

# Hybrid Numerical Optimization for Combinatorial Network Problems

Markus Chimani, Maria Kandyba<sup>\*†</sup>, Mike Preuss<sup>‡</sup>

Dortmund University, 44221 Dortmund, Germany,  
<http://ls11-www.cs.uni-dortmund.de>

**Abstract.** We discuss a general approach to hybridize traditional construction heuristics for combinatorial optimization problems with numerical based evolutionary algorithms. Therefore, we show how to augment a construction heuristic with real-valued parameters, called *control values*. An evolutionary algorithm for numerical optimization uses this enhanced heuristic to find assignments for these control values, which in turn enable the latter to find high quality solutions for the original combinatorial problem. Additionally to the actual optimization task, we thereby experimentally analyze the heuristic’s substeps.

Furthermore, after finding a good assignment for a specific instance set, we can use it for similar yet different problem instances, without the need of an additional time-consuming run of the evolutionary algorithm. This concept is of particular interest in the context of computing efficient bounds within Branch-and-Cut algorithms. We apply our approach to a real-world problem in network optimization, and present a study on its effectiveness.

## 1 Introduction

We consider a hybrid approach to use numerical evolutionary algorithms for solving NP-complete combinatorial network design problems. Most traditional hybridization approaches consist of developing combinatorial representations of solutions for use in evolutionary algorithms, or of developing meta-search heuristics which use traditional heuristics as subroutines. This results in algorithms which have, in general, a much longer running time than traditional combinatorial construction heuristics.

We divert from these approaches by taking an existing construction heuristic and augmenting it with *control values* (CVs). These are simple mostly non-discrete values, which allow us to modify the behaviour of the algorithm and thereby change its outcome: e.g., we may have a CV specifying a degree of

---

<sup>\*</sup>Supported by the German Research Foundation (DFG) through the Collaborative Research Center “Computational Intelligence” (SFB 531)

<sup>†</sup>Alphabetical sorting, corresponding author: [maria.kandyba@cs.uni-dortmund.de](mailto:maria.kandyba@cs.uni-dortmund.de)

<sup>‡</sup>Supported by the DFG project grant no. 252441, “Mehrkriterielle Struktur- und Parameteroptimierung verfahrenstechnischer Prozesse mit evolutionären Algorithmen am Beispiel gewinnorientierter unscharfer destillativer Trennprozesse”.

sortedness when performing some heuristic step on a series of items; we may have a real-valued CV, as a balancing parameter to compute a weighted sum of two different optimization subcriteria, etc.

In general, we will not know in advance which settings for these CVs are in fact beneficial. We optimize the performance of the heuristic on a specific problem instance or instance set with a general-purpose optimization algorithm, namely an *evolutionary algorithm* (EA) designed to operate on numerical values. We find near-optimal settings  $\chi^*$  for these CVs, using the augmented heuristic to compute the objective value for any specific CV instance  $\chi$ . Whereas this may appear as 'yet another EA application' on first sight, it is more than that, because

- the EA actively modifies the behaviour of the heuristic and identifying viable CV settings may not be reasonably possible without the composition with an optimization algorithm, and
- the performance feedback allows us to obtain new insight into the effectiveness of specific steps of the heuristic, which may lead to further ideas for improving the underlying algorithm.

In this paper we analyze this novel approach, by concentrating on a single application as a test case for the suggested high-level relay hybridization [7]. Nonetheless, we believe that this scheme may be successfully utilized for virtually every construction heuristic that contains components which have been added on the basis of ad-hoc or intuitive decisions. Our approach is particularly interesting in the realm of Branch-and-Cut algorithms.

Our test case is the real-world minimization problem [8] which arises when one wants to extend an already existing, city-wide fiber-optics or telecommunications network. This problem is known as the 2-root-connected prize-collecting Steiner network problem (2RPCSN) and has been studied, e.g., in [3, 8, 9]. In these papers, different integer linear programming approaches are proposed, which solve this problem to provable optimality, albeit in exponential worst-case time. These schemes use Branch-and-Cut techniques and employ heuristic algorithms as subroutines to generate upper bounds. The currently most successful such heuristic is described in [3]. The heuristic plays three distinct crucial roles for solving 2RPCSN in practice:

- Due to the NP-completeness of the problem, and the therefore exponential running times, computing a provable optimal solution is only feasible for small to medium sized instances. For large instances, one has to resort to heuristic approaches.
- ILP-based approaches benefit from good initial solutions. They often allow faster computation of LP relaxations, and can be used to obtain initial variable sets in column-generation based approaches, i.e., when certain variables of a linear program are only added later if required.
- Branch-and-Cut approaches use upper bounds to cut off subtrees in their Branch-and-Bound trees. These upper bounds are computed using heuristics which can deal with the fact that certain variables are fixed at some Branch-

and-Bound node. Thereby, sophisticated bounding and efficient heuristics allow to solve larger instances and therefore increase the applicability of optimal algorithms.

Based on these demands, we can differentiate between different use-cases for our hybridization:

**Online:** The heuristic consists of running the evolutionary algorithm and outputs the best solution found. The CV optimization is thereby a vehicle to steer the search for different CV instances. The running time is much slower than using only the combinatorial construction scheme, but the expected quality of the solutions improves.

**Offline:** Alternatively, we can run the CV optimization on a set of test problem instances. The obtained CV instance  $\chi$  can then be used to solve new problem instances. The selection of  $\chi$  is therefore an educated guess which can be expected to be suitable for instances similar to the test problem instances. The major advantage is that the resulting running time is virtually the same as for the original construction heuristic.

