European Journal of Operational Research xxx (xxxx) xxx

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/eor

Discrete Optimization

Hybridizing Carousel Greedy and Kernel Search: A new approach for the maximum flow problem with conflict constraints

F. Carrabs [a], R. Cerulli [a], R. Mansini [b], D. Serra [a], C. Sorgente [a],*

[a] Department of Mathematics, University of Salerno, Fisciano, 84084, Italy
[b] Department of Information Engineering, University of Brescia, Brescia, 25123, Italy

ARTICLE INFO

Keywords:
(O) heuristics
Conflict constraints
Kernel Search
Carousel Greedy
Kernousel

ABSTRACT

This work addresses a variant of the maximum flow problem where specific pairs of arcs are not allowed to carry positive flow simultaneously. Such restrictions are known in the literature as *negative disjunctive constraints* or *conflict constraints*. The problem is known to be strongly NP-hard and several exact approaches have been proposed in the literature. In this paper, we present a heuristic algorithm for the problem, based on two different approaches: Carousel Greedy and Kernel Search. These two approaches are merged to obtain a fast and effective matheuristic, named Kernousel. In particular, the computational results reveal that exploiting the information gathered by the Carousel Greedy to build the set of most promising variables (the *kernel set*), makes the Kernel Search more effective. To validate the performance of the new hybrid method, we compare it with the two components running individually. Results are also evaluated against the best-known solutions available in the literature for the problem. The new hybrid method provides 15 new best-known values on benchmark instances.

1. Introduction

The study of maximum flow problems in network optimization has a long history, driven by their critical applications in transportation and communication systems. In this paper, we address the problem of finding a maximum flow from a source to a sink in a network, while complying with classical capacity constraints and preventing positive flow from being jointly carried out by the arcs of any conflicting pair. To the best of our knowledge, the problem was first studied by Pferschy and Schauer (2013), who separated positive from negative disjunctive constraints showing the NP-hardness of the resulting variants. In addition, Şuvak et al. (2020) investigated the use of negative disjunctive constraints to model conflicts between arc pairs, and provided some mixed-integer linear programming formulations and exact solutions for the problem. The maximum flow problem with conflicts (MFPC) belongs to the class of optimization problems with conflict constraints. Among the existing optimization models dealing with conflicting decisions, we recall the minimum spanning tree problem with conflicting edge pairs (Carrabs et al., 2019, 2021; Carrabs & Gaudioso, 2021), the knapsack problem with conflict constraints (Coniglio et al., 2021; Li et al., 2021; Pferschy & Schauer, 2009), the bin packing problem with conflicts (Capua et al., 2018; Sadykov & Vanderbeck, 2013), the set covering problem with conflicts (Carrabs et al., 2024; Saffari & Fathi, 2022), the shortest path problem with conflicts (Cerulli et al., 2023; Darmann et al., 2011), the multi-vehicle traveling purchaser problem with pairwise incompatibility constraints (Gendreau et al., 2016), the directed profitable rural postman problem with incompatibility constraints (Colombi et al., 2017), and the minimum cost perfect matching with conflict pair constraints (Öncan et al., 2013).

In this paper, we discuss several heuristic approaches to address the maximum flow problem with conflicts. First, we modify the well-known *augmenting path algorithm* (Ahuja et al., 1993), commonly used to solve the maximum flow problem. Based on a greedy criterion, the provided variant can quickly identify a feasible solution dealing with conflicting arcs. Then, we present a more effective algorithm, obtained by embedding the introduced greedy algorithm in a Carousel Greedy framework (Cerrone et al., 2017). Given a feasible solution generated by a greedy algorithm, the core concept behind the Carousel Greedy approach is to enhance such a solution by replacing the *older* choices made by the algorithm with new ones that enable the creation of a new feasible solution. As the decisions made in the initial and final phases of the starting greedy algorithm are the most constrained ones, this process naturally enhances such a procedure, by actually extending the exploration phase, with a reduced computational overhead.

* Corresponding author.
E-mail addresses: fcarrabs@unisa.it (F. Carrabs), raffaele@unisa.it (R. Cerulli), renata.mansini@unibs.it (R. Mansini), dserra@unisa.it (D. Serra), csorgente@unisa.it (C. Sorgente).

https://doi.org/10.1016/j.ejor.2025.02.006
Received 7 February 2024; Accepted 5 February 2025
Available online 13 February 2025

Several well-known combinatorial optimization problems, like the minimum label spanning tree, the minimum vertex cover, the maximum independent set, and the minimum weight vertex cover problems served as the first case study for the Carousel Greedy framework (Cerrone et al., 2017). By now, the resulting scheme has been applied to a variety of problems, including distribution problems (Cerrone et al., 2018), the strong generalized minimum label spanning tree problem (Cerrone et al., 2019), the knapsack problem with forfeits (Capobianco et al., 2022; Cerulli et al., 2020), the close-enough traveling salesman problem (Carrabs et al., 2020), and the wireless sensor networks problems (Carrabs et al., 2017; Cerulli et al., 2022).

To further address the problem, we design an algorithm based on the Kernel Search scheme (Angelelli et al., 2010, 2012), a matheuristic framework that relies on the resolution of a sequence of restricted mixed-integer linear programming problems based on subsets of variables duly selected while setting the remaining ones to zero. The method identifies an initial set of promising variables, namely the *kernel set*, based on the information gathered by solving the continuous relaxation of the problem. A variable is deemed promising when there is a high probability that it will take a positive value in the final optimal integer solution. The remaining variables are partitioned into groups called *buckets* and considered one after the other. Each restricted problem is constructed by considering the variables belonging to the kernel set plus the ones in the current bucket. The goal achieved with the solution of each restricted problem is two-fold: on one side the obtained feasible solution possibly improves the incumbent one, on the other side it contributes to the update of the kernel set by adding new variables (the ones selected in the current bucket when solving the corresponding restricted problem). In such a scheme, the more meaningful the continuous relaxation, the more effective the Kernel Search algorithm.

When first introduced, the Kernel Search scheme was designed to address a complex portfolio optimization problem (Angelelli et al., 2012). It was also successfully applied to the strongly NP-hard multidimensional knapsack problem (Angelelli et al., 2010). Since then, the Kernel Search framework has been adopted to solve a variety of optimization problems, including the multidimensional multiple-choice knapsack problem (Lamanna et al., 2022), a complex multi-objective personnel scheduling (Mansini et al., 2023), the index tracking problem (Guastaroba & Speranza, 2012), lot-sizing problems (Carvalho & Nascimento, 2018; Kirschstein & Meisel, 2019), as well as facility location problems (Filippi et al., 2021; Guastaroba & Speranza, 2014; Mansini & Zanotti, 2022) and routing problems (Gobbi et al., 2023; Hanafi et al., 2020). Beyond the application to specific problems, the Kernel Search framework was also generalized to tackle the solution of general mixed integer linear programs (Guastaroba et al., 2017), used as a primal heuristic inside a Branch-and-Cut approach (Hanafi et al., 2020) and enhanced through a two-phase mechanism able to dynamically set the value of the main parameters according to the type of instance solved (Lamanna et al., 2022).

Finally, in this paper, we propose a novel hybrid approach, named *Kernousel*, consisting of a Kernel Search algorithm that incorporates the previously described Carousel Greedy algorithm.

Contributions. The main contributions of the paper are listed below:

- A non-trivial adaptation of the classical augmenting path algorithm is designed to generate conflict-free augmenting paths;
- A Carousel Greedy and a Kernel Search algorithm for the MFPC are developed;
- A new hybrid method, named Kernousel, merging Carousel Greedy and Kernel Search to obtain a novel more effective matheuristic, is presented. The new method is particularly well-suited for all those combinatorial problems where the Kernel Search has to struggle to identify promising variables since the optimal solution of the continuous relaxation does not provide any relevant information.
- A computational study is carried out on a set of benchmark instances to validate proposed methods. A comparison of our algorithms with the state-of-the-art solution approaches for the MFPC is also included.

The remaining part of the paper is organized as follows. In Section 2, we provide a formal definition of the problem and we report two formulations from the literature. Section 3 is devoted to the description of the solution algorithms implemented to solve the problem. In Section 4, we test the proposed methods comparing their results with those of a state-of-the-art algorithm on benchmark instances. Finally, conclusions are drawn in Section 5.

2. Problem description and formulations

The maximum flow problem with additional conflict constraints (MFPC) is defined on a directed graph $G = (V, A)$, with V denoting the set of nodes and A the set of arcs. The node set includes a source node s and a sink node t. A capacity vector $u \in \mathbb{N}_0^{|V|\times|V|}$ is provided, with u_{ij} indicating the maximum amount of flow that can be sent from node i to node j, $\forall\ i,j \in V$ and $u_{ij} = 0$ if $(i,j) \notin A$. Two arcs $(i,j),(k,\ell) \in A$, $(i,j) \neq (k,\ell)$, are *in conflict* if, in any feasible solution of MFPC, at most one of them can have positive flow. Let $\delta(i,j)$ be the set of arcs in conflict with the arc $(i,j) \in A$. The aim of the problem is to identify the maximum flow from s to t that satisfies both the conflict and the capacity constraints. Any feasible solution S to the problem can be represented, for simplicity, as a (possibly empty) sequence of n_S conflict-free flow augmenting paths $S = \langle P_1, P_2, \dots, P_{n_S}\rangle$. Indeed, a solution is not simply a sequence of paths but is represented by the flows assigned to each arc of the network, derived from the augmenting paths used. It is worth noting that the zero-flow solution is always feasible for the MFPC regardless of the graph and set of conflicts. Useful notation adopted throughout the paper can be found in the first part of Table A.7 in Appendix.

Fig. 1(a) shows a flow network in which the conflict-free arcs are depicted as black thin-line arrows, while arcs in conflict with each other are represented using thick-line arrows of the same color. This means that $\delta(s,a) = \delta(b,a) = \delta(c,t) = \delta(e,t) = \emptyset$ while $\delta(s,b) = \{(c,e),(d,t)\}$, $\delta(a,e) = \{(a,c),(b,d)\}$ and so on. In Fig. 1(b) we show the corresponding optimal solution $S = \langle P_1, P_2\rangle$ of MFPC, where $P_1 = \langle (s,a),(a,e),(e,t)\rangle$ and $P_2 = \langle (s,b),(b,c),(c,t)\rangle$ are two augmenting paths carrying 2 and 3 units of flow, respectively. The units of flows on P_1 and P_2 are set equal to the minimum capacity between the arcs belonging to them. To represent the flow sent through an arc we use the notation flow/capacity: for example, in Fig. 1(b) the value 2/3 on arc (s,a) indicates a flow of 2 units on an arc with capacity 3. Let $z(S)$ denote the value associated with the solution S, i.e., the amount of flow sent from s to t through the augmenting paths in S. In the example, the total flow is equal to 5 units.

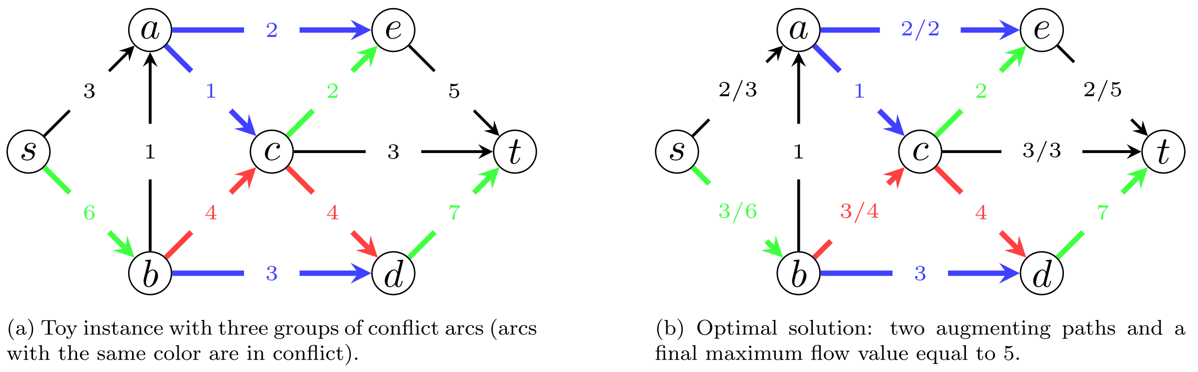
2.1. Mathematical models

In this section, we describe the two Mixed-Integer Linear Programming formulations for the MFPC as introduced in Şuvak et al. (2020), denoted as MFPC$_s$ and MFPC$_w$, respectively. The authors show in the same work that the polyhedron associated with the LP relaxation of MFPC$_w$ contains the one corresponding to the LP relaxation of MFPC$_s$. Thus, the LP relaxation of MFPC$_s$ produces better bounds, and then it is referred to as *strong formulation*, while MFPC$_w$ is named *weak formulation*. In the following, the two formulations are reported.

Formulation 1. *MFPC$_s$ (Strong Formulation)*

$$\max \mathcal{F} \tag{1a}$$

$$\text{s.t.} \sum_{(i,j)\in A} f_{ij} - \sum_{(j,i)\in A} f_{ji} = \begin{cases} \mathcal{F} & \text{if } i = s \\ 0 & \text{if } i \in V \setminus \{s,t\} \\ -\mathcal{F} & \text{if } i = t \end{cases} \qquad \forall\, i \in V \tag{1b}$$



a
e
s
c
t
b
d
2
3
1
2
5
1
3
6
4
4
7
3
2/2
2/3
1
2
2/5
1
3/3
3/6
3/4
4
7
3
(a) Toy instance with three groups of conflict arcs (arcs with the same color are in conflict).
(b) Optimal solution: two augmenting paths and a final maximum flow value equal to 5.

Fig. 1. An example of the MFPC problem. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$$x_{ij} + x_{k\ell} \le 1 \qquad \forall\,(i,j) \in A,\ (k,\ell) \in \delta(i,j) \quad \text{(1c)}$$

$$0 \le f_{ij} \le u_{ij}x_{ij} \qquad \forall\,(i,j) \in A \quad \text{(1d)}$$

$$x_{ij} \in \{0,1\} \qquad \forall\,(i,j) \in A \quad \text{(1e)}$$

$$\mathcal{F} \ge 0. \quad \text{(1f)}$$

Formulation 2. *MFPC$_w$ (Weak Formulation)*

$$\max \mathcal{F} \quad \text{(2a)}$$

$$\text{s.t.} \sum_{(i,j)\in A} f_{ij} - \sum_{(j,i)\in A} f_{ji} = \begin{cases} \mathcal{F} & \textit{if } i = s \\ 0 & \textit{if } i \in V \setminus \{s,t\} \\ -\mathcal{F} & \textit{if } i = t \end{cases} \qquad \forall\, i \in V \quad \text{(2b)}$$

$$|\delta(i,j)|x_{ij} + \sum_{(k,\ell)\in\delta(i,j)} x_{k\ell} \le |\delta(i,j)| \qquad \forall\,(i,j) \in A \quad \text{(2c)}$$

$$0 \le f_{ij} \le u_{ij}x_{ij} \qquad \forall\,(i,j) \in A \quad \text{(2d)}$$

$$x_{ij} \in \{0,1\} \qquad \forall\,(i,j) \in A \quad \text{(2e)}$$

$$\mathcal{F} \ge 0. \quad \text{(2f)}$$

Both formulations are based on the continuous variables $f \in \mathbb{R}_0^{+|A|}$, indicating the flow assigned to each arc, the continuous variable $\mathcal{F}$, representing the overall flow from s to t, and the binary auxiliary variables $x \in \{0,1\}^{|A|}$, where x_{ij} is equal to 1 if the related arc (i,j) carries some flow. Furthermore, both the formulations make use of the classical flow-balance equality constraints (1b) and (2b), as well as the classical flow capacity constraints, modified in order to correctly set the values of the auxiliary variables x to 1 if the flow on the related arcs is positive, (1d) and (2d) respectively.

