5.2 From structure to search

The objective function of the BRSP may not be the best evaluation function for a search. In and by itself it does not provide means for finding potentially better solutions in the same partition as well as for assessing a direction when starting from an infeasible solution, having nothing but infeasible neighbors. To illustrate this behavior, two cases in (5,3) are presented in table 7. The first instance is "artificially" restricted, where very -unrealistic- tight time windows restrict $\mathscr{F}$. The second instance provides a more relaxed and likely instance. For each instance, fleet capacities are set to 30.

The two dimensional vizualization of (5,3) is embellished by connecting neighboring solutions that reside in $\mathscr{F}$. In figure 2a this is shown for the restricted instance, and only 3 distinct paths are

Table 7: Request properties of a restricted instance [left], of a more realisticly constrained instance [right], and the travel time matrix [bottom].

EAT	LAT	DT	Q	P	ID	SID
500	700	220	10	5	0	0
500	700	270	15	3	1	1
500	700	170	-10	5	2	2
500	700	320	-10	3	3	3
500	700	270	-15	4	4	4

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

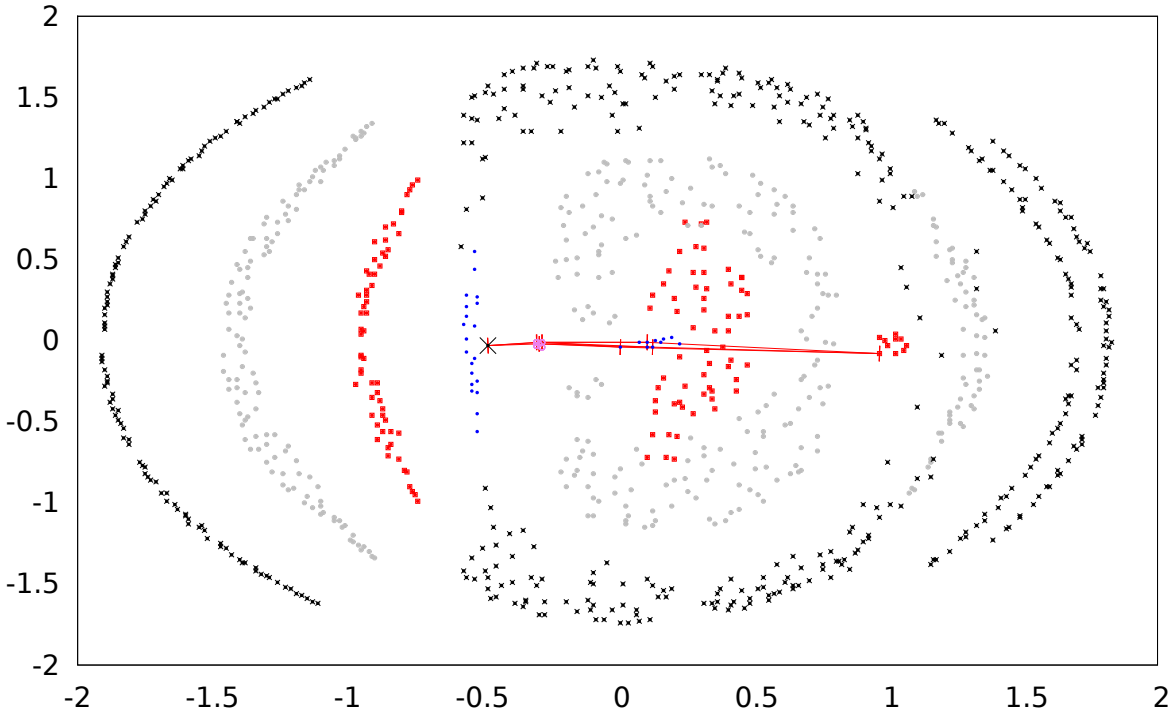
	1	2	3	4	5
1	0	1201.3	1489.46	2099.87	391.24
2	1201.3	0	1439.96	2632.09	1382.99
3	1489.46	1439.96	0	1320.16	1249.9
4	2099.87	2632.09	1320.16	0	1712.6
5	391.24	1382.99	1249.9	1712.6	0

seen. As a path, they all connect the bootstrap solution and the optimal solution [[2][3][4]]. The neighbors of every feasible solution in the path are shown in figure 2b. Note that this is only a small part of the search space. When starting with any disconnected solution in this figure, the objective function cannot provide any information on the direction of change that is advisable, i.e. towards $\mathscr{F}$, which seems to render it a bad choice as an evaluation function for a general local search. Figure 3 provides the same view on the more likely case. In this case the objective function can provide useful information in a substantial larger part of the search space. An iterated hill-climbing method could be applied here and the objective function can be used more effectively.