**Mixed:** In the realm of Branch-and-Cut approaches, we can blend these two approaches in a very natural way: we can use the online variant to generate both a good initial solution for the problem instance, and also obtain a CV instance  $\chi^*$  which is tuned specifically to this instance. During the time-critical steps within the branching strategy, we can use the offline variant with  $\chi^*$  to obtain a good and fast bounding heuristic. Note that we do not have to restrict ourselves to a single  $\chi^*$ , but we can use, e.g., multiple CV instances from the last population of the evolutionary algorithm.

We try to answer the specific questions summarized in Section 2. Afterwards we describe our general hybridization approach and the evolutionary algorithm in Section 3, before centering on our specific construction heuristic and its control values in Section 4. Section 5 and Section 6 focus on the conducted experiments and their analysis.

## 2 Aims

The presented hybridization for the above combinatorial network problem is ment as a proof-of-concept. Therefore our aims are centered on showing its success. Although we cannot guarantee the usefulness of this hybridization technique for completely different problems, we do suggest to try the approach if one can think of a way to make a specific heuristic configurable.

Before we can legitimately consider one of the three outlined uses of our approach, we have to investigate if the hybridization is beneficial at all. Hence, the concrete aims pursued in this work are:

1. We want to determine if our EA is able to identify CV instances for the construction heuristic which improve its performance significantly both on single instances and on sets of similar instances. We thus ask if the heuristic can be automatically adapted to considered problems.

2. If Aim 1 can be positively answered, we want to identify where this adaptability stems from, i.e., which of the heuristic’s steps contribute most to an improved performance, and which CVs can be safely set to default values.

As we expect that the heuristic can benefit from specific properties of certain problem instances, the detection of such properties is a side goal. Furthermore, the applied EA variant itself shall be assessed, to obtain a first impression concerning its usefulness for the given task. In the light of a unified EA approach [4], this resembles the question for a suitable parametrization of the EA.

### 3 Hybridization

In general, a non-trivial construction heuristic for an NP-hard problem consists of multiple steps. As the heuristic cannot guarantee to find an optimal solution, there may be multiple orthogonal reasons for non-optimality, e.g.:

- A single step may be not solvable to optimality in polynomial time, thus requiring a sub-heuristic.
- The optimization goal of some step  $s$  may only be an educated guess in the context of the complete algorithm, i.e., the optimal solution for the step  $s$  may in fact be suboptimal for the subsequent steps.
- Assume there are multiple steps, whereby their order is crucial for the outcome of the whole algorithm. The optimal order may be a priori unknown.

The idea is to identify the parts of the heuristic where some step is not based on theoretically provable facts, but on ad-hoc or statistical decisions. The algorithm is then modified such that these decisions can be controlled via control values. We present some general cases of how to introduce CVs into construction heuristics. Our test application described in Section 4 will introduce a CV for most of the types below. The following list is of course by no means complete, but the list’s items are meant as examples of a general design pattern.

**Selection Decisions:** When an algorithm has to choose between two possibilities, or select a subset of elements, we can introduce a simple CV  $c \in [0, 1] \subset \mathbb{R}$ . Introducing randomization, we can interpret  $c$  as the probability of choosing one out of two possibilities, or as the probability for an element to belong to the chosen subset. If the base set  $S$  for the latter problem is ordered, we can also interpret  $c$  directly as the selection ratio, without any randomization: we simply select the first  $\lceil c \cdot |S| \rceil$  (rounded) elements.

**Balancing Decisions:** In many applications, there are subproblems which have to optimize multiple criteria at once and where it is unclear which of the different measures is most important for the overall optimization goal. We can introduce a CV  $c \in [0, 1] \subset \mathbb{R}$  to balance two such measures  $x$  and  $y$  and obtain a balanced value, e.g., by  $z := c \cdot x + (1 - c) \cdot y$ . Note that, if applicable, the CV  $c$  has not be used directly in a linear balancing computation, but we can use any suitable balancing function. Another balancing problem can occur when the input is partitioned in sets of different types, e.g., weights

on nodes and edges of a graph. It might not be clear how much influence each set has for the overall optimization goal. This is, e.g., the case in our test application and is resolved via the *balance* CVs, see Section 4.

**Ordering Decisions:** Assume an algorithm where there is a sequence of steps and we do not know beforehand, which order will lead to the best solution. Traditional algorithms may choose some arbitrary order, often being the natural order arising from the order in which the input was given; other algorithms may randomize the order on purpose. Often, one has a certain guess, which order may be sensible, and sorts the steps accordingly. We can introduce a CV  $c \in [0, 1] \subset \mathbb{R}$  to steer this behaviour in two different ways:

*Bi-ordered:* Let there be two competing orderings  $o_1$  and  $o_2$  under consideration. Interpreting  $c$  as the probability for selecting  $o_1$ , we can randomly choose the first, yet unselected element from one of these orderings. We can simulate a gradual randomization, by choosing  $o_2$  as a random ordering.

*Random Range:* When mixing a potentially sensible ordering  $o_1$  with a random ordering, we can interpret  $c$  as a ratio of elements. Let  $n$  be the number of ordered elements. For each step, we choose randomly from the first  $m = \lceil c \cdot (n - 1) + 1 \rceil$  elements induced by  $o_1$ . While  $c = 0$  corresponds to the ordering  $o_1$ ,  $c = 1$  resembles a purely random order.