The two formulations only differ in how they prevent conflict violations. In particular, constraints (1c) guarantee, for each pair of conflicting arcs (i,j) and (k,ℓ), that at most one of them has positive flow, while constraints (2c) impose that if arc $(i,j) \in A$ has a positive flow ($x_{ij} = 1$) then all arcs in $\delta(i,j)$ have flow equal to zero. We report both formulations as they are both used in our hybrid approach as explained in Section 3.3. Given any feasible solution $(\hat{f},\hat{x})$ of MFPC$_s$ (MFPC$_w$), we denote by $\mathcal{F}(\hat{f},\hat{x})$ the corresponding objective function value assumed by the $\mathcal{F}$ variable. Moreover, we indicate as LP-MFPC$_s$ and LP-MFPC$_w$ the linear relaxations of the strong and the weak formulations, respectively. Given a subset of arcs $F \subseteq A$, we further denote by MFPC$_s(F)$ and MFPC$_w(F)$ the strong and the weak formulations, in which the set of variables is restricted to those associated with the arcs in F, while LP-MFPC$_s(F)$ and LP-MFPC$_w(F)$ are their linear relaxations, respectively.

3. Heuristic approaches

This section provides an in-depth description of the heuristic and matheuristic methods developed to tackle the MFPC. In particular, Section 3.1 introduces a modified version of the well-established augmenting path algorithm tailored for the MFPC. Section 3.2 outlines an enhanced algorithm derived by integrating the greedy algorithm from Section 3.1 into the Carousel Greedy framework. In Section 3.3, we detail a Kernel Search algorithm. Finally, in Section 3.4, we introduce the innovative hybrid approach *Kernousel*, which combines a Kernel Search algorithm with the previously discussed Carousel Greedy algorithm. Main notation used in the algorithms can be found in the second part of Table A.7 in Appendix.

3.1. Greedy algorithm

In this section, we present a non-trivial heuristic modification designed to accommodate conflict constraints within the conventional augmenting path algorithm typically employed for solving the maximum flow problem.

The key aspects of such an adaptation concern *(i)* the update of the residual network after sending a positive flow along a path P, requiring the exclusion of all the arcs in conflict with the ones traversed by P; and *(ii)* the ad-hoc strategy employed to identify each augmenting path while complying with conflicts. The algorithm employs the widely recognized capacity scaling approach to prioritize finding augmenting paths with the highest capacity.

Formally, given a capacitated network $G = (V,A)$ and a non negative flow $\hat{f} = \{\hat{f}_{ij} | (i,j) \in A\}$, i.e., a vector satisfying constraints (1b) and (1d), the *residual capacity* associated with each pair of nodes $(i,j) \in V \times V$, with respect to $\hat{f}$, is denoted by $r_{ij}^{(\hat{f})} = u_{ij} - \hat{f}_{ij} + \hat{f}_{ji}$. Furthermore, the *residual network* of G induced by $\hat{f}$ is defined as $G_{\hat{f}} = (V_{\hat{f}}, A_{\hat{f}})$, where $V_{\hat{f}} = V$ and $A_{\hat{f}} = \{(i,j) \in V \times V : r_{ij}^{(\hat{f})} > 0\}$. Any directed s-t path in $G_{\hat{f}}$ is an augmenting path. Note that there may be arcs in $A_{\hat{f}}$ that conflict with each other, as well as arcs in $A_{\hat{f}}$ that conflict with arcs $(i,j) \in A$ for which $\hat{f}_{ij} > 0$. The generation of a new augmenting path in $G_{\hat{f}}$ must ensure the exclusion of any conflicting arcs. This requirement is met through the following two steps. First, whenever $G_{\hat{f}}$ is updated, all the arcs conflicting with some of the arcs in $\bigcup_{(i,j)\in A_{\hat{f}} : \hat{f}_{ij}>0} \delta(i,j)$ are removed from $A_{\hat{f}}$. Second, the ComputeNextPath procedure is left to identify a non-conflicting augmenting path in the resulting residual network $G_{\hat{f}}$. By iterating these two stages until no more augmenting paths are detected, the designed greedy algorithm incrementally builds an ordered sequence of paths and finally provides a solution $S = \langle P_1, P_2, \dots, P_{n_S} \rangle$ of non-conflicting augmenting paths (where we indicate by n_S the number of paths belonging to the solution S), along with a vector Δ, indexed by the identified paths, such that $\Delta_{P_k} = \min\{r_{ij}^{(\hat{f})} : (i,j) \in P_k\}$ is the augmenting capacity associated with P_k, $k \in \{1,\dots,n_S\}$. Finally, the amount of flow crossing the arc (i,j) in this solution S is given by $f_{ij} = \max\{0, u_{ij} - r_{ij}^{(\hat{f})}\}$.

The pseudocode of the procedure is reported in Algorithm 1, which takes as input a graph $G = (V,A)$, a source node $s \in V$ and a sink node

Algorithm 1: MFPC-Greedy

```
Data: Original graph $G = (V, A)$; source node $s$; sink node $t$.
Result: Ordered sequence of the identified conflict-free
        augmenting paths $S = \langle P_k \rangle_{k \in \{1,\dots,|S|\}}$; their capacities
        $\Delta = \langle \Delta_{P_k} \rangle_{k \in \{1,\dots,|S|\}}$; final residual graph $G_{\hat{f}}$.
1  $S \leftarrow \langle\rangle$; $\Delta \leftarrow \langle\rangle$;
2  $\hat{f}_{ij} \leftarrow 0, \forall (i,j) \in V \times V$;
3  $G_{\hat{f}} \leftarrow$residual graph of $G$ w.r.t. $\hat{f}$;
4  $(P, \Delta_P) \leftarrow$ComputeNextPath($G_{\hat{f}}, s, t, \langle\rangle, None$);
5  while $P \neq \langle\rangle$ do
6      Send a flow $\Delta_P$ through path $P$ and update $\hat{f}$ and $G_{\hat{f}}$,
         accordingly;
7      Remove from $G_{\hat{f}}$ all the arcs in $\bigcup_{(i,j) \in A_{\hat{f}} : \hat{f}_{ij} > 0} \delta(i,j)$;
8      Add $P$ to $S$ and $\Delta_P$ to $\Delta$;
9      $(P, \Delta_P) \leftarrow$ComputeNextPath($G_{\hat{f}}, s, t, \langle\rangle, None$);
10 end
11 return $S$, $\Delta$, $G_{\hat{f}}$
```

$t \in V$ such that $s \neq t$. In the initialization phase, the empty sequences S and Δ, as well as the zero-flow $\hat{f}$, are initialized, and the auxiliary graph $G_{\hat{f}}$ is built (Lines 1–3). A first augmenting path P that does not violate any conflict in the residual graph $G_{\hat{f}}$ is detected by using the ComputeNextPath procedure described in Algorithm 2 (Line 4). If the conflict-free augmenting path P returned by the algorithm is not empty, an iteration of the main loop (Lines 5–10) is performed, and additional iterations of this loop follow, as long as the ComputeNextPath procedure identifies non-empty paths. In each iteration, the amount of flow Δ_P is sent along P, and the flow and the residual network are updated accordingly (Line 6). Note that, if the amount of flow sent on some arc (i,j) is reset to a positive value by performing this operation, the arcs in $\delta(i,j)$ are re-added to the residual network. Subsequently, all the arcs that are in conflict with the ones having a positive flow in $G_{\hat{f}}$ are removed from the residual network (Line 7), while S and Δ are updated (Line 8). Finally, the ComputeNextPath procedure is newly invoked to possibly obtain an augmenting path P that will be added to the solution in the next iteration (Line 9).

At the end of the kth iteration, S stores exactly k non-empty conflict-free augmenting paths, according to the order in which they have been generated by the algorithm, i.e., $S = \langle P_i \rangle_{i \in \{1,\dots,k\}}$, while $\Delta = \{\Delta_{P_i}\}_{i=1,\dots,k}$ reports the associated augmenting flows. In the style of a constructive algorithm, once a path is added to the solution, it is no longer modified or removed. However, it is worth noting that updating the residual network can still possibly make previously excluded arc(s) available again. This happens when the conflicting arc(s) used in the solution no longer carry some flow. As a trivial example, consider $S = \langle P_1, P_2 \rangle$, where $P_1 = \langle (s,i),(i,j),(j,t) \rangle$, $P_2 = \langle (s,j),(j,i),(i,t) \rangle$ and $\Delta_{P_1} = \Delta_{P_2}$. It is easy to see that the actual flow $\hat{f}_{ij}$ carried by the arc (i,j) is Δ_{P_1} after the first iteration, and zero after the second one. Accordingly, when updating $G_{\hat{f}}$ after the first iteration, all the arcs in $\delta(i,j)$ are removed from $A_{\hat{f}}$, while each of them is re-added after the second iteration unless it is in conflict with some other arc of P_1 or P_2 with associated positive flow.

The ComputeNextPath procedure, used on Line 5 of Algorithm 1, applies a greedy strategy in order to identify an augmenting path satisfying all the conflict constraints in $G_{\hat{f}}$. The well-known capacity scaling approach (Cormen et al., 2022) is adopted to compute the first s-t augmenting path, and a series of recursive iterations are performed on this path to remove possible conflicts. More in detail, the procedure starts from the source node and selects the sequence of arcs of the path until an arc (i,j), in conflict with at least one of the previous arcs of the path, is met. Let (v,w) be the first of the arcs of the path in conflict with (i,j). At this point, the algorithm randomly chooses one between (i,j) and (v,w) as the pivot arc (p_N, q_N), i.e., the first arc to discard in

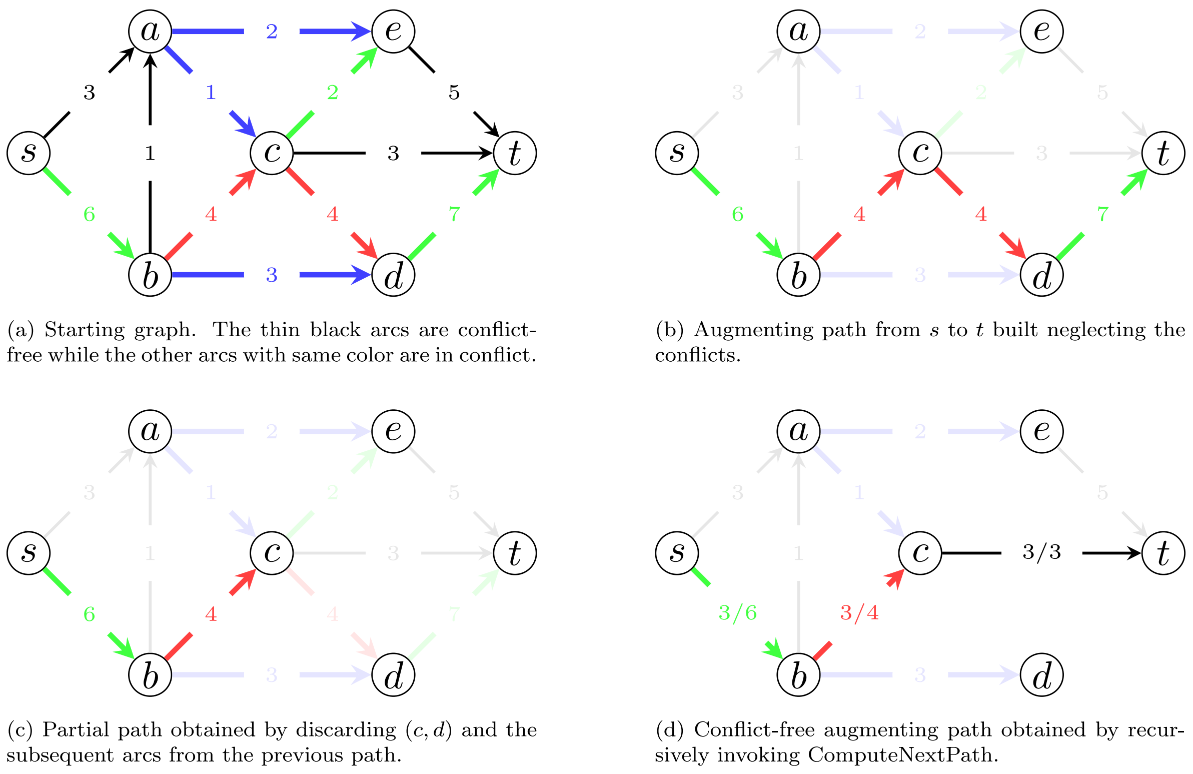
Algorithm 2: ComputeNextPath

```
Data: Residual graph $G_{\hat{f}}$; source node $s$; sink node $t$; possibly
      empty partial solution path $P_{INIT}$; pivot arc $(p,q)$,
      required in case a non-empty $P_{INIT}$ is provided.
Result: Conflict-free $s$-$t$ path $P$, with the associated capacity
        $\Delta_P$, if any is found. An empty path, along with its zero
        capacity, are returned, otherwise.
1  if $P_{INIT} = \langle\rangle$ then $\bar{s} \leftarrow s$;
2  else
3      $\hat{r} \leftarrow$current vector of the residual capacities of the arcs in
         $G_{\hat{f}}$;
4      $r_{ij} \leftarrow 0, \forall (i,j) \in A$ s.t. $\exists\, k \in V \setminus \{j\} : (k,j) \in P_{INIT}$;
5      $r_{pq} \leftarrow 0$; $\bar{s} \leftarrow p$;
6  end
7  $(\hat{P}, \Delta_{\hat{P}}) \leftarrow$CapacityScaling($G_{\hat{f}}, \bar{s}, t$);
8  if $\hat{P} = \langle\rangle$ then return $\langle\rangle$, 0;
9  else
10     $(P, (p_N, q_N)) \leftarrow$ValidatePath($P_{INIT}, \hat{P}$);
11     if $p_N \neq None$ then $(P, \Delta_P) \leftarrow$
         ComputeNextPath($G_{\hat{f}}, s, t, P, (p_N, q_N)$);
12     $r_{ij} \leftarrow \hat{r}_{ij}, \forall (i,j) \in A$ s.t. $\exists\, k \in V \setminus \{j\} : (k,j) \in P_{INIT}$;
13     $r_{pq} \leftarrow \hat{r}_{pq}$;
14     return $P, \Delta_P$;
15 end
```

order to avoid the conflict violation. Then, the partial path from the source to node p_N is saved and a second path, starting from p_N and reaching the sink, is built. This process is recursively repeated until a conflict-free path from s to t is built or the procedure fails to construct such a path.