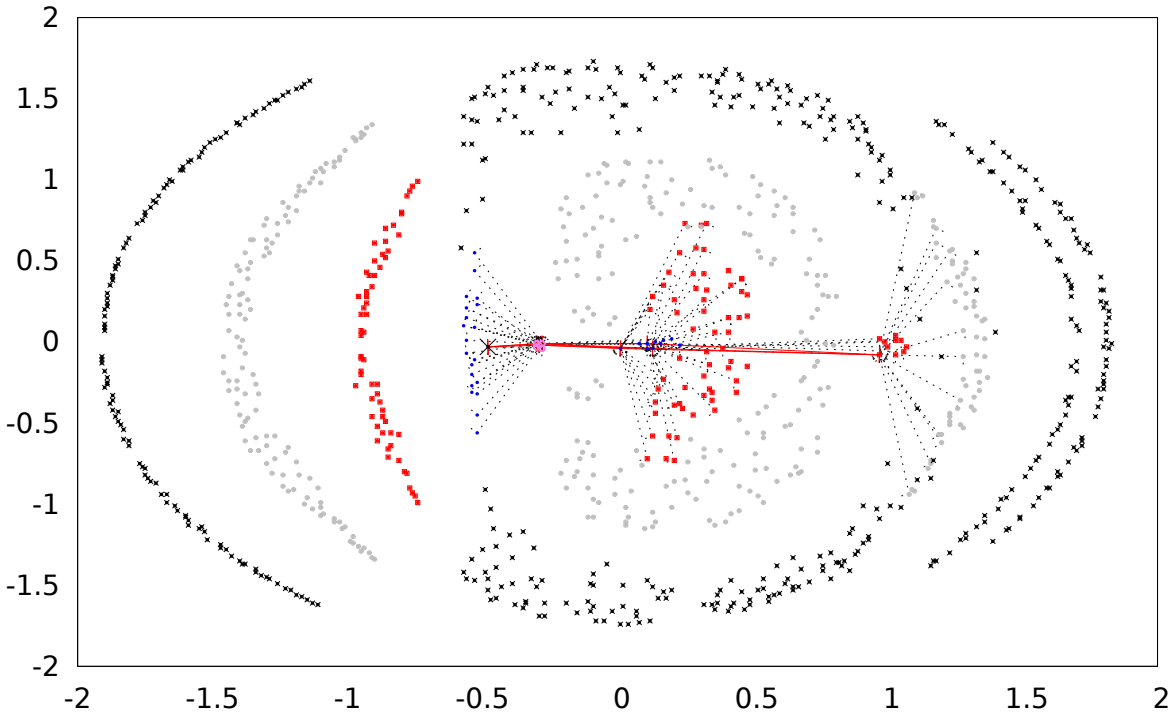
There are other, more sensible ways to search the solution space, as we will argue. Using property 2 and 3, one is motivated, to build a search that always starts from the bootstrap solution s_0 and moves iteratively from S_k to S_{k+1}. By pruning the branches where no feasible solutions could ever reside, like the branch and bound principle, the resource requirements for this search could supposedly be contained in the best way. The idea of starting from the bootstrap solution is based on the observation that, even though the higher order partitions contain more valuable solutions, the probability of finding a feasible one is lower and the search potentially becomes less efficient. Starting from the bootstrap solution guarrantees a search from starting in the feasible area.

Figures 2b and 3b show the evaluations an exhaustive search will have to make, when based on this strategy. The constrained instance is solved very effectively using this strategy, while, for the less constrained instance, the approach seems less effective as a large part of the search space still needs to be evaluated. Even so, intelligent use of the constraint structure and iteratively moving through the partition structure exhibits potential in designing heuristics, a forteriori for constrained cases. Note that this observation also empowers a dynamic programming approach, as one could start from the requests with the latest arrival time and work backwards to the first request(s) while keeping track of the alternative solutions and prune dominated ones.