**Threshold Decisions:** Sometimes a heuristic step requires the identification of certain critical values, patterns, graph structures, etc. A CV can be used as a threshold to determine when a search pattern is considered to be identified. E.g., a simple threshold decision is the classification of fractional values into two groups of small and large values, respectively. One has to be careful with CVs in the context of identifying termination criteria of loops, etc., as a CV is purely for optimizing the solution quality, and not for optimizing the computation time: a CV may be suitable if there is no obvious connection between the number of iterations and the overall optimization goal. But if a CV would purely decide how often a loop, which iteratively improves the final solution, is run, a larger iteration count would always be superior to an earlier termination.

**Discrete Decisions:** There can be situations when augmenting a specific heuristic step with numerical values seems not reasonable. There may be intricately discrete decisions to make. Our approach is stable enough to allow integer CVs, although it comes with the usual problems known from optimizing integer parameters via an evolutionary algorithm, see below.

**Natural CVs:** Some construction heuristics already offer numerical parameters, e.g., for coarsening the input instance. We call these *natural CVs* and can directly use them within our optimization framework. Sometimes subproblems are solved via a PTAS, i.e., a polynomial approximation algorithm which approximates the solution within a choosable bound  $\epsilon$ . If it is not clear that an optimal solution for the subproblem will lead to an optimal solution for the whole problem, this  $\epsilon$  can also be seen as a natural CV already in the original algorithm.

Note that the selection of the CVs not only changes the behaviour of the construction heuristic, but also directly influences the evolutionary algorithm and the applicability of the overall algorithm, due to time constraints: if we have many CVs, the EA will require more generations to find a near-optimal CV instance. If many CVs introduce statistical noise due to randomization techniques, the evolutionary algorithm will require more calls to the augmented heuristic to assess the quality of a single CV instance. Hence, we suggest to drop a CV  $c$  when experiments show a clearly winning setting for  $c$  or when it becomes clear that  $c$  has no measurable influence at all.

### 3.1 Why Using Evolutionary Algorithms for CV Adjustment?

EAs are known as general-purpose direct optimization algorithms which only require a quality criterion (*fitness*) but no other additional data, e.g., on the gradient. They provide generic search operators for virtually every canonic representation, e.g., boolean, ordinal discrete, nominal discrete, and real-valued variables. As they are able to deal with mixed representations, they allow a ‘natural’ problem formulation and are therefore easy to apply. Emmerich et al. [5] give an example for a mixed integer problem and summarize the search operators for the basic variable types. Population-based EAs are by design a compromise between global and local search. If no good starting point is known, an EA usually starts with a small random sample and learns an internal model from the feedback obtained by subsequent search steps.

However, there are cases where a naïve EA does not perform well, namely on approximately random functions, and on simple unimodal functions. In the first case, there is no structure that could be learned, and in the second case, more specialized algorithms, e.g., quasi-Newton methods, are much faster. We assume that the CV optimization problem is neither of them and thus EAs are applicable. Furthermore, some of the previously described CV types introduce non-determinism into the target function evaluation, which can be seen as ‘noise’ from the viewpoint of the optimization algorithm. Recent investigations on noisy functions report that EAs still work reasonably well under such conditions [1].

Summarizing, it makes sense to apply an EA, not only because it can be easily done, but also since probably stronger analytical methods are unavailable as virtually nothing is known about the CV-induced search space topology. The EA shall perform as least as good as random search; if the EA is able to detect some exploitable structure, it will perform even better.

In our test application, we establish the adaptation of the CVs via a naïve  $(\mu, \lambda)$  *evolution strategy* (ES) as described in [2]. According to the CV list given in Section 4, we have six real-valued and two nominal discrete optimization variables and thus a mixed genome. While this does not pose a problem for recombination which is performed as a 2-point crossover, the mutation operator requires special attention: we employ a commonly used normal distributed mutation for the real-valued variables and conditioned random selection for the nominal discrete variables. For the random selection, a mutation probability is

used instead of the mutation strength, and if a change is determined, the new value is chosen randomly from all other allowed values but the current one.

Mutation step sizes and mutation probabilities, respectively, are controlled by self-adaptation, thereby learning strategy parameters from successful steps. As the domain sizes of the six real-valued variables are very similar—either 1 or 2, see below—we resort to a single strategy parameter, the mutation strength  $\sigma$ . It is varied by means of a learning rate  $\tau$ , which also applies to the second strategy parameter  $\sigma_{dis}$ , that resembles the mutation rate for the two nominal discrete variables. The chosen values for population size, offspring number, initial step sizes, minimum and maximum step sizes, and learning rate are given below. The last four have also been used for  $\sigma_{dis}$ . Currently, they have not been verified systematically as this would require too much computation power; instead, they have been tested against some representative alternative parameter sets.