Fig. 2 provides an example of how the MFPC-Greedy algorithm works. To simplify the description of the example, we do not show how the residual network is updated at each iteration. Fig. 2(a) shows a flow network in which the conflict-free arcs are depicted as black thin-line arrows, while arcs in conflict with each other are represented using thick-line arrows of the same color. Fig. 2(b) shows the first s-t path of highest capacity $\langle (s,b),(b,c),(c,d),(d,t) \rangle$ computed by ComputeNextPath, temporarily ignoring the conflict constraints. If the just obtained path does not contain any pair of arcs in conflict with each other, then the procedure sends the flow from s to t along it, otherwise it looks for the first pair of arcs in the path in conflict with each other, that is the pair $\{(b,c),(c,d)\}$ in this example. ComputeNextPath randomly selects one of these two arcs, let us say (c,d), and it removes (c,d) and the following arcs from the path obtaining the new partial path $\langle (s,b),(b,c) \rangle$ depicted in Fig. 2(c). ComputeNextPath is recursively invoked, in order to build a conflict-free path toward the sink t. In this example, ComputeNextPath adds to the partial path the arc (c,t), obtaining the conflict-free augmenting path $\langle (s,b),(b,c),(c,t) \rangle$, along which three units of flow are sent (Fig. 2(d)). Notice that, since this path uses the arcs (s,b) and (b,c), then all the arcs that are in conflict with them (i.e., the ones represented as green and red thick-line arrows) are removed from the network because they cannot produce any additional conflict-free augmenting path.

The pseudocode of the ComputeNextPath procedure is reported in Algorithm 2. It receives as input the residual graph $G_{\hat{f}}$, the source and the sink nodes s and t, together with two optional parameters: a partial solution P_{INIT} and a pivot arc (p,q), where p is the last node of P_{INIT}, if $P_{INIT} \neq \langle\rangle$. If available, P_{INIT} is used as the initial path. However, as mentioned earlier, when a conflict is identified during the construction process, the algorithm randomly chooses a new pivot arc. This pivot arc is selected from P_{INIT} with a 50% probability. In the end, if a conflict-free augmenting path P is found, then the algorithm returns P and the amount of flow Δ_P sent through P. Otherwise, an empty path is



(a) Starting graph. The thin black arcs are conflict-free while the other arcs with same color are in conflict.
(b) Augmenting path from s to t built neglecting the conflicts.
(c) Partial path obtained by discarding (c, d) and the subsequent arcs from the previous path.
(d) Conflict-free augmenting path obtained by recursively invoking ComputeNextPath.

Fig. 2. Example of the MFPC-Greedy algorithm. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

returned, along with its zero capacity, to indicate that no conflict-free augmenting path has been found.

More in detail, the first operation performed by the algorithm is to check whether a partial path P_{INIT} is provided (Line 1). If this is not the case, the current source node $\bar{s}$ and the original source s coincide. Otherwise, $\bar{s}$ is initialized with the tail of the provided pivot arc, the residual capacities of the pivot arc (p, q) and those of the arcs entering any node traversed by P_{INIT} are stored and temporarily set to zero in $G_{\hat{f}}$, in order to prevent the creation of non-simple paths (Lines 3–5). The original residual capacities are afterward restored on Lines 12–13. To compute an augmenting s-t path, the well-known capacity scaling approach (Cormen et al., 2022) is adopted (Line 7). The method consists of imposing a limit on the minimum amount of flow sent along the desired path and iteratively decreasing such a limit until a path is found. If it is not possible to send additional flow from s to t on $G_{\hat{f}}$, the path $\hat{P}$ is empty and the flow $\Delta_{\hat{P}}$ is zero. In this case, the ComputeNextPath procedure stops and returns the empty path (Line 8). Otherwise, in order to check for conflicting arcs, potentially included in P_{INIT} and $\hat{P}$, the ValidatePath procedure is exploited (Line 10). If the concatenation of P_{INIT} and $\hat{P}$ contains two conflicting arcs, then the algorithm returns (p_N, q_N) as the selected pivot arc, and P as the partial conflict-free solution path, which contains all the arcs preceding p_N in the concatenation of P_{INIT} and $\hat{P}$. In order to obtain a conflict-free s-t path, starting from the new pivot arc (p_N, q_N), ComputeNextPath is recursively invoked (Line 11). Otherwise, if no conflict violation is detected, no value is assigned to (p_N, q_N), whereas P becomes the concatenation of P_{INIT} and $\hat{P}$ and is finally returned (Line 14).

Algorithm 3 provides the pseudocode of the ValidatePath procedure used to validate the generated augmenting paths with respect to the defined set of conflicts. The algorithm receives two paths as input: P_{sk}, a partial path from the source node s to an intermediate node $k \neq t$, which does not include any conflicting arc pairs and may be empty; and P_{kt}, a candidate completing path from node k to the sink node t that may violate conflicts. Clearly, if P_{sk} is empty, $k = s$, i.e., P_{kt} starts from the source node s. The output of the algorithm consists of a conflict-free

Algorithm 3: ValidatePath

```
Data: Conflict-free partial path $P_{sk}$; candidate completing path $P_{kt}$.
Result: Conflict-free $s$-$t$ solution path $P$ if no conflict is
        detected in $\langle P_{sk}, P_{kt} \rangle$; conflict-free partial path $P$ with a
        new pivot arc $(p_N, q_N)$, otherwise.
1  $P \leftarrow P_{sk}$; $(p_N, q_N) \leftarrow None$;
2  for $(i, j)$ in $P_{kt}$ do
3      if $\delta(i, j) \cap P = \emptyset$ then $P$.append($(i, j)$);
4      else
5          if $(i, j)$ is in conflict with the first arc in $P$ then
6              $(p_N, q_N) \leftarrow (i, j)$;
7          else if $random(0, 1) = 0$ then
8              $(p_N, q_N) \leftarrow (i, j)$;
9          else
10             $(k, l) \leftarrow$ first arc of P belonging to $\delta(i, j)$;
11             $P \leftarrow \langle (p, q) \in P : (p, q)$ precedes $(k, l) \rangle$;
12             $(p_N, q_N) \leftarrow (k, l)$;
13         end
14         return $P, (p_N, q_N)$;
15     end
16 end
17 return $P, (p_N, q_N)$;
```

path P, which is a s-t path if no conflicting arc pair has been detected among all the arcs of P_{sk} and P_{kt}. However, if any conflict is detected, P is a partial path, and a pivot arc (p_N, q_N), i.e., the first excluded arc of P_{kt}, is returned.

In the initialization phase, P is assigned the partial path P_{sk}, whereas (p_N, q_N) is initialized to $None$ (Line 1). Then, the arcs in the candidate path P_{kt} are processed in the order they appear until a violated conflict has been detected or the whole path has been analyzed (Lines 2–16). In particular, each arc $(i, j) \in P_{kt}$ is added to P if such an insertion does not compromise the feasibility of P, i.e., if (i, j) does not conflict with any of the arcs currently in P (Line 3), that is, with a slight abuse of notation, if $\delta(i, j) \cap P = \emptyset$. Otherwise, if (i, j) is in conflict with the first arc of P, then (i, j) becomes the first excluded arc of P_{kt}, i.e., $(p_N, q_N) = (i, j)$; in the remaining cases, (i, j) is selected as pivot arc with half the probability; with the same probability, the choice falls on the first arc $(k, l) \in \delta(i, j) \cap P$, namely the first arc of P conflicting with (i, j), and all the arcs placed after (k, l) are removed from P (Lines 7–12, where *random*(x, y) is a function returning integer values between x and y with discrete uniform probability). In both cases, the execution stops and the extended partial path is returned, along with the selected pivot arc (Line 14). If no conflicting arcs are detected in any iteration, at the end of the execution P is a conflict-free s-t path, coinciding with the concatenation of P_{sk} and P_{kt}. Such a complete path is returned along with the unassigned pivot arc (p_N, q_N) (Line 17).

3.1.1. *Computational complexity*

The first procedure, ValidatePath, plays a critical role in the whole process, as it is invoked every time that a conflict is detected while building an augmenting path and involves operations that scale quadratically with the size of the input. Indeed, the most expensive operations carried out by this procedure involve checking for conflicts between the provided conflict-free partial path P_{sk} and the candidate completing path P_{kt} and building the conflict-free path P, accordingly (Lines 2–3). At the beginning, P is set to P_{sk}. Then, in the worst case, no conflict is detected and an arc is added to P at each iteration of the **for** loop in Line 2 of Algorithm 3. Verifying, in Line 3, that no arc is contained in both the two lists P and $\delta(i, j)$ for a given $(i, j) \in P_{kt}$, requires at most $|\delta(i, j)| \cdot |P|$ comparisons. As the size of P at iteration h can be written as $|P_{sk}| + h$, the overall number of these comparisons in the **for** loop is upper bounded by $\sum_{h=1}^{|P_{kt}|} \left(D_{\max} \left(|P_{sk}| + h\right)\right)$, where $D_{\max}$ denotes the maximum number of conflicts involving an arc of G. This may be rewritten as $D_{\max} \cdot \left(|P_{sk}||P_{kt}| + \frac{|P_{kt}|(|P_{kt}|-1)}{2}\right)$, that is the worst-case computational complexity of the ValidatePath procedure is $\mathcal{O}\left(D_{\max} \cdot \max\left(|P_{sk}||P_{kt}|, |P_{kt}|^2\right)\right)$, which in turn is $\mathcal{O}\left(|V|^2 D_{\max}\right)$.

Due to its recursive nature, the computational complexity of the ComputeNextPath algorithm depends on the maximum number of recursions occurring during the computation. In terms of the underlying network, this corresponds to the maximum number of pivot arcs that may be identified in Line 10 of Algorithm 2. As the capacity of every identified pivot arc is set to zero in Line 5 of the algorithm, the maximum number of pivots, as well as the maximum number of recursions, corresponds to the number of arcs of the network, i.e., $|A|$. Furthermore, the computational time complexity associated with each recursive call of the algorithm results from the sum of the complexities of both the CapacityScaling and ValidatePath procedures, the former being $\mathcal{O}\left(|A|^2 \log_2 C_{\max}\right)$ (Cormen et al., 2022), where $C_{\max}$ denotes the largest capacity associated with an arc of G, and the latter being $\mathcal{O}\left(|V|^2 D_{\max}\right)$. Thus, the overall worst-case complexity of the ComputeNextPath algorithm is $\mathcal{O}\left(|A| \left(|A|^2 \log_2 C_{\max} + |V|^2 D_{\max}\right)\right)$.

Finally, the complexity of MFPC-Greedy depends on the number of times the ComputeNextPath procedure is invoked. In the worst case, the maximum flow $\mathcal{F}^*$ is achieved using $\mathcal{F}^*$ different augmenting paths, each one carrying a unit flow. Thus, the worst-case time complexity of MFPC-Greedy is $\mathcal{O}\left(\mathcal{F}^* \left(|A| \left(|A|^2 \log_2 C_{\max} + |V|^2 D_{\max}\right)\right)\right)$.

3.2. *Carousel greedy*

In the following, we provide a detailed description of the Carousel Greedy (CG) algorithm designed for the MFPC. CG is based on the greedy procedure introduced in Section 3.1.

Fig. 3 schematically illustrates how our CG algorithm works. CG starts from a feasible solution provided by MFPC-Greedy procedure. As described in Section 3.1, such a solution is composed of a sequence $S = \langle P_1, P_2, \ldots, P_{|S|} \rangle$ of conflict-free augmenting paths, appearing in the order they have been selected by the greedy algorithm. This sequence S is iteratively updated by CG based on two parameters α and β. More in detail, CG removes from S the last $\beta|S|$ augmenting paths, obtaining the so-called *carousel start*. In this context, β denotes the percentage of augmenting paths to be removed from the tail of S. At this point, CG carries out $\alpha|S|$ iterations, where α is an integer value used to define the overall number of iterations of the algorithm. In each iteration, the left-most item, i.e., the oldest selected augmenting path, is removed from S and the ComputeNextPath procedure is exploited to obtain another conflict-free augmenting path, if any, with respect to the updated S. In Fig. 3, at the first iteration, CG removes the oldest path P_1 and ComputeNextPath procedure adds the path P_4. At the second iteration, the oldest path P_2 is removed and path P_5 is added, and so on. After the last iteration, the resulting collection of non-conflicting augmenting paths $S = \langle P_5, P_6, P_7 \rangle$ is completed by invoking the MFPC-Greedy procedure that represents the final step of the procedure.

Unlike the classical CG approach, introduced in Cerrone et al. (2017), where a feasible solution is obtained only at the end of the computation, the designed CG algorithm for the MFPC, by construction, yields a feasible solution at the end of each iteration. Furthermore, to enhance the exploration of the solution space, when an augmenting path is excluded from the current solution, its first arc is temporarily assigned a residual capacity of zero. This is done to prevent the reselection of the same augmenting path in the subsequent iteration. Likewise, a frequency vector is kept to record the occurrences of each generated path, and the arcs of the most frequently chosen path have their capacity temporarily set to zero. Moreover, the MFPC-Greedy algorithm is run at the end of each iteration, thus obtaining an intermediate complete solution, i.e., a collection of paths that the algorithm is no longer able to extend with additional augmenting paths. Although the paths generated to obtain such a solution are not added to S, the value of the obtained solution is still compared with the incumbent one, potentially leading to an update of the best solution found. Contextually, a collection $\mathcal{P}$ of all the paths computed by the algorithm is built and returned at the end of the computation.

Fig. 4 provides an example of how CG works. To simplify the description of the example, we do not show how the residual network is updated at each iteration. Fig. 4(a) shows a feasible solution composed of three conflict-free augmenting paths: $\langle(s, b), (b, a), (a, e), (e, t)\rangle$, $\langle(s, b), (b, c), (c, t)\rangle$ and $\langle(s, a), (a, e), (e, t)\rangle$. The flow sent through these paths is 5. Carousel Greedy creates the *carousel start* solution by removing the last generated augmenting path $\langle(s, a), (a, e), (e, t)\rangle$ (Fig. 4(b)). Then it starts the iterations on this solution. At the first iteration, CG removes the first augmenting path $\langle(s, b), (b, a), (a, e), (e, t)\rangle$ and, to avoid rebuilding the same augmenting path, it sets the residual capacity of the first arc (s, b) of this path equal to zero (Fig. 4(c)). CG completes the iteration by invoking the ComputeNextPath procedure on s obtaining the new augmenting path $\langle(s, a), (a, c), (c, e), (e, t)\rangle$(Fig. 4(d)) with flow 3. The total flow is now equal to 6.

The pseudocode of the designed CG algorithm is reported in Algorithm 4, which takes as input the original graph G, the source node s, the sink node t and the α and β parameters. At the end of the computation, the output of the algorithm consists of the best solution found S^*, i.e., an ordered list of conflict-free augmenting paths yielding the highest known flow, along with the collection $\mathcal{P}$ of all the generated augmenting paths and a frequency vector τ, where τ_P indicates the number of times that the path P has been added to a partial solution by the algorithm, for each $P \in \mathcal{P}$. The collection $\mathcal{P}$ and the frequency vector τ play a key role in the initialization phase of the KO algorithm, providing information used to populate the kernel set.