Another approach to improve search effectiveness is to handle the bad aptitude of the objective function in providing a direction of optimization. By ranking any two solutions in the search space, such that finding the most extremely ranked solution in a specified direction is equivalent to finding the instance optimum, one has defined an ordinal evaluation function (Michalewicz and Fogel [2000]). As explained before, the objective function does suggest such a function, by calculating the difference in evaluation, but in this case a direct application is not effective as the evaluation landscape it produces is not useful in $\bar{\mathscr{F}}$. Only for solutions in $\mathscr{F}$ or their neighbors can

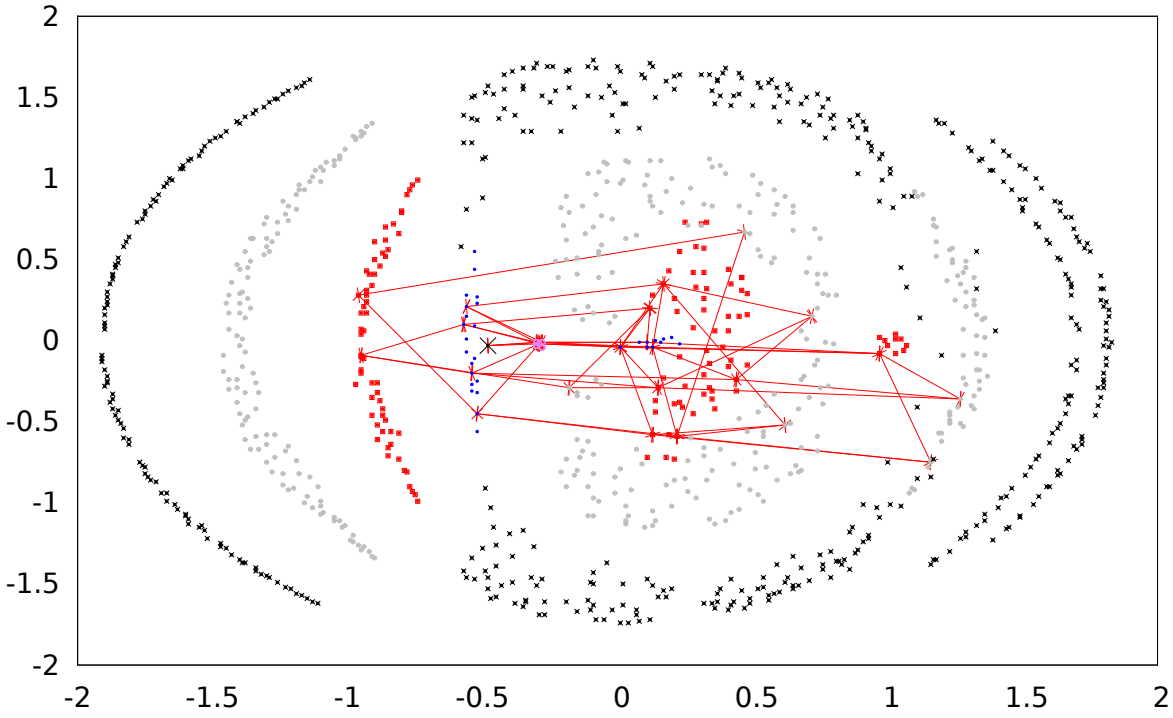


(a) Feasible paths

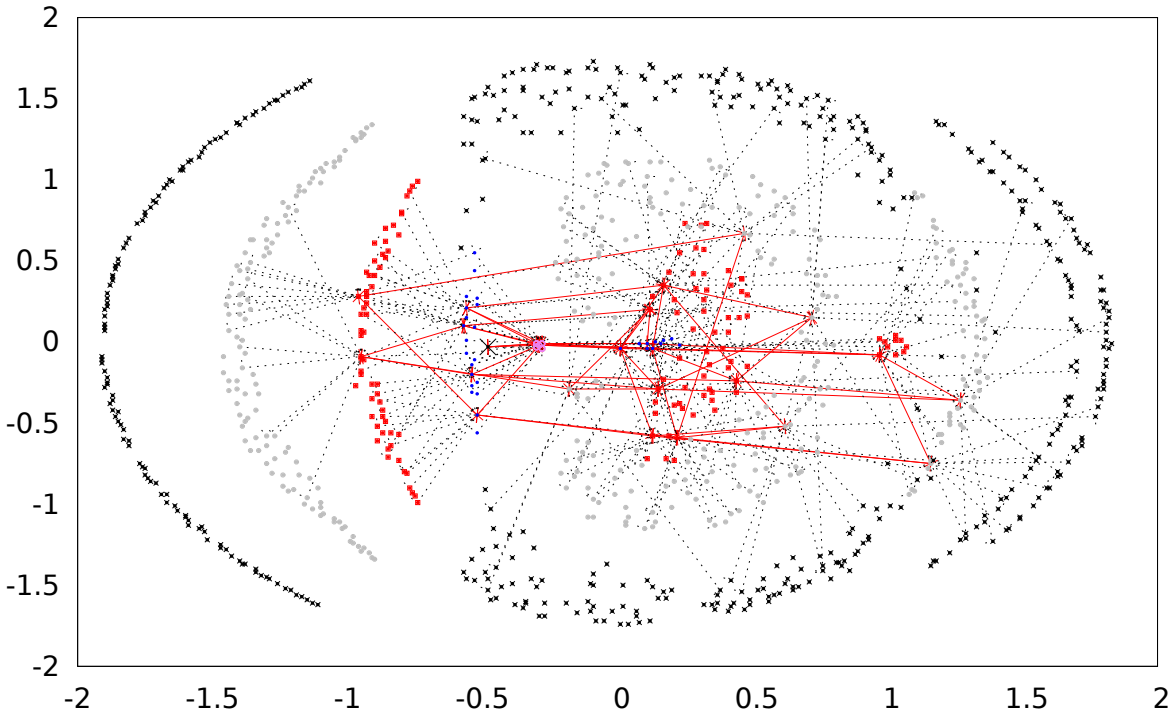


(b) Neighbors for feasible solutions

Figure 2: Visualization of a restricted instance



(a) Feasible paths



(b) Neighbors for feasible solutions

Figure 3: Visualization of a more common instance

the evaluation of the objective function lead to a direction of optimization. A simple extension of the objective function is proposed to provide useful information in $\bar{\mathscr{F}}$ as follows:

$$\begin{aligned} &find(s_a): \\ &\qquad eval(s_a, s_b) > 0 \qquad\qquad \forall s_a, s_b \in S, s_a \neq s_b \end{aligned}$$

with:

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

Here $\mathscr{O}(s)$ stands for the objective value of solution s. $\mathscr{V}^c$ represents the total number of capacity violations encountered in the solution, calculated as:

$$\mathscr{V}^c_{jk}(s) = \begin{cases} \sum_i \sum_{n' \leq j} x^k_{in'} b_i, & \sum_i \sum_{n' \leq j} x^k_{in'} b_i \geq C, \forall k \in K, j \in N \\ -\sum_i \sum_{n' \leq j} x^k_{in'} b_i, & \sum_i \sum_{n' \leq j} x^k_{in'} b_i \leq 0, \forall k \in K, j \in N \\ 0, & 0 \leq \sum_i \sum_{n' \leq j} x^k_{in'} b_i \leq C, \forall k \in K, j \in N \end{cases}$$