$\mu$	$\lambda$	$\sigma_{init}$	$\sigma_{min}$	$\sigma_{max}$	$\tau$
15	100	0.3	0.001	0.5	0.2

## 4 The Network Problem and A Corresponding Heuristic

Formally, we can describe the 2-root-connected Prize-Collecting Steiner Network Problem (2RPCSN) as follows: We are given an undirected graph  $G = (V, E)$ , a root node  $r \in V$ , a set of customer nodes  $C = C_1 \dot{\cup} C_2 \subset V$ , a prize function  $p : C \rightarrow \mathbb{R}^+$  for the customers, and a cost function  $c : E \rightarrow \mathbb{R}^+$  on the edges. We ask for a subgraph  $N = (V_N, E_N)$  of  $G$  with  $r \in V_N$  which minimizes  $\sum_{e \in E_N} c(e) - \sum_{p \in C \cap V_N} p(v)$  and satisfies the following connectivity property: for every node  $v \in C_k \cap V_N$ ,  $k \in \{1, 2\}$ ,  $N$  contains at least  $k$  node-disjoint paths connecting  $v$  to  $r$ . Informally, this means that we look for the most cost-efficient network, where we can choose which customers we want to connect, based on the connection costs  $c$  and their estimated profits  $p$ . For certain customers  $C_2$  we can only decide whether we want to connect them via two disjoint connection paths, or not at all.

**Overview.** We sketch the heuristic presented in [3], and explain its modifications in order to support certain control values: The heuristic starts by choosing subsets  $C_i^* \subseteq C_i$  ( $i = 1, 2$ ) of customers to be included in the solution. Based on this customer set  $C^* := C_1^* \cup C_2^*$  we heuristically compute a minimal Steiner tree  $T$  as a basis for the next steps. Note that  $T$  resembles a feasible solution if  $C_2^* = \emptyset$ . We then extend this tree by adding additional paths for the  $C_2^*$  customers, and obtain a feasible solution  $S$ . Finally, we perform some postprocessing to shrink  $S$  by removing certain nodes and edges without losing feasibility.

**Choose Customers and Construct  $T$ .** The heuristic is especially suited for the use within a Branch-and-Bound algorithm. Thereby intermediate fractional solutions are used to infer suitable sets  $C_1^*$  and  $C_2^*$ . In [3], the setting  $C_1^* = C_1$

and  $C_2^* = C_2$  was suggested for the use as a constructive start heuristic. We introduce our first two real-valued control values:

$$\text{choose}_1, \text{choose}_2 \in [0, 1] \subset \mathbb{R} \quad (1)$$

The CV  $\text{choose}_k$ ,  $k \in \{1, 2\}$ , determines the fraction of  $C_k$  customers, which is chosen for  $C_k^*$ . I.e., having an ordered set of  $C_k$  customers, we choose the first  $\lceil \text{choose}_k \cdot |C_k| \rceil$  vertices as the set  $C_k^*$ . Of course the ordering of  $C_k$  is crucial. This leads to two additional integer CVs. Note that the differences between real-valued and integer CVs are quite interesting, as we will see in Section 6.

$$\text{sort}_1, \text{sort}_2 \in \{0, 1, 2, 3\} \subset \mathbb{N} \quad (2)$$

The interpretation of the image set of  $\text{sort}_k$ ,  $k \in \{1, 2\}$ , is as follows:

- 0:**  $C_k$  is permuted randomly.
- 1:**  $C_k$  is sorted by decreasing profit  $p(v)$  of  $v \in C_k$ .
- 2:**  $C_k$  is sorted by increasing cost  $z(r \rightarrow v)$  of the shortest path from  $r$  to  $v \in C_k$ .
- 3:**  $C_k$  is sorted by decreasing efficiency, which is defined as the ratio  $\frac{p(v)}{z(r \rightarrow v)}$ .

After choosing  $C^*$ , we apply the Minimum Steiner Tree heuristic by Mehlhorn [6] to compute the tree  $T$ . This algorithm requires the shortest paths between all pairs of nodes of  $C^* \cup \{r\}$ . In [3], these shortest paths are computed using only the edge costs  $c$ . This approach does not take into account that a path might include a customer node which offers some profit and therefore renders the costly path cheap or even profitable.

Based on the observation that an inner node of a path is incident to exactly two edges of the path we would like to use the modified edge costs  $c'(u, w) := c(u, w) - \frac{1}{2}(p(u) + p(w))$ . However,  $c'$  cannot be easily used for shortest paths computations, as its values may be negative and thus induce negative cycles. On the other hand, using  $c'' := \max\{c', 0\}$  as a cost function is problematic as well: all edges incident to a profitable vertex will have the same cost of 0, and the algorithm is likely to choose a random edge instead of making a well-grounded decision. We therefore use the cost function  $c^*(u, w) := \max\{c(u, w) - \alpha \frac{1}{2}(p(u) + p(w)), 0\}$ , using the real-valued  $\alpha \in [0, 1]$  to balance the influence of the customer profits. As initial experiments showed,  $\alpha = 0.5$  and  $\alpha = 1$  lead to very similar cost functions, since in both cases edge costs are often truncated to 0. Hence we do not choose  $\alpha$  as a CV directly, but use the control value  $\text{balance}_T$  which is then transformed into a suitable value for  $\alpha$ :

$$\text{balance}_T \in [0, 1] \subset \mathbb{R} \quad (3)$$

To make up for the aforementioned skew in the  $\alpha$  values, we use  $\alpha = \text{balance}_T^3$ . Note that we use the resulting cost function not only within the minimum Steiner tree heuristic, but also for the sorting of  $C_k$  if  $\text{sort}_k \geq 2$ .