In the initialization phase, the MFPC-Greedy procedure is used to obtain a starting initial solution $S = \langle P_1, \ldots, P_{|S|} \rangle$, along with the associated flow vector Δ and residual graph $G_{\hat{f}}$ (Line 1). Then, the best solution found S^*, the related flow value z^*, the collection of generated paths $\mathcal{P}$ and the frequency vector τ are initialized, accordingly (Lines 2–3). At the beginning, τ indicates that each path of the starting solution

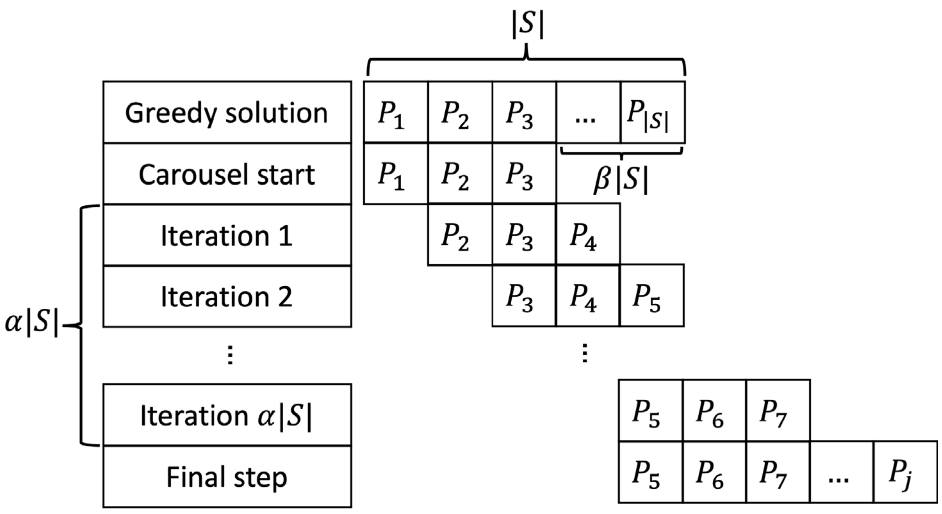
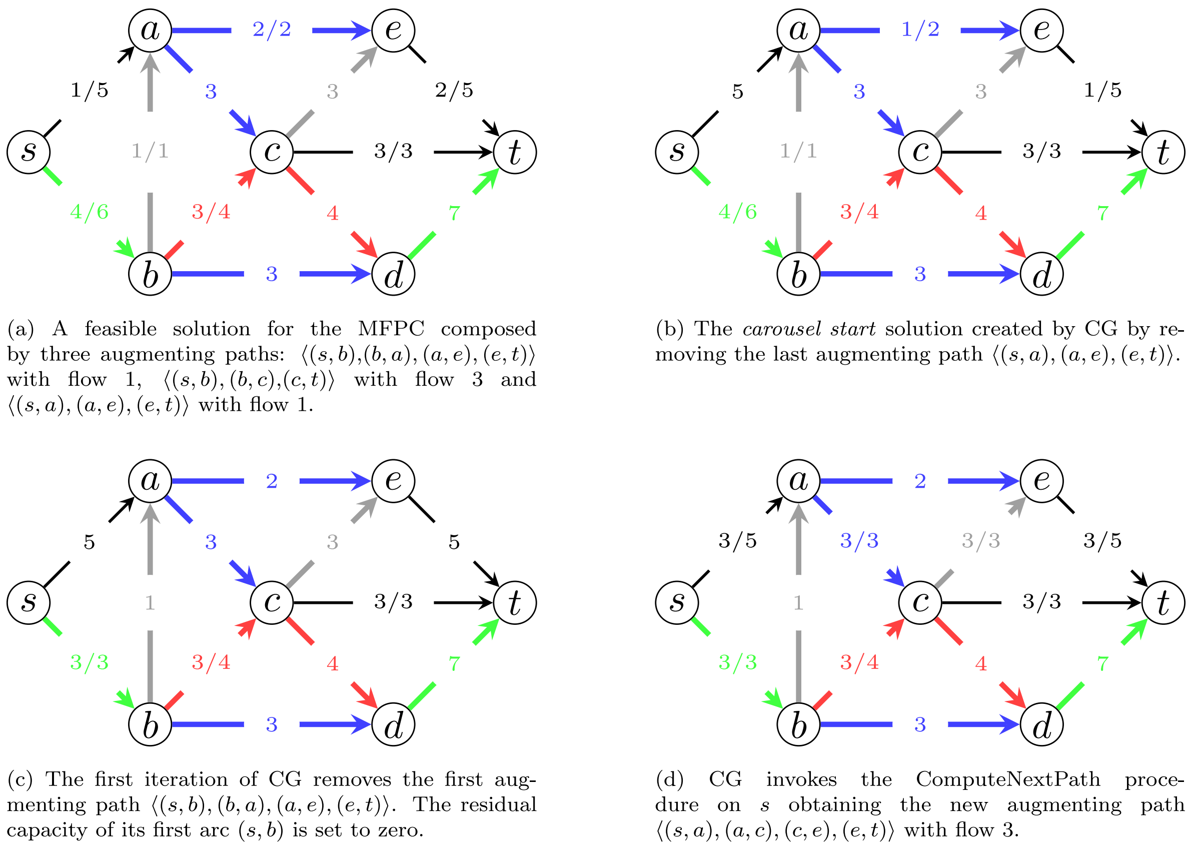


Fig. 3. Scheme of the Carousel Greedy algorithm designed for the MFPC.



(a) A feasible solution for the MFPC composed by three augmenting paths: $\langle (s,b),(b,a),(a,e),(e,t)\rangle$ with flow 1, $\langle (s,b),(b,c),(c,t)\rangle$ with flow 3 and $\langle (s,a),(a,e),(e,t)\rangle$ with flow 1.

(b) The *carousel start* solution created by CG by removing the last augmenting path $\langle (s,a),(a,e),(e,t)\rangle$.

(c) The first iteration of CG removes the first augmenting path $\langle (s,b),(b,a),(a,e),(e,t)\rangle$. The residual capacity of its first arc (s,b) is set to zero.

(d) CG invokes the ComputeNextPath procedure on s obtaining the new augmenting path $\langle (s,a),(a,c),(c,e),(e,t)\rangle$ with flow 3.

Fig. 4. Example of the Carousel Greedy algorithm.

has been chosen exactly once. Subsequently, the so-called *carousel start* S^C is obtained, by removing from S the last $\beta\%$ of its paths (Line 4), and the $\alpha|S|$ CG iterations are performed (Lines 5–19).

At the beginning of each iteration, the oldest path chosen according to the greedy strategy, denoted by P^O, is removed from S^C (Line 6), then the residual graph $G_{\hat{f}}$ associated with the flow $\hat{f}$ induced by the resulting solution S^C is computed and all the arcs conflicting with some of the used arcs are removed from $G_{\hat{f}}$ (Lines 7–8). Subsequently, the path $P_{\tau_{max}} \in \mathcal{P}$ with the highest frequency is selected, breaking ties by preferring the earlier generated one (Lines 9–10). After the removal of P^O from S, the residual capacity of its first arc is set to zero (Line 11), in order to assure that such a path cannot be immediately rebuilt by ComputeNextPath. This operation allows the selection of new paths that are enough different from P^O. Unfortunately, this strategy did not prove effective when applied on the most frequent path $P_{\tau_{max}}$. Indeed, during the computational tests, we observed that, by forbidding only the first arc of $P_{\tau_{max}}$, the new path identified by ComputeNextPath is usually very similar to $P_{\tau_{max}}$. This may be justified by the fact that the most frequent paths typically differ for a few arcs. As a consequence, by forbidding only the first arc, the procedure would often identify another frequent path that is already in $\mathcal{P}$. Since our aim is to obtain

Algorithm 4: MFPC-CarouselGreedy

```
Data: Original graph $G$; source node $s$; sink node $t$; $\alpha$ and $\beta$
      parameters.
Result: Best solution found $S^*$; collection $\mathcal{P}$ of all the
        augmenting paths generated during the computation;
        frequency vector $\tau$.
1  $(S, \Delta, G_{\hat{f}}) \leftarrow$MFPC-Greedy$(G, s, t)$;
2  $S^* \leftarrow S$; $z^* \leftarrow z(S)$;
3  $\mathcal{P} \leftarrow \{P : P \in S\}$; $\tau_P \leftarrow 1, \forall P \in S$;
4  initialize the carousel start $S^C$ with the first $\lfloor (1-\beta)|S| \rfloor$ paths
     in $S$;
5  for $k \in \{1, \dots, \alpha|S|\}$ do
6      remove the oldest path $P^O$ from $S^C$;
7      compute the residual network $G_{\hat{f}}$ of $G$ w.r.t. the flow $\hat{f}$
         induced by $S^C$;
8      remove from $G_{\hat{f}}$ all the arcs in conflict with some of the
         arcs of any path in $S^C$;
9      let $\tau_{\max} \leftarrow \max_{P \in \mathcal{P}}\{\tau_P\}$ be the highest frequency of a path
         in $\mathcal{P}$;
10     let $P_{\tau_{max}}$ be the oldest among the paths in $\mathcal{P}$ with frequency
         $\tau_{\max}$;
11     set to zero the residual capacity of the first arc in $P^O$ and
         of all the arcs in $P_{\tau_{max}}$;
12     $(P, \Delta_P) \leftarrow$ComputeNextPath$(G_{\hat{f}}, s, t, \langle\rangle, None)$;
13     if $P \neq \langle\rangle$ then
14         $\mathcal{P} \leftarrow \mathcal{P} \cup \{P\}$; $\tau_P \leftarrow \tau_P + 1$;
15         add $P$ to $S^C$ and update $\hat{f}$ and $G_{\hat{f}}$, accordingly;
16     end
17     obtain an intermediate complete solution $S^{C+}$ by running
         MFPC-Greedy$(G, s, t)$ with the starting partial solution
         $S = S^C$;
18     $\mathcal{P} \leftarrow \mathcal{P} \cup \{P\}$, $\tau_P \leftarrow \tau_P + 1, \forall P \in S^{C+} \setminus S^C$;
19     if $z^* \le z(S^{C+})$ then $S^* \leftarrow S^{C+}$; $z^* \leftarrow z(S^{C+})$;
20 end
21 return $S^*$, $\mathcal{P}$, $\tau$;
```

a collection $\mathcal{P}$ containing paths that are as different as possible from each other, we opted for a stronger policy that forbids the selection of all the arcs of $P_{\tau_{max}}$. At this point, the ComputeNextPath procedure is used to identify the next most promising non-conflicting path (Line 12). If any augmenting path P is found, it is added to the collection $\mathcal{P}$ and the number τ_P of times P has been chosen is updated, as well as the current solution S^C, the current flow $\hat{f}$ and the resulting residual graph $G_{\hat{f}}$ (Lines 13–16). Finally, a customized version of the MFPC-Greedy algorithm that receives a starting solution as an additional input, is used to get an intermediate complete solution. Such a customized version is obtained by modifying Lines 1–3 in Algorithm 1 in such a way that S is initialized with an input starting solution consisting of a sequence of augmenting paths, Δ is the corresponding sequence of flows associated with the paths in S, while the flow $\hat{f}$ and the residual graph $G_{\hat{f}}$ are built accordingly. In this context, S^C is provided to the MFPC-Greedy algorithm as starting solution (Line 17). According to the resulting solution, denoted by S^{C+}, the occurrences of each path and, possibly, the best-found solution, are updated (Lines 18–19). Note that, since S^C is not modified, this operation has no impact on the current solution, however, it may influence the computation of the most frequently chosen path $P_{\tau_{max}}$ at the next iteration. In the end, the algorithm returns the best solution found S^*, the collection of generated paths $\mathcal{P}$ and the frequency vector τ.

3.3. Kernel search

In this subsection, we describe the Kernel Search (KS) algorithm developed for the MFPC. As aforementioned, the idea behind the KS framework is to identify the most promising variables of a MILP model and construct and solve restricted subproblems based on them. The set of promising variables (called the *kernel set*) is not computed through a one-shot procedure but changes dynamically during the search. The initial kernel set is identified by using information (values of variables and reduced costs) provided by the optimal solution of the linear relaxation of the MILP problem. In particular, initial kernel set (Λ) consists of all variables with a value greater than zero in the optimal solution of the linear relaxation. The remaining variables are sorted in non-decreasing order of the absolute value of their reduced costs and partitioned into a collection of sets named buckets. At this point, the algorithm solves a restricted MILP problem by considering only the variables inside Λ while setting the remaining ones to zero. The just obtained solution, if any, is improved in the next steps by iteratively solving a sequence of restricted MILP problems. Each subproblem is obtained by considering the variables of Λ together with those belonging, in turn, to each of the buckets. At each iteration, the Λ set is possibly updated by including the variables belonging to the bucket that have been selected in the solution of the current restricted MILP.

In our KS we use the strong formulation MFPC$_s$ to generate the initial kernel set Λ, since its linear relaxation, yielding tighter bounds, may provide better insights about the optimal solution of the problem with respect to the weak formulation MFPC$_w$. Conversely, MFPC$_w$ proves to be faster in solving the restricted MILP problems (i.e., kernel set + bucket). To apply KS to the solution of the MFPC, one should notice that the information provided by the continuous relaxation is limited since the objective function only includes the max flow variable and thus reduced costs are not good predictors of promising variables. For this reason, we decided to build the kernel set and the buckets, according to the values of the flow variables f_{ij} only. Let us observe that, if a variable f_{ij} is assigned to the kernel set or to a bucket, the corresponding x_{ij} is also assigned to it. Moreover, if f_{ij} takes the value zero (i.e., no flow crosses the arc (i,j)) although the corresponding x_{ij} is positive, the variable is not included in the promising set (indeed an equivalent solution can be obtained by setting $x_{ij} = 0$).

KS can be divided into two main phases: *Initialization* and *Improvement*.

- **Initialization phase**. During the Initialization phase, KS solves the LP-MFPC$_s$ problem, obtaining a solution (f^{LP}, x^{LP}). KS sorts the pairs of variables (f_{ij}, x_{ij}) according to non-increasing values of f^{LP}. Let $L_> = \langle (f_{ij}, x_{ij}) : f_{ij}^{LP} > 0 \rangle$ be the sorted list of the most promising variables. The next step of the algorithm consists of defining a priority among the zero-variables, too. Since reduced costs do not provide any information about the relevance of out-of-the-basis variables, for each variable f_{ij} such that $f_{ij}^{LP} = 0$, we compute ρ_{ij} as the number of conflicts among the arc (i,j) and the arcs associated with variables in $L_>$. KS sorts, in non-decreasing order of ρ_{ij}, the pairs of variables (f_{ij}, x_{ij}) such that f_{ij}^{LP} is null. Ties are broken giving higher priority to the arcs with greater capacity since the lower ρ_{ij} the higher the probability that the arc (i,j) may be selected in optimal/near-optimal solution. We indicate by $L_= = \langle (f_{ij}, x_{ij}) : f_{ij}^{LP} = 0 \rangle$ the resulting sorted list. Finally, we refer to L as the sorted list of all pairs of variables obtained by concatenating $L_>$ and $L_=$. Since variables in $L_>$ are more promising than variables in $L_=$, the former precedes the latter in the global list L.
  After the construction of L, KS populates the initial kernel set Λ with the first λ pairs of L, where λ is an integer parameter defined by the user. The remaining pairs of L are partitioned into a collection of *buckets* of fixed size $B = \langle B_1, \dots, B_q \rangle$ with $q = \lceil \frac{|A|-\lambda}{\gamma} \rceil$ and γ the size of each bucket, possibly except the last one. Finally, KS solves the first restricted problem on the initial kernel set MFPC$_w(\Lambda)$, obtaining a starting feasible solution (f^*, x^*) of MFPC, with a corresponding value of flow equal to $\mathcal{F}(f^*, x^*)$.