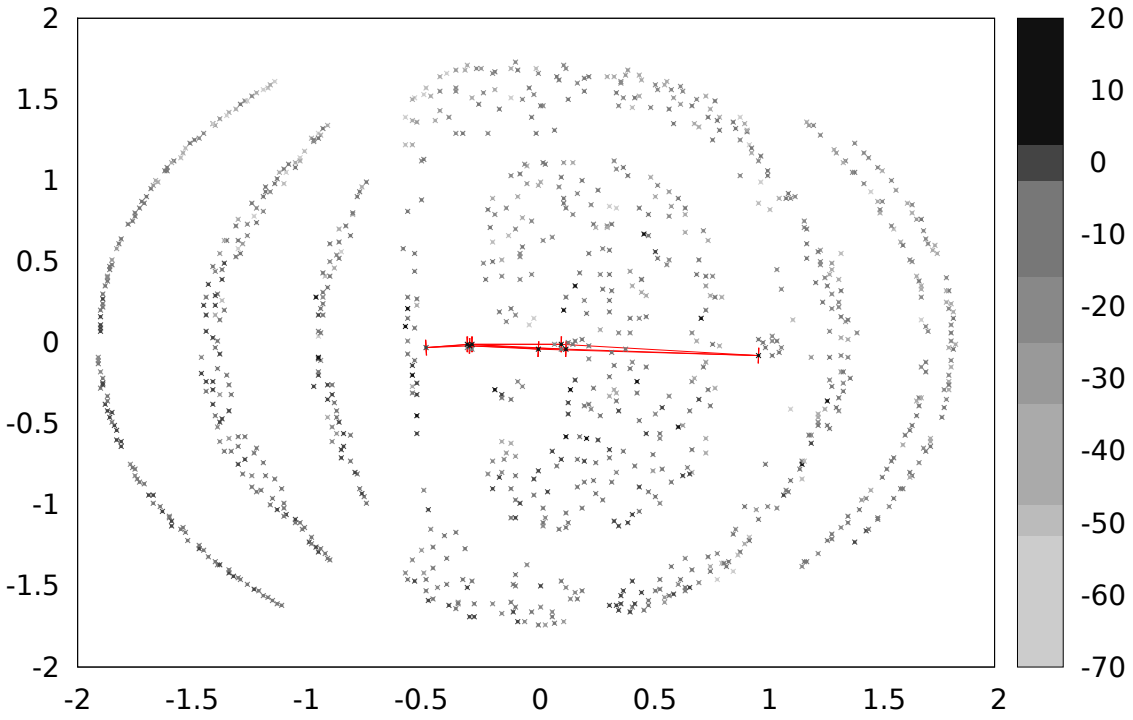
and

$$\mathscr{V}^c(s) = \sum_{\forall j,k} \mathscr{V}^c_{jk}(s) \tag{25}$$

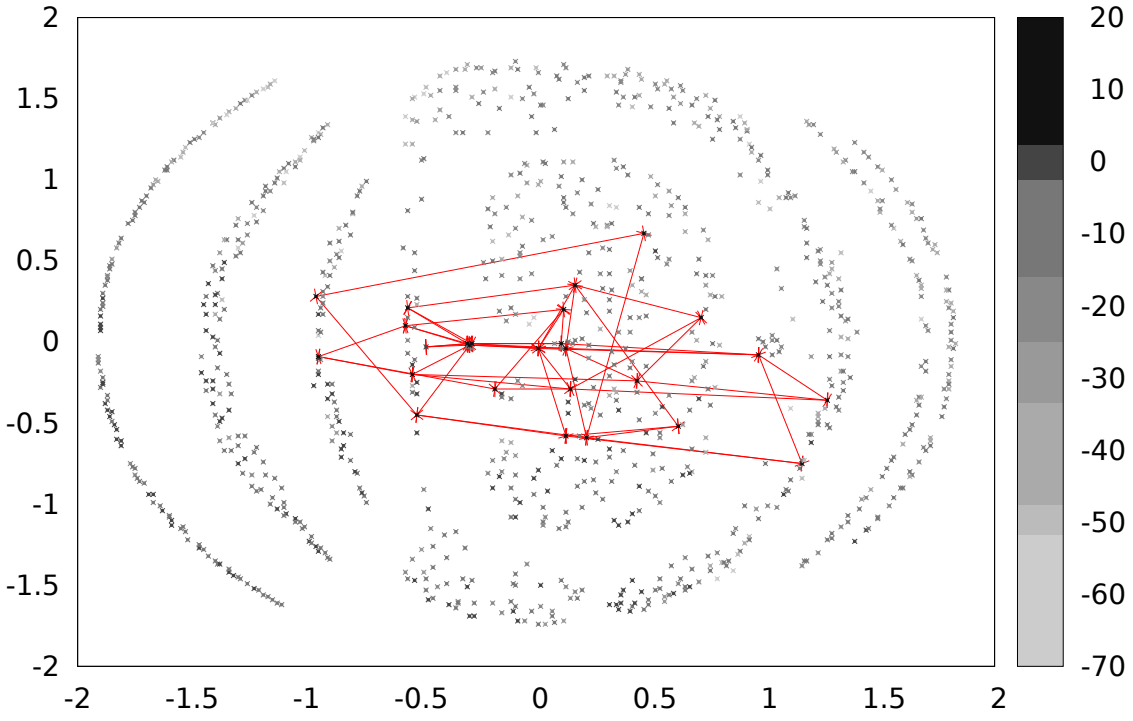
This evaluation function aims to extends the useful range of evaluation of the objective function to the entire search space. Moreover, in $\bar{\mathscr{F}}$, the direction of optimization is towards the direction of lowering the capacity violations and increasing the objective function, which is a greedy strategy. Figure 4 visualizes this strategy by using the evaluation function value as a grayscale value. The strategy does not seem to be highly effective in guiding any search towards the feasible/optimal solution(s) for the restricted case, as the evaluation function indicates for some solutions in the search space that the optimal solution is located in the leftmost part of the search space. For the "more common" instance, the results seem slightly better, as there are less dark solutions in the infeasible areas. This behavior is explained by the fact that in the first instance, time constraints are very important compared to the capacity constraints, while the evaluation function completely ignores this.