**Assure 2-connectivity.** The idea is to iteratively extend  $T$  by adding shortest  $(v \rightarrow r)$ -paths for the customer vertices  $v \in C_2^*$  for which the 2-connectivity requirement is not satisfied yet. Let  $v$  be such a customer and let  $P_v$  the inner nodes of the unique  $(r \rightarrow v)$ -path in the original  $T$ . To find a disjoint second path between  $r$  to  $v$ , we look at the graph  $G \setminus P_v$ , i.e., the graph obtained by removing all inner nodes of the original tree path, and all their incident edges. We set the costs of the edges which are already in the solution to 0, and use the cost function  $c^*$  for the other edges. We then apply Dijkstra's shortest path algorithm to identify a path between  $v$  and  $r$ , and add the resulting path to our solution. This step uses its own control value

$$balance_S \in [0, 1] \subset \mathbb{R} \quad (4)$$

to skew the cost function  $c^*$ , analogously to  $balance_T$ . This augmentation is performed for all nodes  $v \in C_2^*$  which do not contain any further  $C_2$  customers in their subtrees of  $T$ ; let  $L$  be the list of these  $C_2^*$  nodes. Note that 2-connecting  $v$  results in proper 2-connectedness for all nodes  $w \in P_v$ .

The quality of the resulting subgraph  $S$  can depend on the order of the nodes in  $L$ . An intuitive idea, as proposed in [3], is to sort these nodes  $v$  in decreasing order by the number of  $C_2$  customers in  $P_v$ . If this number is identical for different nodes, we sort those by decreasing overall number of elements in  $P_v$ . However, experiments show that choosing another order can change and improve the solution quality. Hence, we introduce the real-valued control value

$$order^+ \in [-1, 1] \subset \mathbb{R}. \quad (5)$$

Let  $L_r$  and  $L_o$  be copies of the list  $L$ . While  $L_r$  is a random permutation, the list  $L_o$  is sorted as described above. If  $order^+$  is negative, we will reverse  $L_o$ . The absolute value of  $order^+$  indicates the probability of taking the next node from  $L_o$ ; otherwise the next node from  $L_r$  is chosen.

**Shrinking.** In general, the subgraph  $S$  obtained by the previous steps can be further optimized by removing some nodes and edges from  $S$  without losing feasibility.

We know that due to the construction,  $S$  consists of one or more non-trivial 2-connected components, which have only the root node  $r$  in common. All other components of the graph form trees, which are attached to some 2-connected component. Having this decomposition in mind we can optimize  $S$  in two steps:

As described in [10], the rooted price-collecting Steiner tree problem can be solved to optimality in linear time, when applied to trees by dynamic programming. We use this algorithm to optimize all attached trees, using the attachment node as its root. For the next step, these root nodes are considered to be  $C_1$  customers with corresponding prizes. These optimizations are optimal and independent of each other, and hence there is no need for any CVs.

In the final step we try to optimize each non-trivial 2-connected component  $B$  of  $S$ . Such components may contain redundant edges, i.e., their removal does

not break the feasibility of the solution. For every  $B$  we compute its *core graph*  $\tilde{B}$ . Thereby, every chain of edges only containing nodes  $v \in V \setminus (C_2 \cup \{r\})$  is replaced by a single edge. We then successively consider the edges  $e$  of  $\tilde{B}$ : it can be removed from  $\tilde{B}$  if all connectivity requirements are still satisfied for  $\tilde{B} - e$ . In this case, the corresponding path  $P_e$  will be broken apart and partially removed. The optimization within the path  $P_e$  can be done optimally and efficiently.

Similar to the steps before, the order in which the core edges are considered and removed from  $\tilde{B}$  may be crucial for the solution quality. An intuitive sorting criterion is to use the weight of the core edge, i.e., the sum of the edges in  $P_e$  minus the sum of the profits of inner nodes of  $P_e$ . We parameterize this order by a real-valued control value:

$$order^- \in [-1, 1] \subset \mathbb{R} \quad (6)$$

Again, we have a randomly permuted and a sorted list of core edges. A positive  $order^-$  leads to a decreasing order, a negative value leads to an increasing order. The absolute value of  $order^-$  gives the probability of choosing the next core edge from the ordered list, instead of from the randomly permuted list.

Overall, we can give the CV instance which resembles the behaviour of the original heuristic. Technically, the original heuristic does not use any sorting to obtain  $C^*$ : since the *choose* CVs are 1, these parameters have no influence.

<i>choose</i> <sub>1</sub>	<i>choose</i> <sub>2</sub>	<i>sort</i> <sub>1</sub>	<i>sort</i> <sub>2</sub>	<i>balance</i> <sub>T</sub>	<i>balance</i> <sub>S</sub>	<i>order</i> <sup>+</sup>	<i>order</i> <sup>-</sup>
1	1	any	any	0	0	1	1

## 5 Experimental Assessment of Performance Improvement

We test our hybrid algorithm on three different test sets, which were used and described in [3]. The groups  $K$  and  $P$  consist of graphs each containing 400 nodes and a high percentage of customer nodes, allowing us to better analyze the influence of the *choose* CVs. In particular, the set  $K$  consists of random geometric instances which were designed to have a structure similar to street maps. All of these instances could be solved to optimality with the Branch-and-Cut approach presented in [3]. These instances are chosen in order to evaluate the absolute solution quality achieved by our algorithm. Additionally, we test against the instance set  $ClgM^+$ . These instances are real-world graphs based on a street map of Cologne. The graphs have 1757 nodes and relatively small number of customers (up to 15–20  $C_1$  and  $C_2$  customers, respectively). As these instances are hard for the ILP-approach—only one instance could be solved to optimality within 2 hours—we are in particular interested in improving the heuristic solutions and in analyzing our approach for its applicability within the mixed scenario (cf. Section 1).