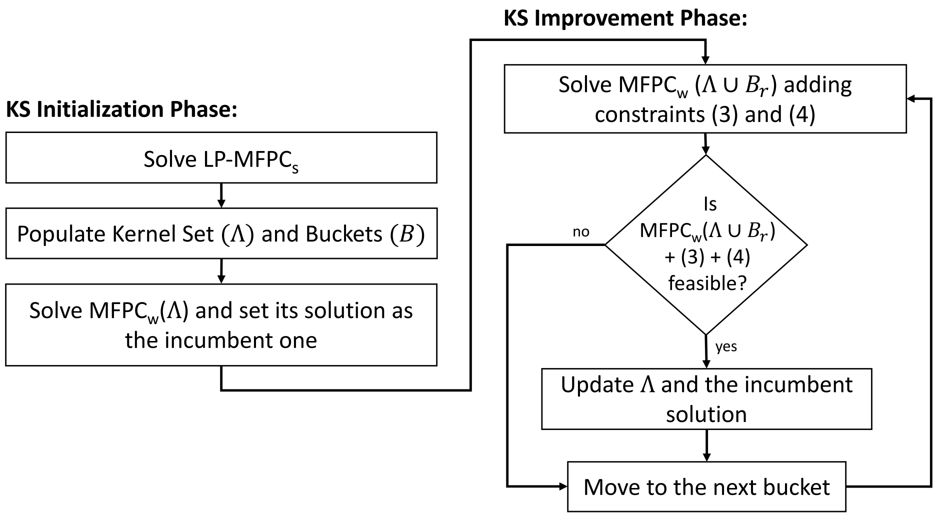

KS Initialization Phase:
Solve LP-MFPC$_s$
Populate Kernel Set (Λ) and Buckets (B)
Solve MFPC$_w$(Λ) and set its solution as the incumbent one
KS Improvement Phase:
Solve MFPC$_w$ ($\Lambda \cup B_r$) adding constraints (3) and (4)
Is MFPC$_w$($\Lambda \cup B_r$) + (3) + (4) feasible?
no
yes
Update Λ and the incumbent solution
Move to the next bucket

Fig. 5. Kernel search diagram.

It is worth noting that, the kernel size λ and the bucket size γ represent the two key parameters of a KS-based algorithm and an appropriate choice of their values is crucial for the effectiveness and efficiency of the resulting algorithm.

- **Improvement phase**. During this phase, KS solves a sequence of q restricted problems each of which is formulated by considering the variables in the current kernel set Λ plus those belonging to a bucket. More in detail, at iteration r, KS solves the restricted problem MFPC$_w(\Lambda \cup B_r)$ with the following two additional constraints:

$$\sum_{(i,j)\in A_r} f_{ij} \geq 1 \tag{3}$$

$$\mathcal{F} \geq \mathcal{F}(f^*, x^*). \tag{4}$$

where $A_r \subset A$ are the arcs associated with variables in bucket B_r. Note that, in the computational experiments described in Section 4, constraint (4) has been implemented by using the lower cutoff parameter of the CPLEX solver.
Constraint (3) requires that at least one of the f_{ij} variables in B_r takes a value greater than or equal to 1 in the solution, whereas constraint (4) imposes that the value of the solution identified at iteration r should be better than or equal to the value of the best incumbent integer solution value $\mathcal{F}(f^*, x^*)$.
Notice that, because of constraints (3) and (4), MFPC$_w(\Lambda \cup B_r)$ may be infeasible: in this case the next restricted problem considering bucket B_{r+1} is analyzed. If, on the contrary, a feasible solution is found, then the incumbent solution is possibly updated and the kernel set Λ is extended by including the pairs of variables (f_{ij}, x_{ij}) in B_r having a positive value for f_{ij} in the provided solution. It is easy to see that the larger the size of the restricted problems, the higher the quality of the final solution. However, larger restricted problems require more computing time to be solved. The right trade-off is determined through the setting of parameters λ and γ. Finally, notice that we decided to implement a simple KS procedure. It is obvious that, given the relevance of variable sorting in KS and the limited information provided by the optimal solution of the continuous relaxation, the implementation of an iterative variant of the method where buckets are scrolled more than once might be beneficial.

The diagram shown in Fig. 5 graphically depicts the main steps of the KS-based algorithm designed for the MFPC.

3.4. Kernousel

In this section, we introduce a hybrid heuristic algorithm for the MFPC, named *Kernousel* (KO), which consists in enhancing a KS-based algorithm by exploiting a CG algorithm in the initialization phase. KO shows a fast and easy-to-implement alternative approach to solving the linear relaxation of the problem for the initialization of the kernel set. The necessity to have this alternative approach arises in optimization problems where, as in our case, the solution of the linear relaxation does not provide useful information to predict promising variables, thus penalizing the effectiveness of the KS algorithm. The idea behind the proposed approach is to exploit the knowledge gained by the CG algorithm during its exploration of the solution space to obtain a more effective partitioning of the variables into the kernel set and the buckets. To this end, we do not use only the final solution S produced by CG, but we also collect in $\mathcal{P}$ all the conflict-free augmenting paths built by the algorithm during the computation. More in detail, to each arc belonging to at least one path of $\mathcal{P}$, we assign a score equal to the number of occurrences (frequency) of this arc in all the paths of $\mathcal{P}$. Therefore, the collection $\mathcal{P}$ allows us to have a more accurate classification of the most promising variables since the higher the score associated with an arc, the more promising the related variable, as it has been used in several conflict-free augmenting paths. KO populates the kernel set Λ with all the variables associated with the arcs of S. If $|\Lambda| < \lambda$, then the kernel set is completed by using first the arcs inside the paths of $\mathcal{P}$, according to their frequency, and later with the arcs inside L. As for KS, once Λ has been populated, KO partitions the remaining $|A| - \lambda$ variables into a collection of *buckets* of fixed size $B = \langle B_1, \ldots, B_q \rangle$ with $q = \lceil \frac{|A|-\lambda}{\gamma} \rceil$ and γ the size of each bucket, possibly except the last one. Once the kernel set and the buckets have been initialized through the information collected by the CG algorithm, the remaining steps of the classical KS are performed.

The block diagram shown in Fig. 6 graphically depicts the main steps of a KO-based algorithm designed for the MFPC. First, the algorithm invokes the CG algorithm to obtain the feasible solution S and the set $\mathcal{P}$ of augmenting paths generated during the computation. Then, it solves the linear relaxation of MFPC$_s$ and builds the sorted list L of variables (see Section 3.3). $\mathcal{P}$ and L are then used to create the initial kernel set and the buckets. In the last step, the algorithm finds a feasible solution S' of MFPC by solving MFPC$_s(\Lambda)$ problem and then sets the best solution between S and S' as the incumbent one. The improvement phase of KO is the same as the one of KS. We now describe in more detail the initialization phase of KO through its pseudocode reported in Algorithm 5.

This procedure builds the initial kernel set, as well as the collection of buckets, and provides a starting feasible solution. More in detail, it takes as input the graph G, the source node s and the sink node t, as well as the CG parameters α and β, the kernel size λ and the bucket

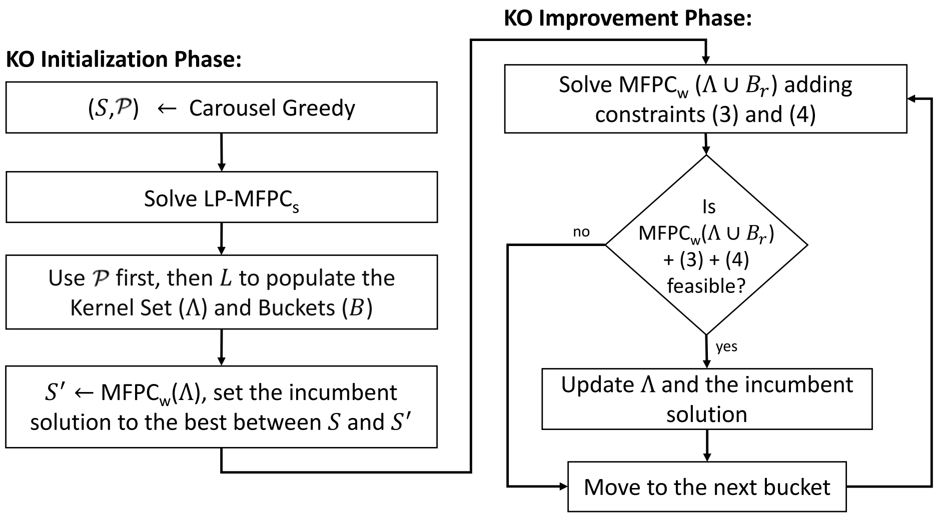

KO Initialization Phase:
$(S,\mathcal{P}) \leftarrow$ Carousel Greedy
Solve LP-MFPC$_s$
Use $\mathcal{P}$ first, then L to populate the Kernel Set (Λ) and Buckets (B)
$S' \leftarrow$ MFPC$_w$(Λ), set the incumbent solution to the best between S and S'
KO Improvement Phase:
Solve MFPC$_w$ ($\Lambda \cup B_r$) adding constraints (3) and (4)
Is MFPC$_w$($\Lambda \cup B_r$) + (3) + (4) feasible?
no
yes
Update Λ and the incumbent solution
Move to the next bucket

Fig. 6. Kernousel diagram.

Algorithm 5: Kernousel-Initialization

```
    Data: Graph $G$; source node $s$; sink node $t$; GC parameters $\alpha$ and $\beta$; minimum kernel size $\lambda$; bucket size $\gamma$.
    Result: Kernel set $\Lambda$; buckets collection $B$; incumbent solution $(f^*, x^*)$.
 1  $(S, \mathcal{P}, \tau) \leftarrow$MFPC-CarouselGreedy$(G, s, t, \alpha, \beta)$;
 2  $E_{\mathcal{P}} = \{(i,j) : \exists\ P \in \mathcal{P}$ s.t. $(i,j) \in P\}$;
 3  $\tau_{ij} = \sum_{P\in\mathcal{P}:(i,j)\in P} \tau_P, \forall\ (i,j) \in E_{\mathcal{P}}$;
 4  $(f^*, x^*) \leftarrow$ incumbent solution built from $S$;
 5  $\Lambda \leftarrow \{(f_{ij}, x_{ij}) : (i,j) \in P, P \in S\}$;
 6  $\bar{L} \leftarrow \langle (f_{ij}, x_{ij}) : (i,j) \in E_{\mathcal{P}}, (f_{ij}, x_{ij}) \notin \Lambda \rangle$;
 7  sort $\bar{L}$ in nonincreasing order of $\tau_{ij}$, $(i,j) \in E_{\mathcal{P}}$ ;
 8  if $|\Lambda| < \lambda$ then
 9      Add to $\Lambda$ the first $\min(|\bar{L}|, \lambda - |\Lambda|)$ elements of $\bar{L}$;
10  end
11  $(f^{LP}, x^{LP}) \leftarrow$LP-MFPC$_s$;
12  Build $L_>, L_=$ and $L$;
13  Remove from $L$ all the pairs already present in $\Lambda$;
14  if $|\Lambda| < \lambda$ then
15      Move from $L$ to $\Lambda$ the first $\lambda - |\Lambda|$ elements of $L$;
16  end
17  $n_B \leftarrow \lceil |L|/\gamma \rceil$;
18  $B \leftarrow \langle B_1, \dots, B_{n_B} \rangle$;   // collection of buckets each with at most $\gamma$ pairs of variables
19  $(f', x') \leftarrow$MFPC$_w(\Lambda)$;
20  if $\mathcal{F}(f', x') > \mathcal{F}(f^*, x^*)$ then
21      $(f^*, x^*) \leftarrow (f', x')$;
22  end
23  return $\Lambda, B, (f^*, x^*)$;
```

size γ and returns the kernel set Λ, the collection of buckets B and a feasible solution (f^*, x^*), having value $\mathcal{F}(f^*, x^*)$. In Line 1, the CG algorithm is invoked to generate a feasible solution S, the collection $\mathcal{P}$ of all the paths built during the computation and the frequency vector τ indicating how many times each path has been added to a partial solution by CG. In Line 2, the set $E_{\mathcal{P}}$ is initialized as the set of arcs belonging to at least one path generated by the CG. In Line 3, the algorithm computes τ_{ij} for each arc $(i,j) \in E_{\mathcal{P}}$, corresponding to the number of times the arc (i,j) appears in a path generated by the CG algorithm. Firstly, the values of the variables f^* and x^* are retrieved from the final solution S identified by the CG algorithm, (Line 4) and the kernel set Λ is initialized with all the variables associated with the arcs appearing in any of the paths belonging to the solution S found by CG (Line 5). Then, the list $\bar{L}$ of variables associated with the arcs used by the CG that have not yet been inserted into the kernel Λ is built and sorted in non-increasing order of the τ_{ij} values (Lines 6–7). If the set Λ has fewer than λ elements the procedure inserts new variables into Λ taking up to the first $\lambda - |\Lambda|$ variables from $\bar{L}$ (Lines 8–10). Subsequently, the procedure solves LP-MFPC$_s$ (Line 11) obtaining the solution (f^{LP}, x^{LP}), from which the lists $L_>$, $L_=$ and L are built (Line 12), as described in Section 3.3. On Line 13, all the variables in Λ are removed from L. The **if** statement at Line 14 verifies if the number of variables inside the kernel set is still lower than its maximum size λ. If this is the case, then the first $\lambda - |\Lambda|$ flow variables in L, or all of them if $|L| < \lambda - |\Lambda|$, are moved from L to Λ (Line 15). On Lines 17–18, the procedure partitions the variables of L in $\lceil |L|/\gamma \rceil$ buckets. This means that the first γ variables of L are placed in B_1, the subsequent ones in B_2 and so on. Consequently, all but the last bucket will have exactly γ variables. Finally, MFPC$_w$ is solved by considering only the variables in Λ and the solution (f', x') is compared with the incumbent one (f^*, x^*) (Lines 19–20). If the new solution yields a higher flow value, then the incumbent solution is updated (Line 21). In the end, the procedure returns the kernel set Λ, the collection of buckets B and the incumbent solution (f^*, x^*).

The pseudocode of the Kernousel-Improvement procedure, implementing the KS paradigm, is reported in Algorithm 6. The **for** loop on Line 1 iterates over all the buckets $B_r \in B$. At each iteration, the set of variables Λ_r is built by merging the variables of Λ with the ones of bucket B_r. Then, the MFPC$_w$ formulation, modified with the introduction of constraints (3) and (4) and restricted to the variables in Λ_r, is used to solve the problem. If the problem results to be infeasible, no further operations are carried out with the current bucket and the procedure returns to Line 1. Otherwise, a feasible solution, better or at least as good as the current one, is found; if strictly better, the incumbent solution $(f^*,\ x^*)$ is updated accordingly (Line 5–7). Furthermore, regardless of the solution value, the subset of variables of B_r, having a positive flow in the just computed solution, are moved into the kernel set (Lines 8–9). In the end, the best solution variables (f^*, x^*) are returned.