Introducing time constraint violations into the evaluation function can be done in alternative ways. Priority rules could be used, but unless one has a priori information on which constraints are more important, a linearization of both constraints seems a logical choice for combining the violations. It is less straightforward how to quantify time violations compared to the capacity violations in the simple case, as any choice implies a trade-off between the two types of constraints. A possible choice could be to use a piecewise structure such as

$$\mathscr{T}(s) = \begin{cases} \mathrm{e}^{\frac{a_i - a^l_i}{600}}, & a_i - a^l_i < 2400, \forall i \in I \\ \mathrm{e}^4, & a_i - a^l_i \geq 2400, \forall i \in I \\ 0, & a_i \leq a^l_i, \forall i \in I \end{cases}$$



(a) The restricted instance



(b) The common instance

Figure 4: A simple evaluation function

and

$$\mathscr{T}(s) = \sum_{\forall i} \mathscr{T}_i(s) \tag{26}$$

$$\mathscr{V}(s) = \mathscr{T}(s) + \mathscr{V}^c(s) \tag{27}$$

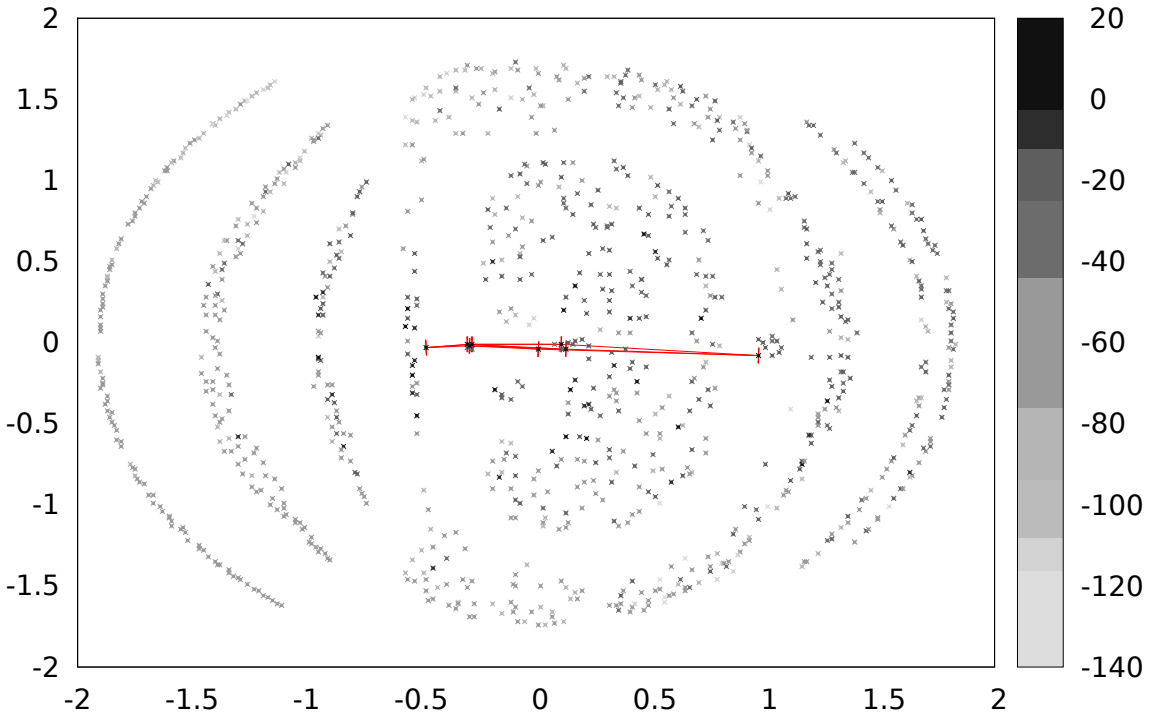
For each request, a violation of 10 minutes in the instance (=600 s), a term e is added. Around that value, it's more likely that capacity violations are prefered to be tackled. At a time violation of 40 minutes, time becomes very important (rounded value of 55). These values are used for illustration purposes and probably better values can be conceived. Figure 5 shows the effect of using $\mathscr{V}(s)$ instead of $\mathscr{V}^c(s)$ in the evaluation function. Clearly this function exhibits better behavior. A search relying on this evaluation function is guided more effectively towards the feasible solutions: The farther one ventures from the feasible area in any of both time and quantity dimensions, the more negative the evaluation function becomes.