The following two experiments investigate the question raised as Aim 1 in Section 2, i.e., does the hybrid approach improve the solution quality when applied to a single instance or to a group of instances.

### 5.1 Experiment 1: Does the hybrid approach improve the performance over the default heuristic on single problem instances?

**Pre-experimental planning.** By initial experiments, we found out that the run-length of the EA should be limited to at most 10000 evaluations for two reasons: firstly, the absolute running times, especially for small problem instances, are getting too large, and secondly, the optimization typically stagnates after ca. 5000–6000 evaluations. We also tried different numbers of repeats for evaluating a single CV instance, to smooth the noise introduced by the randomization. It turned out that 5–10 repeats seem to be a good compromise between speed and fitness value stability; we perform 10 repeats for the following experiment.

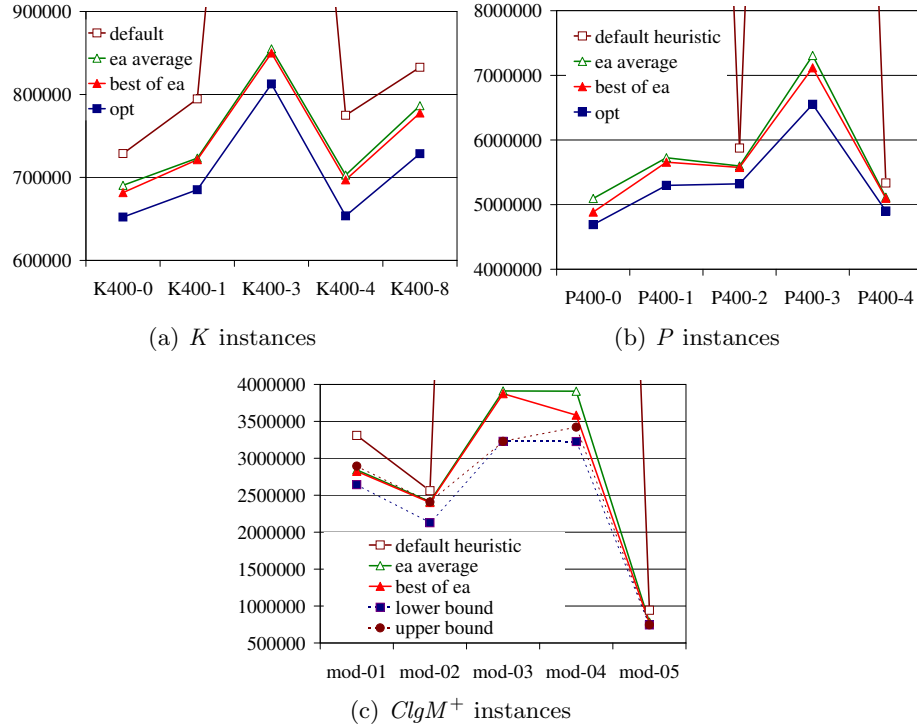
**Setup.** To test the performance of our hybrid algorithm on single problem instances, we perform 10 repeats for every problem instance and compute the best and the average objective values over these 10 independent solutions.

**Task.** We are interested in the improvement of the solutions—both the best and the average value—compared to the default heuristic, as described in Section 4. In order to state that the configured hybrid heuristic performs better than the default heuristic, we require that a Wilcoxon rank sum test is significant at the 5% level at least for the best of the 10 resulting CV instances for each problem instance, over 100 validation runs of the heuristic.

We are also interested in the solutions’ quality with respect to the known optima [3]. We try to identify successful and unsuccessful steps within the heuristic, which is useful when applying the heuristic to larger problems. Although we do not know the optima for any but one instance of the  $ClgM^+$  group, we have lower and upper bounds stemming from two hours of Branch-and-Cut computation: they help to estimate the quality of our solutions in these cases.

**Results & Observations.** See Figure 1 for visualizations of our experimental results. For various instances, the default heuristic is not able to find any feasible solution other than the trivial *root-solution*, i.e., no customers are selected. Such root-solutions are 115–689% away from the corresponding optimum. In contrast, our hybrid algorithm always computes at least one non-trivial feasible solution for all test instances. For the  $K$  and  $P$  instances, these solutions are 4–8% away from the optimum, on average; the gap is 11–15% for the non-trivial solutions of the default heuristic. The situation is similar for  $ClgM^+$ , where the default heuristic can only find the root-solution for two instances. Thereby the improvement by the hybrid algorithm is about 90%. For the other  $ClgM^+$  instances the improvement is between 5 and 18%.

**Discussion.** The obtained results show that our approach reliably leads to good approximations, even when the default heuristic fails completely. By tuning certain parts it is therefore possible to obtain a better configured heuristic. The applied statistical tests confirm a significant improvement, as all p-values except for K400-8 are below  $10^{-5}$ . In this special case, a significant improvement is still detected, but slightly below the required level (p-value 0.145). We also see that the difference between the best and the average performance of 10 adapted heuristics is rather small. Hence running the EA one or few times seems to be sufficient to exploit most of the available performance potential.