4. Computational tests

In this section, we present the results achieved by the CG, KS and KO algorithms and conduct a comparative analysis with the outcomes of the state-of-the-art algorithm for the MFPC. Our algorithms are

Algorithm 6: Kernousel-Improvement

```
   Data: Kernel set $\Lambda$; buckets collection $B$; incumbent solution
         $(f^*, x^*)$.
   Result: Best solution found $(f^*, x^*)$.
 1 for $B_r \in B$ do
 2     $\Lambda_r \leftarrow \Lambda \cup B_r$;
 3     $(\hat{f}, \hat{x}) \leftarrow$optimal solution of MFPC$_w(\Lambda_r)$ + constraints (3)
        and (4);
 4     if $(\hat{f}, \hat{x})$ is feasible then
 5         if $\mathcal{F}(\hat{f}, \hat{x}) > \mathcal{F}(f^*, x^*)$ then
 6             $(f^*, x^*) \leftarrow (\hat{f}, \hat{x})$;
 7         end
 8         $\bar{\Lambda}_r \leftarrow \{(f_{ij}, x_{ij}) \in B_r : \hat{f}_{ij} > 0\}$;
 9         $\Lambda \leftarrow \Lambda \cup \bar{\Lambda}_r$;
10     end
11 end
12 return $(f^*, x^*)$;
```

coded in Python and the mathematical formulations are solved using CPLEX 22.1.0. All CPLEX parameters are set to their default values. The computational tests are carried out on an Intel(R) Core(TM) i7-3770 CPU @ 3.40 GHz processor with 8 GB of RAM.

4.1. Benchmark instances and experimental settings

We tested our algorithms on the collection of benchmark instances provided by Şuvak et al. (2020). By construction, each instance is constructed to have at least one feasible solution with a non-zero flow, and its size depends on three parameters: the number of nodes (n); the arc density ($p = \frac{m}{n(n-1)}$, where m denotes the number of arcs); and the conflict density ($d = \frac{2w}{m(m-1)}$, where w denotes the number of conflicting arc pairs). Parameter n takes value in the set $\{40, 50, 60, 70, 80\}$, p in $\{0.3, 0.4, 0.5, 0.6\}$ and d in $\{0.3, 0.4, 0.5, 0.6\}$. For each triple (n, p, d), Şuvak et al. (2020) generated two random instances; this implies 160 instances altogether. The difference between the two instances lies in the arc capacity range I, with $I \in \{[10, 15], [15, 20]\}$. Each instance is assigned an ID between 1 and 160. Furthermore, according to the number of nodes, instances 1–96 (with $n \in [40, 60]$) and instances 97–160 (with $n \in [70, 80]$) are classified as Small and Large Instances, respectively. Table 1 reports in detail the characteristics of each instance.

The parameters of CG, KS and KO are tuned with the widely used Iterated Racing for Automatic Algorithm Configuration (IRACE) package (López-Ibáñez et al., 2016) that automatically determines the parameter values selecting them from a finite pool of parameter configurations. To ensure a sufficiently diverse sample of instances based on arc capacity ranges, the tuning process alternates between selecting instances with I values in the ranges [10,15] and [15,20]. More in detail, we start by selecting the instance with ID equal to 1, then we process the remaining instances in increasing order of ID and we repeatedly select the one having both a different arc capacity range (I) and at least one of the three remaining parameters (n, p, d) different with respect to the last selected instance. From Table 1, we can observe that the range I for instance 1 is $[10, 15]$, then the next instance selected according to this policy is instance 4 because instance 2 has the same parameters n, p and d as instance 1, while instance 3 has the same arc capacity range as instance 1. Similarly, the next selected instance is the one with ID equal to 5, as both the value of parameter d and the range I are different from the ones of instance 4. By repeating this process until all the instances have been processed, the selected instances are the ones with IDs $1, 4, 5, 8, 9, 12, 13, \dots, 160$, for a total of 80 instances. Using IRACE on this subset of instances allowed us to include at least one instance for each combination of characteristic values during the tuning phase. To verify that this choice did not introduce overfitting, we conducted additional tests using a configuration obtained by tuning on only 10% of the instances. The results confirmed that for at least 78% of the instances, the solution values remained unchanged, while the comparison of the main performance indicators with those of the state-of-the-art approach remained consistent, ruling out overfitting. Since this alternative configuration was faster but slightly less effective, as it identifies 13 new best-known solutions instead of 15, we show in the following the results obtained with the configuration identified by IRACE using the above-mentioned set of 80 instances.

The parameter values are reported in Table 2. The first two columns report the parameter name (*Parameter*) and its description (*Description*), while the third column (*Range Values*) shows, for each parameter, the possible values that IRACE, with a tuning budget of 10 000 runs, used to compute the best configuration. The recommended parameter configurations are displayed under the headings *CG*, *KS*, and *KO* for Carousel Greedy, Kernel Search, and Kernousel, respectively. These parameters were adopted in all our computational experiments.

The range of λ values, provided to IRACE, was defined with the aim to have a size of the initial kernel set that, on one hand, is sufficiently large to allow the construction of an initial feasible solution (possibly different from the one already provided by CG) and, on the other hand, sufficiently small to create a restricted problem that is solvable in a reasonable amount of time. The range of γ values is chosen to obtain an appropriate trade-off between the size and the number of restricted problems generated during the computation. Notice that the γ value recommended by IRACE is different for KS and KO. Likely due to the policy employed for constructing the kernel set and buckets, KS requires a larger bucket size compared to KO. This suggests that KS increases the bucket size to offset a less efficient sorting of variables, a feature achieved more effectively by KO.

It is noteworthy that the value of α employed by CG and KO is notably higher than that typically found in the CG literature. In the case of KO, this is because higher values of α facilitate an increase in the overall number of conflict-free augmenting paths generated by CG. Consequently, more promising variables are available during the KO initialization phase for constructing the kernel set. In the context of CG, a higher α value implies a greater number of iterations, providing more opportunities to enhance the quality of the final solution.

4.2. Computational results with a fixed time limit

The first computational comparison we carry out concerns the effectiveness of CG, KS, and KO when the same time limit is set for all of them. To this end, we run the three algorithms, on the same machine, with a time limit equal to 5, 20, 35, and 50 s. These values are chosen sufficiently small, with respect to the average computational times observed for the three algorithms on large instances (see Table 4), to ensure that they affect their effectiveness. The results of this comparison are shown in Fig. 7 in which the graphics show how the quality of the solutions is influenced by the computational time provided to the algorithms. The x-axis reports the set of all instances, while the y-axis shows the percentage gap from the best solution. This gap is computed with the formula $100 \times \frac{best-alg}{best}$, where *best* is equal to the highest value found by the three algorithms within the imposed time limit, while *alg* is the solution value of the considered algorithm. To better highlight the impact of the time limit, for each algorithm we sort the instances according to the percentage gap, in non-decreasing order. This means that, on the x-axis, the sequence of instances can be different for the three algorithms but this is not relevant for the purposes of the test. Finally, we recall here that the starting solution of KO is the feasible solution provided by CG (see Kernousel-Initialization). Therefore, if CG returns a feasible solution, within the time limit, then KO will return a solution at least as good as that of the CG.

From the result in Fig. 7(a) we observe that, with a time limit of 5 s, the results of CG and KO are very similar. Indeed, CG finds the best

Table 1
Characteristics of the test instances.

ID	n	p	d	I	ID	n	p	d	I	ID	n	p	d	I	ID	n	p	d	I
1	40	30%	30%	[10, 15]	41	50	40%	30%	[10, 15]	81	60	50%	30%	[10, 15]	121	70	60%	30%	[10, 15]
2	40	30%	30%	[15, 20]	42	50	40%	30%	[15, 20]	82	60	50%	30%	[15, 20]	122	70	60%	30%	[15, 20]
3	40	30%	40%	[10, 15]	43	50	40%	40%	[10, 15]	83	60	50%	40%	[10, 15]	123	70	60%	40%	[10, 15]
4	40	30%	40%	[15, 20]	44	50	40%	40%	[15, 20]	84	60	50%	40%	[15, 20]	124	70	60%	40%	[15, 20]
5	40	30%	50%	[10, 15]	45	50	40%	50%	[10, 15]	85	60	50%	50%	[10, 15]	125	70	60%	50%	[10, 15]
6	40	30%	50%	[15, 20]	46	50	40%	50%	[15, 20]	86	60	50%	50%	[15, 20]	126	70	60%	50%	[15, 20]
7	40	30%	60%	[10, 15]	47	50	40%	60%	[10, 15]	87	60	50%	60%	[10, 15]	127	70	60%	60%	[10, 15]
8	40	30%	60%	[15, 20]	48	50	40%	60%	[15, 20]	88	60	50%	60%	[15, 20]	128	70	60%	60%	[15, 20]
9	40	40%	30%	[10, 15]	49	50	50%	30%	[10, 15]	89	60	60%	30%	[10, 15]	129	80	30%	30%	[10, 15]
10	40	40%	30%	[15, 20]	50	50	50%	30%	[15, 20]	90	60	60%	30%	[15, 20]	130	80	30%	30%	[15, 20]
11	40	40%	40%	[10, 15]	51	50	50%	40%	[10, 15]	91	60	60%	40%	[10, 15]	131	80	30%	40%	[10, 15]
12	40	40%	40%	[15, 20]	52	50	50%	40%	[15, 20]	92	60	60%	40%	[15, 20]	132	80	30%	40%	[15, 20]
13	40	40%	50%	[10, 15]	53	50	50%	50%	[10, 15]	93	60	60%	50%	[10, 15]	133	80	30%	50%	[10, 15]
14	40	40%	50%	[15, 20]	54	50	50%	50%	[15, 20]	94	60	60%	50%	[15, 20]	134	80	30%	50%	[15, 20]
15	40	40%	60%	[10, 15]	55	50	50%	60%	[10, 15]	95	60	60%	60%	[10, 15]	135	80	30%	60%	[10, 15]
16	40	40%	60%	[15, 20]	56	50	50%	60%	[15, 20]	96	60	60%	60%	[15, 20]	136	80	30%	60%	[15, 20]
17	40	50%	30%	[10, 15]	57	50	60%	30%	[10, 15]	97	70	30%	30%	[10, 15]	137	80	40%	30%	[10, 15]
18	40	50%	30%	[15, 20]	58	50	60%	30%	[15, 20]	98	70	30%	30%	[15, 20]	138	80	40%	30%	[15, 20]
19	40	50%	40%	[10, 15]	59	50	60%	40%	[10, 15]	99	70	30%	40%	[10, 15]	139	80	40%	40%	[10, 15]
20	40	50%	40%	[15, 20]	60	50	60%	40%	[15, 20]	100	70	30%	40%	[15, 20]	140	80	40%	40%	[15, 20]
21	40	50%	50%	[10, 15]	61	50	60%	50%	[10, 15]	101	70	30%	50%	[10, 15]	141	80	40%	50%	[10, 15]
22	40	50%	50%	[15, 20]	62	50	60%	50%	[15, 20]	102	70	30%	50%	[15, 20]	142	80	40%	50%	[15, 20]
23	40	50%	60%	[10, 15]	63	50	60%	60%	[10, 15]	103	70	30%	60%	[10, 15]	143	80	40%	60%	[10, 15]
24	40	50%	60%	[15, 20]	64	50	60%	60%	[15, 20]	104	70	30%	60%	[15, 20]	144	80	40%	60%	[15, 20]
25	40	60%	30%	[10, 15]	65	60	30%	30%	[10, 15]	105	70	40%	30%	[10, 15]	145	80	50%	30%	[10, 15]
26	40	60%	30%	[15, 20]	66	60	30%	30%	[15, 20]	106	70	40%	30%	[15, 20]	146	80	50%	30%	[15, 20]
27	40	60%	40%	[10, 15]	67	60	30%	40%	[10, 15]	107	70	40%	40%	[10, 15]	147	80	50%	40%	[10, 15]
28	40	60%	40%	[15, 20]	68	60	30%	40%	[15, 20]	108	70	40%	40%	[15, 20]	148	80	50%	40%	[15, 20]
29	40	60%	50%	[10, 15]	69	60	30%	50%	[10, 15]	109	70	40%	50%	[10, 15]	149	80	50%	50%	[10, 15]
30	40	60%	50%	[15, 20]	70	60	30%	50%	[15, 20]	110	70	40%	50%	[15, 20]	150	80	50%	50%	[15, 20]
31	40	60%	60%	[10, 15]	71	60	30%	60%	[10, 15]	111	70	40%	60%	[10, 15]	151	80	50%	60%	[10, 15]
32	40	60%	60%	[15, 20]	72	60	30%	60%	[15, 20]	112	70	40%	60%	[15, 20]	152	80	50%	60%	[15, 20]
33	50	30%	30%	[10, 15]	73	60	40%	30%	[10, 15]	113	70	50%	30%	[10, 15]	153	80	60%	30%	[10, 15]
34	50	30%	30%	[15, 20]	74	60	40%	30%	[15, 20]	114	70	50%	30%	[15, 20]	154	80	60%	30%	[15, 20]
35	50	30%	40%	[10, 15]	75	60	40%	40%	[10, 15]	115	70	50%	40%	[10, 15]	155	80	60%	40%	[10, 15]
36	50	30%	40%	[15, 20]	76	60	40%	40%	[15, 20]	116	70	50%	40%	[15, 20]	156	80	60%	40%	[15, 20]
37	50	30%	50%	[10, 15]	77	60	40%	50%	[10, 15]	117	70	50%	50%	[10, 15]	157	80	60%	50%	[10, 15]
38	50	30%	50%	[15, 20]	78	60	40%	50%	[15, 20]	118	70	50%	50%	[15, 20]	158	80	60%	50%	[15, 20]
39	50	30%	60%	[10, 15]	79	60	40%	60%	[10, 15]	119	70	50%	60%	[10, 15]	159	80	60%	60%	[10, 15]
40	50	30%	60%	[15, 20]	80	60	40%	60%	[15, 20]	120	70	50%	60%	[15, 20]	160	80	60%	60%	[15, 20]

Table 2
Parameter settings.

Parameter	Description	Range values	CG	KS	KO
λ	Kernel set size	{150,200,250}		200	200
γ	Bucket size	{200,250,300}		250	200
α	Regulates the numberof iterations	{20,30,40}	40		40
β	Regulates the carouselstart solution	{20%,30%,40%}	40%		40%

solution in 106 out of 160 instances while KO in 108 instances and their maximum percentage gap (peak) is equal to 60%. These results show that 5 s represents a time too small to allow KO to improve the solution provided by CG. KS finds the best solution in 106 out of 160 instances and its peak is equal to 62%. However, the graphic shows that there are several (39) instances where the percentage gap of KS is greater than the one of CG and this means that even for this algorithm 5 s are not enough to overcome CG. It is worth noting that there are several instances where the three algorithms show a percentage gap greater than 0%. This means that the best solution is not found always by the same algorithm but that all of them contribute to providing this best solution. This behavior changes as the time limit increases because the KO algorithm becomes the predominant algorithm in finding the best solution. By increasing the time limit to 20 s (Fig. 7(b)), the effectiveness of the three algorithms significantly changes. With respect to the results obtained with the time limit of 10 s, we observe that KS finds the best solution in 108 out of 160 instances while CG in 98 instances. Moreover, the peak of CG remains equal to 60% while the one of KS decreases to ∼46.5%. Finally, only 8 times the percentage gap of KS exceeds the one of CG. Despite a time limit of 20 s only, the results of KO are much better than the other two algorithms. Indeed, it finds the best solution in 140 out of 160 instances, its peak is equal to ∼42.4% and its percentage gap never exceeds the one of KS. Notice that the percentage gap peaks can change among the subfigures because as the time limit increases the algorithms become more effective and then the best solutions, found for each instance, can change.