6 Conclusion

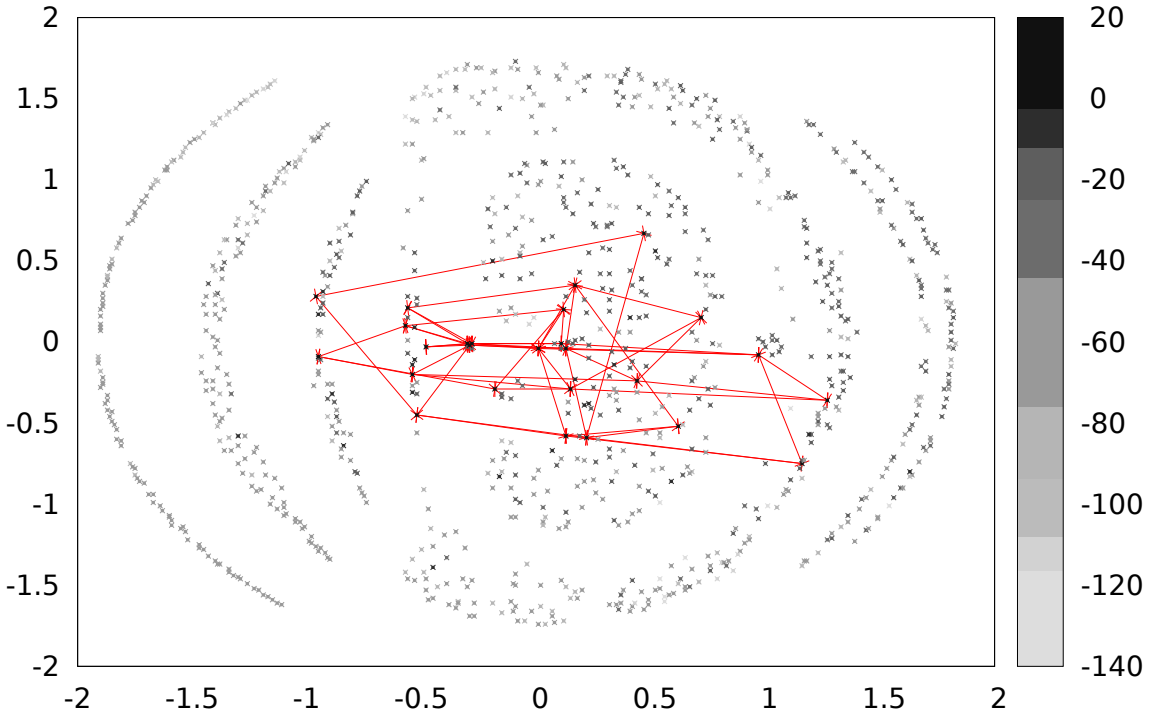
In this contribution we have illustrated a method of approaching algorihm design for solving combinatorial problems by first studying the search space. We argue that in this way one is able to motivate in a better way his or her algorithmic design decisions. The method was applied on the city bike repositioning problem (BRSP), which is a new approach for solving bicycle repositioning problems. Three aspects were considered. And the main results of applying the method are presented below

First, based on a cardinality analysis we enhanced the MIP formulation, by introducing symmetry-breaking constraints in order for the model search space to resemble the true BRSP search space. A formula for calculating search space cardinality was found as well which shows that the search space grows in a super exponential rate with increasing number of requests. By proposing a request-insertion (RI) operator as well as a partitioning structure, three properties could be formulated that can aid heuristic design. Other metric structures can be conceived in future research, and it is not yet clear if they could lead to more powerful properties. It would be interesting, in this regard, to find an indication of which metric structures to use or not to use, based on problem characteristics. Not only does this contribution lead to this interesting research question, it makes clear that useful properties can emerge from the proposed methodological process that are not evident a priori. The partitioning leads to the theoretical observation that one could reduce the computational effort of finding the optimal solution by starting from the empty, or bootstrap solution and keeping track of smaller sets of feasible solutions, as their infeasible neighbors restrict possibly large parts of the solution space that can never contain any feasible solution. This idea provided the basis for an efficient branch and bound. Thirdly, the search space was visualized. The partitioning structure also emerged in the visualization. Interestingly, this approach shows that using the objective function in a direct way as an evaluation function is not very effective within the RI-metric.