**Fig. 1.** Comparing the behaviour of the default heuristic and the hybrid algorithm on the single test instances

**Additional Validation.** In order to verify that the EA performs consistently at least as well as the random search and often better, we applied both methods to three problem instances for which the default heuristic fails completely, namely K400-3, P400-3, and  $ClgM^+$ -03. The comparison of the mean best values of 20 repeats shows that indeed, the EA always achieves the same or a better fitness level. However, due to large standard deviations, Wilcoxon rank sum tests do not get significant at the 5% level (p-values 0.47, 0.91, and 0.08, respectively). Nonetheless, we can see a wide performance gap for the largest instance  $ClgM^+$ -03. This seems to be due to the fact that there is some potential for further improvement, whereby for the two smaller instances, both methods already operate near the optimum.

## 5.2 Experiment 2: Does the hybrid approach lead to improved performance on problem instance sets?

**Pre-experimental planning.** Experiments show that 10 evaluations for each problem of a set would severely slow down the EA. As computing less generations seemed not sensible, we resort to a ‘weighted optimistic average’ scheme that uses only 2 evaluations per instance and averages over the best of these two. This

is necessary to avoid distorting effects of bad outliers. The average is weighted relative to the best fitness obtained on each problem during the computation of the first generation; we do this to avoid scaling effects stemming from different magnitudes of the instances’ objective values.

**Task.** We accept the adapted heuristic as dominant to the default heuristic, if the resulting best CV instance leads at least to the same performance as the latter for every single problem instance of the test set. Additionally, we require the same for additional instances of the same type, which are used during the EA optimization. Overall, we require that statistical testing reveals a significant advantage on at least half of the tested instances per group.

**Setup.** We only consider the  $K$  and  $ClgM^+$  problem sets. During the validation phase of the best found CV instance, we add 2 previously unused  $K$  instances and one  $ClgM^+$  instance, respectively.

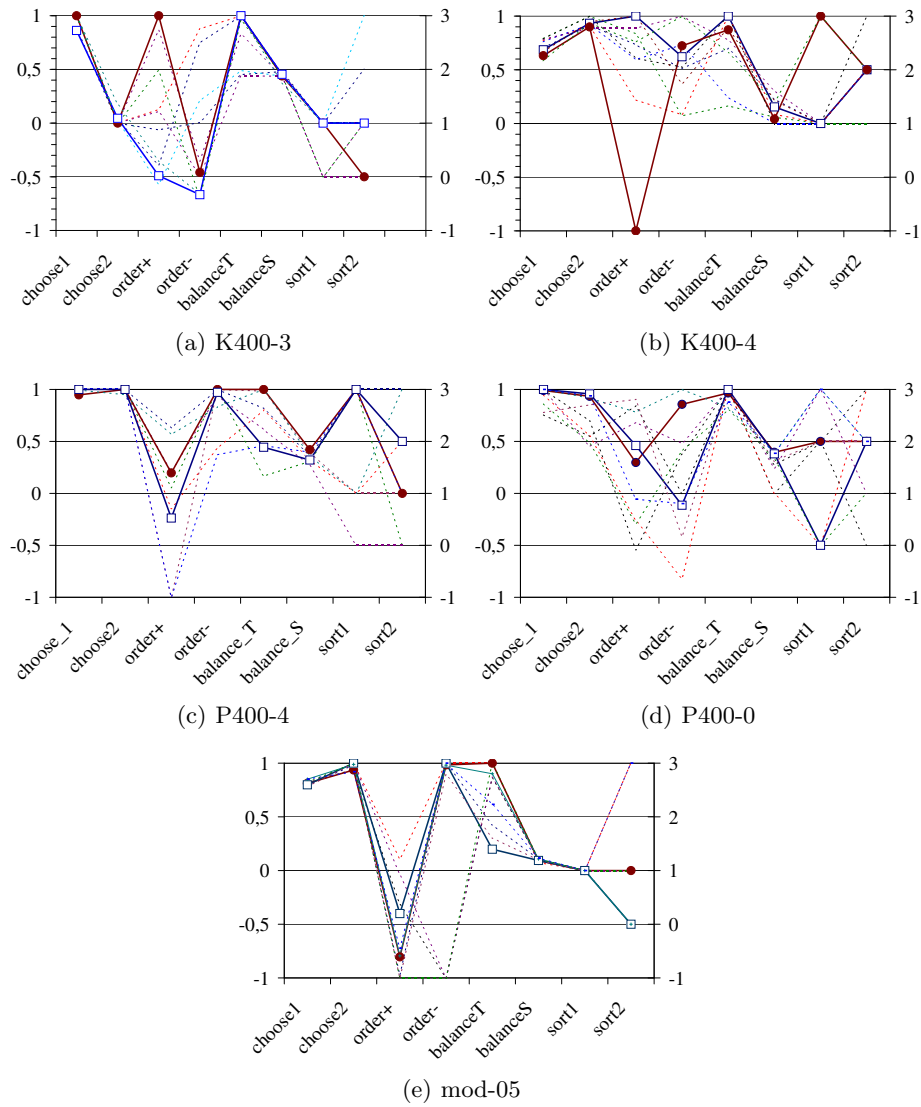
**Results & Observations.** The validation of the obtained best and average CV instances revealed that training with a full set instead of a single problem instance leads to clearly worse performance than reported in Experiment 1. The average CV instance leads to worse solutions than the default heuristic for several problem instances. The best CV instance however appears to be better than the default heuristic in most cases.

**Discussion.** The Wilcoxon rank sum test confirms significant performance differences between the best obtained CV instance and the default heuristic, in all cases except two. The two critical problem instances are  $ClgM^+-4$  with a p-value of 0.036, and  $K400-3$  with a p-value of 1. Whereas in the first case, the attained advantage so small that it may stem from lucky sampling, we cannot report any improvement at all for the second case. However, for all other problem instances, we obtain significant differences with  $p < 10^{-5}$ , i.e., the CV instance leads to improved performance even on the instances that have not been used for training. We can therefore state that learning a suitable CV instance for a set of problem instances is successful, although the performance gain is not as large as for single problem instances.