The results obtained by CG and KS with a time limit of 35 s (Fig. 7(c)) are very similar to the ones obtained within 20 s. There is only to report a reduction of the number of best solutions found by CG from 98 to 95 and a lightly better peak of KS that decreases to ∼42.3%. The quality of the solutions provided by KO within 35 s significantly improves. Indeed, this algorithm returns the best solution for 147 instances and its peak is decreased to 8.33% which represents a relevant improvement with respect to the 42.4% observed with the time limit of 20 s. The results of the three algorithms are essentially confirmed by increasing the time limit to 50 s (Fig. 7(d)). There are some further improvements of KO which returns the best solution for 151 instances and its peak decreases to 5.68% showing that this

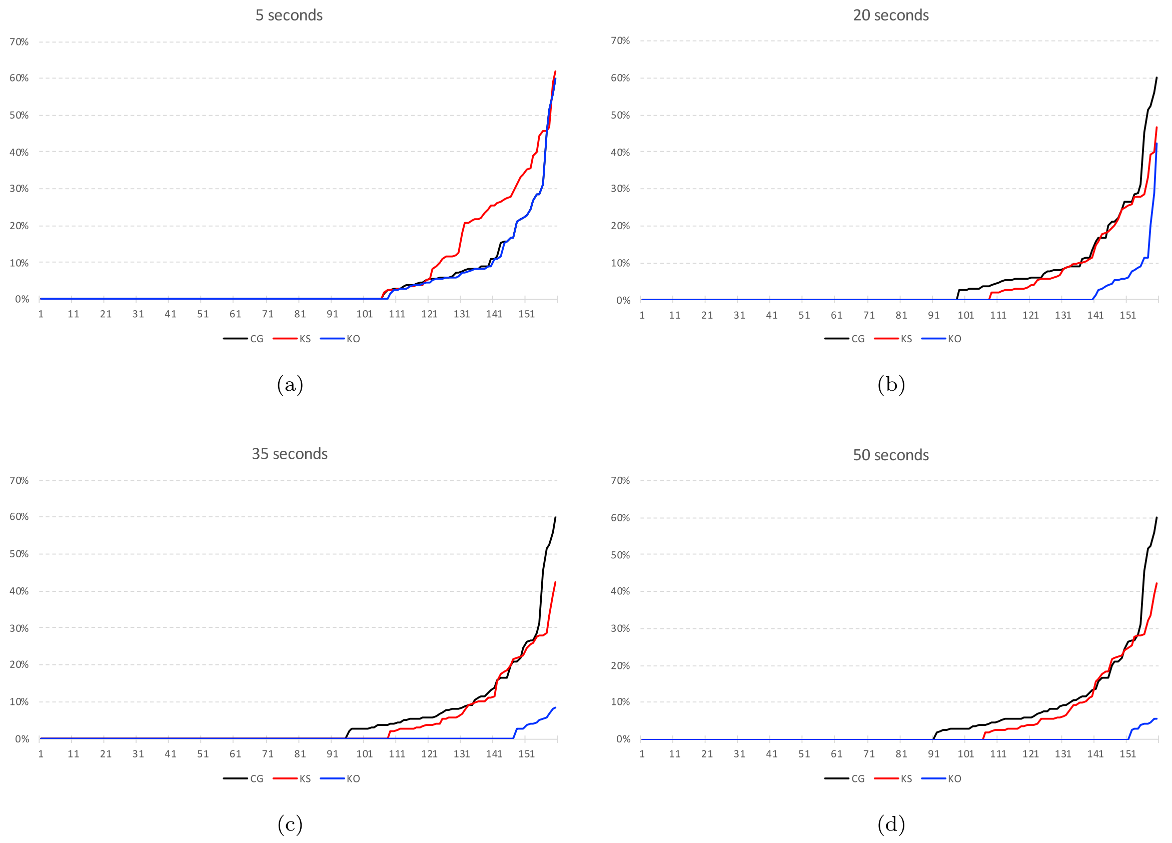


Fig. 7. Comparison of the solution quality of CG, KS, and KO: time limit fixed to 5, 20, 35, and 50 s, respectively.

algorithm remains very effective even in the nine instances where it does not find the best solution. With respect to the previous time limit of 35 s, the peaks of CG and KS do not change while the best solutions found are decreased to 91 and 106, respectively.

4.3. Computational results and comparison with the state-of-the-art approach

In this section, we compare the results and performance of CG, KS, and KO against the ones of the best-performing approach proposed in Şuvak et al. (2020), i.e., a Benders Decomposition (BD) algorithm based on Formulation 2. Since the computational results of the BD algorithm, reported in Şuvak et al. (2020), refer to tests carried out by using a `Intel Xeon CPU E5-2687 W 3.10 GHz` processor with 3.74 gigaflops, while we conducted the numerical experiments related to all the methods presented in this work using a `Intel(R) Core(TM) i7-3770 CPU @ 3.40 GHz` processor with 4.74 gigaflops, in order to have a fair comparative study, the CPU time reported in Şuvak et al. (2020) has been scaled according to the Whetstone benchmark.[1] In particular, we scaled all the running times of BD by dividing them for the conversion ratio $\mu = \frac{4.74}{3.74} \approx 1.27$. Clearly, by applying this conversion, the one-hour time limit, imposed by Şuvak et al. (2020) corresponds to 2840.51 s in our setting.

Computational results are shown in , reporting information about the tests performed on the Small and Large instances, respectively. The first three columns of report the instance ID, the instance name, and the value of the best-known feasible solution (*Best*) in the literature, respectively. In particular, for each instance, the *Best* column reports the highest among the solution values obtained by all the approaches presented in Şuvak et al. (2020). When the reported value is marked with an asterisk, the associated solution has been certified to be optimal. Then, for each of the four compared approaches, namely BD, CG, KS and KO, three additional columns are reported, indicating: the value associated with the solution identified by that approach (*Value*), the execution time in seconds (*Time*), and the percentage gap measuring how far, in terms of value, the identified solution is from the best-known one in the literature (*Gap*). Negative gap values indicate improvements with respect to the best solution values available in the literature and are marked in bold. On Small instances, the average percentage gap of the solutions identified by the CG, with respect to the best-known solutions, is 4.77%, the KS approach achieves a gap of 3.47%, while the KO approach achieves an average gap of 0.33%. On Large instances, the average percentage gap achieved by CG, KS, and KO are 4.66%, 5.97%, and −1.18%, respectively. The obtained results show the increase in the solution quality resulting from the combination of CG and KS. On the other side, the state-of-the-art BD method reports an average percentage gap, with respect to the best-known solution, of 0.64% on the Small Instances and 4.47% on Large Instances. The average percentage gap values produced by the KO approach are lower than the ones yielded by BD on both the classes of instances, even though such a difference is more evident on Large Instances. Furthermore, concerning the computational times, the KO requires, on average, 54.22 s to solve a small instance and 198.60 to solve a large one. Thus, the proposed approach is faster than the state-of-the-art method, which requires, on average, 388.32 and 1660.93 s to solve a small and a large instance, respectively. Nevertheless, the KO approach improves the best-known solution value available in the literature for 2 small and 13 large instances, while it identifies a

[1] http://gene.disi.unitn.it/test/cpu_list.php.

Table 3
Computational results on small instances.

ID	Best	BD			CG			KS			KO		
		Value	Time	Gap	Value	Time	Gap	Value	Time	Gap	Value	Time	Gap
1	37*	37	12.89	0.00%	37	7.99	0.00%	37	4.64	0.00%	37	13.09	0.00%
2	49*	49	4.03	0.00%	37	5.72	24.49%	49	3.73	0.00%	49	9.39	0.00%
3	24*	24	8.67	0.00%	24	1.88	0.00%	24	6.31	0.00%	24	7.58	0.00%
4	33*	33	4.97	0.00%	33	4.57	0.00%	33	4.65	0.00%	33	9.56	0.00%
5	12*	12	6.77	0.00%	12	1.49	0.00%	12	12.09	0.00%	12	7.44	0.00%
6	34*	34	6.01	0.00%	34	2.51	0.00%	34	4.92	0.00%	34	8.40	0.00%
7	13*	13	14.96	0.00%	13	1.98	0.00%	13	6.67	0.00%	13	8.35	0.00%
8	15*	15	20.96	0.00%	15	1.73	0.00%	15	5.53	0.00%	15	7.65	0.00%
9	35*	35	14.98	0.00%	34	6.63	2.86%	34	6.84	2.86%	35	12.70	0.00%
10	51*	51	178.84	0.00%	51	3.01	0.00%	51	19.57	0.00%	51	16.98	0.00%
11	26*	26	20.46	0.00%	26	4.55	0.00%	26	10.55	0.00%	26	14.29	0.00%
12	50*	50	17.65	0.00%	39	4.77	22.00%	50	14.83	0.00%	50	14.40	0.00%
13	13*	13	30.50	0.00%	13	2.54	0.00%	13	12.62	0.00%	13	12.91	0.00%
14	19*	19	22.28	0.00%	19	2.25	0.00%	19	9.97	0.00%	19	11.68	0.00%
15	25*	25	26.00	0.00%	25	4.32	0.00%	25	9.74	0.00%	25	14.48	0.00%
16	19*	19	29.38	0.00%	19	2.19	0.00%	19	11.32	0.00%	19	13.53	0.00%
17	43*	43	143.72	0.00%	34	13.53	20.93%	35	37.51	18.60%	38	55.73	11.63%
18	55*	55	25.49	0.00%	55	10.49	0.00%	55	8.21	0.00%	55	19.85	0.00%
19	33*	33	29.14	0.00%	33	9.72	0.00%	32	13.23	3.03%	33	23.38	0.00%
20	35*	35	40.50	0.00%	33	14.01	5.71%	33	11.26	5.71%	33	26.57	5.71%
21	27*	27	54.10	0.00%	26	6.40	3.70%	27	15.82	0.00%	27	21.73	0.00%
22	34*	34	37.69	0.00%	32	6.17	5.88%	34	14.51	0.00%	34	19.91	0.00%
23	15*	15	25.41	0.00%	13	4.03	13.33%	14	13.99	6.67%	15	17.32	0.00%
24	34*	34	26.44	0.00%	34	4.61	0.00%	34	13.80	0.00%	34	17.01	0.00%
25	45*	45	518.28	0.00%	41	15.31	8.89%	45	14.58	0.00%	45	33.95	0.00%
26	80*	80	78.77	0.00%	69	16.91	13.75%	71	26.03	11.25%	80	38.00	0.00%
27	30*	30	130.41	0.00%	25	4.74	16.67%	24	37.98	20.00%	25	42.98	16.67%
28	48*	48	130.58	0.00%	33	16.76	31.25%	48	17.57	0.00%	48	35.06	0.00%
29	26*	26	56.46	0.00%	24	7.56	7.69%	26	17.69	0.00%	26	25.29	0.00%
30	36*	36	64.61	0.00%	35	5.50	2.78%	36	20.47	0.00%	36	26.94	0.00%
31	24*	24	19.43	0.00%	24	3.51	0.00%	24	22.55	0.00%	24	27.80	0.00%
32	37*	37	7.29	0.00%	37	7.03	0.00%	37	19.19	0.00%	37	26.22	0.00%
33	37*	37	17.86	0.00%	37	10.88	0.00%	35	10.08	5.41%	37	20.46	0.00%
34	48*	48	34.49	0.00%	48	10.09	0.00%	47	10.73	2.08%	48	21.90	0.00%
35	24*	24	31.94	0.00%	20	2.61	16.67%	22	10.03	8.33%	22	13.01	8.33%
36	33*	33	46.59	0.00%	18	3.06	45.45%	33	11.06	0.00%	33	15.86	0.00%
37	21*	21	32.94	0.00%	21	4.30	0.00%	21	11.14	0.00%	21	16.51	0.00%
38	33*	33	31.26	0.00%	30	5.57	9.09%	33	12.80	0.00%	33	19.23	0.00%
39	13*	13	20.27	0.00%	13	3.01	0.00%	13	11.97	0.00%	13	14.66	0.00%
40	37*	37	10.40	0.00%	37	4.29	0.00%	37	10.49	0.00%	37	15.30	0.00%
41	37*	37	616.52	0.00%	36	9.63	2.70%	28	30.96	24.32%	36	31.54	2.70%
42	53*	53	124.67	0.00%	53	10.61	0.00%	50	20.46	5.66%	53	25.60	0.00%
43	35*	35	200.12	0.00%	34	8.47	2.86%	25	23.58	28.57%	35	33.97	0.00%
44	38*	38	58.04	0.00%	37	5.01	2.63%	37	18.72	2.63%	38	24.92	0.00%
45	21*	21	78.21	0.00%	12	5.18	42.86%	14	26.22	33.33%	21	32.61	0.00%
46	18*	18	47.58	0.00%	18	6.31	0.00%	17	16.60	5.56%	18	25.42	0.00%
47	24*	24	15.91	0.00%	24	11.06	0.00%	24	22.86	0.00%	24	32.10	0.00%
48	37*	37	26.18	0.00%	37	10.13	0.00%	37	25.10	0.00%	37	36.78	0.00%
49	40*	40	381.28	0.00%	40	21.47	0.00%	39	35.33	2.50%	40	83.74	0.00%
50	57*	57	138.01	0.00%	55	16.15	3.51%	55	35.86	3.51%	55	49.09	3.51%
51	40*	40	177.09	0.00%	38	13.59	5.00%	40	29.53	0.00%	40	41.50	0.00%
52	53*	53	145.24	0.00%	53	6.53	0.00%	53	27.91	0.00%	53	37.14	0.00%
53	25*	25	70.21	0.00%	25	17.80	0.00%	25	29.64	0.00%	25	46.92	0.00%
54	38*	38	66.33	0.00%	38	4.81	0.00%	38	28.44	0.00%	38	33.16	0.00%
55	25*	25	41.72	0.00%	11	5.04	56.00%	25	31.12	0.00%	25	34.45	0.00%
56	18*	18	18.84	0.00%	18	5.25	0.00%	18	28.07	0.00%	18	34.61	0.00%
57	49	45	2840.51	8.16%	48	34.11	2.04%	48	71.16	2.04%	49	192.28	0.00%
58	73*	73	2147.34	0.00%	73	23.63	0.00%	73	89.62	0.00%	73	153.32	0.00%
59	38*	38	318.29	0.00%	34	13.77	10.53%	37	40.27	2.63%	38	67.54	0.00%
60	51*	51	734.78	0.00%	51	22.81	0.00%	49	56.14	3.92%	51	96.11	0.00%
61	39*	39	133.29	0.00%	38	13.38	2.56%	38	39.59	2.56%	38	53.49	2.56%
62	37*	37	198.06	0.00%	37	17.98	0.00%	37	41.90	0.00%	37	61.29	0.00%
63	26*	26	72.82	0.00%	26	14.98	0.00%	23	49.41	11.54%	26	75.00	0.00%
64	33*	33	62.96	0.00%	33	12.82	0.00%	33	53.83	0.00%	33	66.10	0.00%
65	36*	36	919.26	0.00%	34	11.47	5.56%	26	19.76	27.78%	36	32.72	0.00%
66	53*	53	451.98	0.00%	50	11.65	5.66%	50	16.03	5.66%	53	29.69	0.00%
67	22*	22	60.62	0.00%	21	5.03	4.55%	22	16.68	0.00%	22	24.08	0.00%
68	34*	34	24.20	0.00%	33	16.97	2.94%	32	14.29	5.88%	34	33.39	0.00%
69	27*	27	18.02	0.00%	27	11.84	0.00%	27	25.45	0.00%	27	38.82	0.00%
70	33*	33	33.98	0.00%	33	6.31	0.00%	33	21.05	0.00%	33	32.04	0.00%
71	12*	12	21.56	0.00%	12	5.13	0.00%	12	23.77	0.00%	12	32.00	0.00%

(*continued on next page*)

Table 3 (*continued*).