When considering heuristic solution algorithms, it seems that, within the limits of the imposed RI metric structure, large changes in solution structure are needed in order to escape local optima which can be located far from each other. Restart approaches as well as destroy/repair methods seem suitable for the BRSP, especially in highly constrained instances, where the objective function is poor in suggesting a direction of optimization. The main caveat in using the destroy/repair methods is that a modified evaluation function should be used. If this were not the case, the search



(a) The restricted instance



(b) The common instance

Figure 5: A combined-constraint evaluation function

would not be guided in an intelligent way once the incumbent solution would be infeasible, reducing the search strategy to a quasi-random search.

Some evaluation functions were explored that have better guidance potential compared to the objective function. More complex evaluation functions can be conceived, in order to modify or change the evaluation landscape and to smoothen it, which can be explored in future research.

References

G.E. Andrews. *The Theory of Partitions*. Cambridge University Press, 1984. ISBN 9780511608650. URL http://dx.doi.org/10.1017/CBO9780511608650. Cambridge Books Online.

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

T.H. Cormen, C. Stein, R.L. Rivest, and C.E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001. ISBN 0070131511.

B. Eksioglu, A. Volkan Vural, and A. Reisman. The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, 57(4):1472 – 1483, 2009. ISSN 0360-8352.

J.C. García-Palomares, J. Gutiérrez, and M. Latorre. Optimizing the location of stations in bike-sharing programs: A {GIS} approach. *Applied Geography*, 35(12):235 – 246, 2012. ISSN 0143-6228. doi: http://dx.doi.org/10.1016/j.apgeog.2012.07.002. URL http://www.sciencedirect.com/science/article/pii/S0143622812000744.

B.L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics (NRL)*, 34(3):307–318, 1987.

A. Gunawan, H.C. Lau, and P. Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, pages –, 2016. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2016.04.059. URL http://www.sciencedirect.com/science/article/pii/S037722171630296X.

H. Hernández-Pérez and J.-J. Salazar-González. Heuristics for the one-commodity pickup-and-delivery traveling salesman problem. *Transportation Science*, 38(2):245–255, 2004.

A. Kaltenbrunner, R. Meza, J. Grivolla, J. Codina, and R. Banchs. Urban cycles and mobility patterns: Exploring and predicting trends in a bicycle-based public transport system. *Pervasive and Mobile Computing*, 6(4):455 – 466, 2010. ISSN 1574-1192. doi: http://dx.doi.org/10.1016/j.pmcj.2010.07.002. URL http://www.sciencedirect.com/science/article/pii/S1574119210000568. Human Behavior in Ubiquitous Environments: Modeling of Human Mobility Patterns.

J.R. Lin, T.H. Yang, and Y.C. Chang. A hub location inventory model for bicycle sharing system design: Formulation and solution. *Computers & Industrial Engineering*, 65 (1):77 – 86, 2013. ISSN 0360-8352. doi: http://dx.doi.org/10.1016/j.cie.2011.12.006. URL http://www.sciencedirect.com/science/article/pii/S0360835211003792. Intelligent Manufacturing Systems.

Z. Michalewicz and D.B. Fogel. *How to Solve It: Modern Heuristics*. Springer-Verlag New York, Inc., New York, NY, USA, 2000. ISBN 3-540-66061-5.

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

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.

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. ISBN 978-3-319-27340-2. doi: 10.1007/978-3-319-27340-2_37.

P. Vogel, T. Greiser, and D.C. Mattfeld. Understanding bike-sharing systems using data mining: Exploring activity patterns. *Procedia - Social and Behavioral Sciences*, 20:514 – 523, 2011. ISSN 1877-0428. doi: http://dx.doi.org/10.1016/j.sbspro.2011.08.058. URL `http://www.sciencedirect.com/science/article/pii/S1877042811014388`. The State of the Art in the European Quantitative Oriented Transportation and Logistics Research 14th Euro Working Group on Transportation & 26th Mini Euro Conference & 1st European Scientific Conference on Air Transport.

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

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Trans. Evol. Comp*, 1(1):67–82, apr 1997. ISSN 1089-778X.