## 6 Experimental Analysis of Adaptability

This section investigates the changes induced to the heuristic by automatically adapting it to the given instance or instance set. This corresponds to Aim 2 formulated in Section 2. The results documented in Section 5 clearly suggest that for a given instance, certain CV settings may be especially useful. Hence, for each instance we analyze the CV instances computed by the hybrid algorithm. Figure 2 shows the results for some representative problem instances.

Before analyzing each single CV, we may observe that there are certain evident dependencies between the CVs. E.g., if a *choose* CV is set to 0 or 1, the corresponding *sort* CV does not have any importance. Analogously,  $choose_2 = 0$  makes both *order* CVs insignificant. We see such an effect in Figure 2(a). We know that it is difficult for our EA to handle nominal discrete CVs as they do



**Fig. 2.** CV instances for representative problem instances. The two solid lines identify the CV instances for the best solution (solid circles) and for the solution which has a value nearest to the average solution value (empty squares).

not provide any gradient information but simply have to be tried out. However, additional experiments show that their value in optimal CV instances is in fact meaningful: by changing a *sort* CV in a given CV instance, the solution quality usually decreases dramatically. For most instances where  $choose_1 \neq 1$ , we observe that the optimal CV instances have  $sort_1$  set to either 1 or 3. We can deduce that sorting  $C_1$  customers by their weight or by their efficiency is superior to random sorting or sorting by distance.

We can obtain much insight about the problem instance K400-3 by analyzing its CVs:  $choose_2$  is very small, indicating that it is either not worthwhile to select any  $C_2$  customer or it is not possible to connect them feasibly. The fact that the default heuristic cannot find any feasible solution suggests the latter.

The *balance* CVs are probably the most interesting control values, as we had no idea for a suitable setting, prior to the experiments. By interpreting their settings, we obtain some insight into the involved construction heuristic steps. These CVs specify how much the node profits should be considered for the shortest path computations. Most surprisingly, it turns out that there is a significant difference between  $balance_S$  and  $balance_T$ . While the former is near to 1 for most problem instances, the latter usually is between 0 and 0.5. Although this fact may be quite surprising at first sight, there is a natural interpretation. Remember that a large *balance* value will result in paths which contain more profitable edges, but has the drawback that the edges incident to a profitable node become cost-wise indistinguishable. The CV  $balance_T$  is used when computing shortest paths in the whole graph. The density of the node customers on a single path in the full graph is rather small, and choosing the perfect incident edges is not as important as the benefit of attaching a profitable node. This is not the case for  $balance_S$ : when using this CV we look for rather short and local connection path, whereby it is more important to choose optimal edges.

The diagrams also show that the *order* CVs do not clearly influence the solution quality. Although some problem instances indeed exhibit certain tendencies, these cannot be generalized. Therefore, we can set these CVs to default values, if we have no knowledge about the specific problem instance we are dealing with.

## 7 Conclusions

We introduced a general *CV hybridization* scheme, which allows to utilize numerical-based evolutionary algorithms for combinatorial problems. We showed that our hybridization approach works well for our test application, and it therefore seems reasonable to suggest to try this approach for different combinatorial problems and heuristics. We can also conclude that CV hybridization should focus on real-valued CVs, as nominal discrete CVs pose difficulties in general.

We assume that an analytical optimization of the parameters for the EA, i.e., population size, offspring number, etc., might lead to even further improvements.

## References

1. D. V. Arnold and H.-G. Beyer. A comparison of evolution strategies with other direct search methods in the presence of noise. *Computational Optimization and Applications*, 24(1):135–159, 2003.
2. H.-G. Beyer and H.-P. Schwefel. Evolution strategies—A comprehensive introduction. *Natural Computing*, 1:3–52, 2002.
3. M. Chimani, M. Kandyba, and P. Mutzel. A new ILP formulation for a 2-connected prize collecting steiner network problem. In *Proc. ESA '07*, 2007. to appear.

4. K. de Jong. *Evolutionary Computation: A Unified Approach*. MIT Press, Cambridge, MA, 2006.
5. M. Emmerich, M. Grötzner, B. Groß, and M. Schütz. Mixed-integer evolution strategy for chemical plant optimization with simulators. In *Evolutionary Design and Manufacture – Sel. papers ACDM'00*, pages 55–67. Springer, 2000.
6. K. Mehlhorn. A faster approximation for the steiner problem in graphs. *Information Processing Letters*, 27:125–128, 1988.
7. El-G. Talbi. A taxonomy of hybrid metaheuristics. *J. Heuristics*, 8(5):541–564, 2002.
8. D. Wagner, G. R. Raidl, U. Pferschy, P. Mutzel, and P. Bachhiesl. A multi-commodity flow approach for the design of the last mile in real-world fiber optic networks. In *Proc. GOR '06*. Springer, 2006.
9. D. Wagner, G. R. Raidl, U. Pferschy, P. Mutzel, and P. Bachhiesl. A directed cut for the design of the last mile in real-world fiber optic networks. In *Proc. INOC '07*, pages 103/1–6, 2007.
10. L. A. Wolsey. *Integer Programming*. Wiley-Interscience, 1998.