72	19*	19	23.19	0.00%	17	10.80	10.53%	18	26.32	5.26%	18	41.86	5.26%
73	39	39	2840.51	0.00%	39	22.51	0.00%	35	56.62	10.26%	39	108.13	0.00%
74	64*	64	774.87	0.00%	64	20.40	0.00%	54	40.06	15.63%	64	56.51	0.00%
75	26*	26	122.32	0.00%	26	24.82	0.00%	26	27.17	0.00%	26	55.92	0.00%
76	38*	38	651.71	0.00%	37	11.88	2.63%	37	39.55	2.63%	37	52.99	2.63%
77	29*	29	94.07	0.00%	29	24.20	0.00%	28	45.88	3.45%	29	76.97	0.00%
78	37*	37	63.34	0.00%	37	18.99	0.00%	37	42.61	0.00%	37	61.93	0.00%
79	14*	14	57.10	0.00%	12	7.85	14.29%	14	42.66	0.00%	14	61.74	0.00%
80	37*	37	39.12	0.00%	34	7.14	8.11%	37	36.61	0.00%	37	51.24	0.00%
81	53	40	2840.51	24.53%	53	51.01	0.00%	41	113.62	22.64%	53	214.70	0.00%
82	68	52	2840.51	23.53%	64	29.16	5.88%	67	76.26	1.47%	68	171.98	0.00%
83	40*	40	1429.18	0.00%	40	15.01	0.00%	36	89.42	10.00%	40	118.88	0.00%
84	39*	39	595.57	0.00%	39	16.25	0.00%	39	64.92	0.00%	39	75.20	0.00%
85	26*	26	534.17	0.00%	26	21.25	0.00%	26	78.57	0.00%	26	85.30	0.00%
86	37*	37	213.58	0.00%	35	14.57	5.41%	37	60.25	0.00%	36	71.67	2.70%
87	25*	25	75.77	0.00%	25	18.75	0.00%	25	77.08	0.00%	25	86.69	0.00%
88	19*	19	140.48	0.00%	18	9.53	5.26%	19	59.84	0.00%	19	66.11	0.00%
89	53	53	2840.51	0.00%	54	38.60	**−1.89%**	50	179.79	5.66%	55	317.89	**−3.77%**
90	54	51	2840.51	5.56%	54	38.84	0.00%	52	224.51	3.70%	68	319.71	**−25.93%**
91	41*	41	2088.25	0.00%	41	56.48	0.00%	41	98.24	0.00%	41	170.25	0.00%
92	53*	53	2270.18	0.00%	53	32.39	0.00%	53	96.15	0.00%	53	142.38	0.00%
93	27*	27	777.61	0.00%	26	11.64	3.70%	27	81.54	0.00%	27	103.82	0.00%
94	53*	53	439.42	0.00%	53	39.29	0.00%	53	81.48	0.00%	53	122.69	0.00%
95	24*	24	191.54	0.00%	24	23.32	0.00%	24	83.42	0.00%	24	107.16	0.00%
96	33*	33	217.88	0.00%	32	11.99	3.03%	33	84.85	0.00%	33	95.38	0.00%
AVG			**388.32**	**0.64%**		**12.54**	**4.77%**		**35.97**	**3.47%**		**54.22**	**0.33%**
#Impr.							**1**			**0**			**2**
#NBest							**54**			**60**			**86**

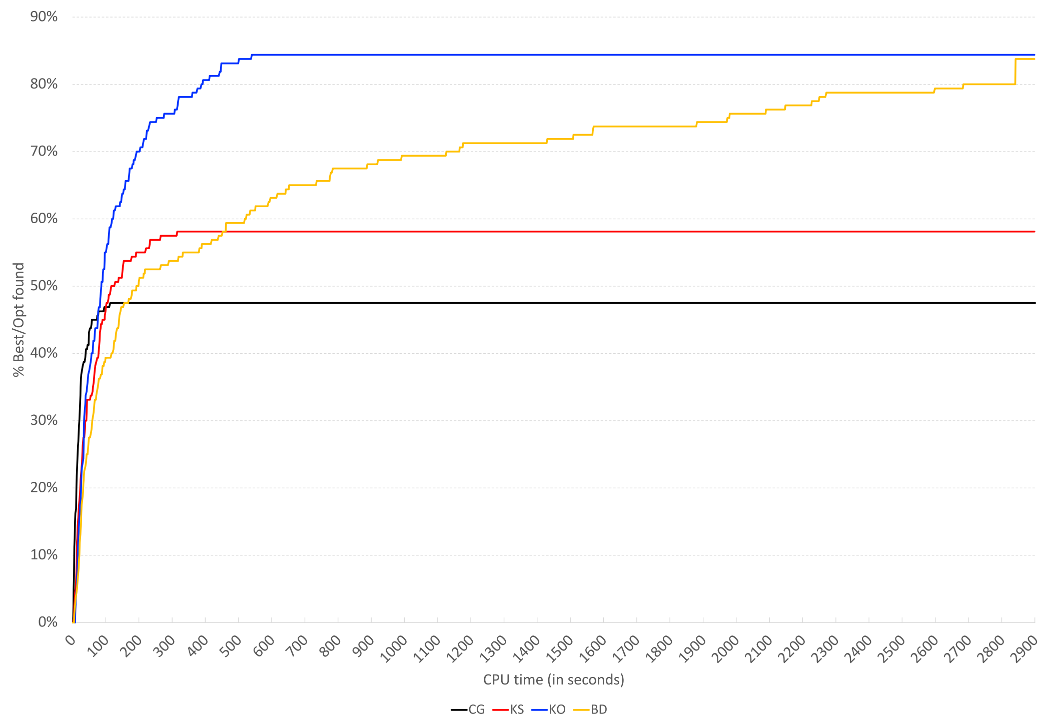


Fig. 8. Percentage of best/optimal solutions found by algorithms within the computational time reported on the *x*-axis.

solution whose value corresponds to the new best-known value, i.e., the one considering also the results computed in this work, for 86 out of 96 small instances and 49 out of 64 large ones.

Fig. 8 depicts a cumulative chart with time, expressed in seconds, on the *x*-axis and the percentage of best/optimal solutions, identified within a given time, on the *y*-axis. Notice that the best/optimal solutions considered in this chart are those having the highest values, identified by comparing all the approaches presented in Şuvak et al. (2020) and all the ones presented in this work. The curves show that, in general, the KO approach is able to identify a larger number of best solutions in less time, compared with the other approaches. More in detail, in the first 80 s, the CG algorithm identifies more best/optimal solutions than the remaining approaches but is not able to identify further ones after about 115 s, identifying at the end 76 out of 160

Table 4
Computational results on large instances.

ID	Best	BD			CG			KS			KO		
		Value	Time	Gap	Value	Time	Gap	Value	Time	Gap	Value	Time	Gap
97	39*	39	991.9	0.00%	36	32.07	7.69%	39	37.69	0.00%	39	96.56	0.00%
98	47*	35	2840.51	25.53%	35	15.10	25.53%	34	35.79	27.66%	47	82.05	0.00%
99	23*	23	139.57	0.00%	22	20.03	4.35%	20	29.70	13.04%	22	52.05	4.35%
100	36*	36	88.02	0.00%	36	14.16	0.00%	36	31.49	0.00%	36	45.81	0.00%
101	25*	25	84.43	0.00%	12	6.86	52.00%	25	35.06	0.00%	25	42.23	0.00%
102	19*	19	89.68	0.00%	16	6.32	15.79%	19	42.41	0.00%	19	49.85	0.00%
103	14*	14	46.59	0.00%	13	5.59	7.14%	14	41.26	0.00%	14	53.33	0.00%
104	20*	20	48.86	0.00%	20	6.11	0.00%	20	37.94	0.00%	20	44.53	0.00%
105	48	45	2840.51	6.25%	48	50.22	0.00%	45	107.13	6.25%	56	275.08	**−16.67%**
106	65*	52	2840.51	20.00%	50	20.53	23.08%	36	43.96	44.62%	53	103.91	18.46%
107	31*	31	887.9	0.00%	31	20.12	0.00%	31	80.90	0.00%	31	82.76	0.00%
108	36*	36	1569.11	0.00%	36	27.25	0.00%	35	51.11	2.78%	36	84.13	0.00%
109	26*	26	388.1	0.00%	24	22.57	7.69%	26	70.50	0.00%	26	110.79	0.00%
110	33*	33	461.74	0.00%	32	18.33	3.03%	32	71.11	3.03%	32	98.09	3.03%
111	25*	25	168.39	0.00%	25	15.13	0.00%	25	73.64	0.00%	25	90.32	0.00%
112	18*	18	116.5	0.00%	16	8.15	11.11%	18	67.82	0.00%	18	78.83	0.00%
113	39	34	2840.51	12.82%	40	49.63	**−2.56%**	41	190.97	**−5.13%**	41	500.48	**−5.13%**
114	50	48	2840.51	4.00%	52	44.96	**−4.00%**	49	124.06	2.00%	52	307.15	**−4.00%**
115	35*	35	2226.85	0.00%	28	16.87	20.00%	35	79.48	0.00%	35	109.29	0.00%
116	55	55	2840.51	0.00%	54	44.49	1.82%	54	100.17	1.82%	55	252.09	0.00%
117	25*	25	1979.85	0.00%	21	21.47	16.00%	24	103.93	4.00%	24	144.46	4.00%
118	50*	50	550.95	0.00%	50	11.13	0.00%	50	99.93	0.00%	50	117.00	0.00%
119	24*	24	331.96	0.00%	22	12.62	8.33%	24	101.54	0.00%	24	123.49	0.00%
120	19*	19	641.12	0.00%	18	10.28	5.26%	19	107.49	0.00%	19	129.60	0.00%
121	50	41	2840.51	18.00%	54	48.12	**−8.00%**	41	113.82	18.00%	54	360.24	**−8.00%**
122	72	55	2840.51	23.61%	86	77.92	**−19.44%**	72	131.45	0.00%	86	412.12	**−19.44%**
123	35	34	2840.51	2.86%	34	36.35	2.86%	37	115.49	**−5.71%**	37	189.33	**−5.71%**
124	54	52	2840.51	3.70%	52	39.03	3.70%	52	110.04	3.70%	52	184.84	3.70%
125	32*	32	1880.64	0.00%	32	14.21	0.00%	32	138.87	0.00%	32	170.44	0.00%
126	53*	53	2597.75	0.00%	51	43.80	3.77%	46	131.53	13.21%	51	176.84	3.77%
127	25*	25	461.39	0.00%	25	15.00	0.00%	25	151.39	0.00%	25	183.79	0.00%
128	36*	36	523.14	0.00%	34	45.97	5.56%	36	151.84	0.00%	36	203.22	0.00%
129	38	31	2840.51	18.42%	33	55.26	13.16%	33	107.56	13.16%	33	183.82	13.16%
130	69*	69	2249.74	0.00%	69	19.75	0.00%	54	56.36	21.74%	69	95.32	0.00%
131	23*	23	1566.41	0.00%	22	9.70	4.35%	22	49.51	4.35%	22	60.30	4.35%
132	37*	37	589.97	0.00%	34	23.32	8.11%	33	65.05	10.81%	34	85.46	8.11%
133	15*	15	265.75	0.00%	12	7.83	20.00%	15	63.03	0.00%	15	76.84	0.00%
134	50*	50	155	0.00%	50	25.89	0.00%	36	66.61	28.00%	50	101.64	0.00%
135	15*	15	98.34	0.00%	12	19.95	20.00%	15	62.44	0.00%	15	90.40	0.00%
136	34*	34	124.6	0.00%	32	15.88	5.88%	34	67.85	0.00%	34	91.36	0.00%
137	54	54	2840.51	0.00%	54	37.55	0.00%	54	127.91	0.00%	54	157.36	0.00%
138	52	49	2840.51	5.77%	65	38.01	**−25.00%**	53	115.20	**−1.92%**	65	223.33	**−25.00%**
139	36*	29	2840.51	19.44%	27	24.81	25.00%	29	123.46	19.44%	29	141.90	19.44%
140	50	50	2840.51	0.00%	45	38.92	10.00%	36	90.82	28.00%	50	148.17	0.00%
141	28*	28	783.28	0.00%	23	24.98	17.86%	26	99.21	7.14%	26	133.55	7.14%
142	50*	50	1165.51	0.00%	50	48.24	0.00%	50	108.47	0.00%	50	179.60	0.00%
143	26*	26	289.23	0.00%	25	27.22	3.85%	15	119.11	42.31%	26	158.24	0.00%
144	34*	34	419	0.00%	32	25.73	5.88%	34	113.30	0.00%	34	146.74	0.00%
145	48	40	2840.51	16.67%	50	71.19	**−4.17%**	49	248.71	**−2.08%**	56	442.11	**−16.67%**
146	51	48	2840.51	5.88%	64	73.11	**−25.49%**	52	194.31	**−1.96%**	64	447.31	**−25.49%**
147	39	38	2840.51	2.56%	38	67.22	2.56%	28	131.12	28.21%	38	234.84	2.56%
148	55	38	2840.51	30.91%	51	63.24	7.27%	38	137.75	30.91%	51	238.90	7.27%
149	25*	25	2840.51	0.00%	24	58.21	4.00%	25	153.08	0.00%	25	232.39	0.00%
150	35*	35	2683.31	0.00%	35	48.90	0.00%	35	148.92	0.00%	35	229.05	0.00%
151	23*	23	1125.09	0.00%	23	29.25	0.00%	14	180.39	39.13%	23	221.67	0.00%
152	33*	33	1175.32	0.00%	19	19.45	42.42%	33	177.44	0.00%	33	211.60	0.00%
153	41	41	2840.51	0.00%	61	112.36	**−48.78%**	47	401.11	**−14.63%**	61	539.29	**−48.78%**
154	81	68	2840.51	16.05%	85	155.51	**−4.94%**	88	314.64	**−8.64%**	85	656.44	**−4.94%**
155	37	33	2840.51	10.81%	37	78.72	0.00%	38	220.60	**−2.70%**	38	375.76	**−2.70%**
156	46	37	2840.51	19.57%	46	100.28	0.00%	53	232.28	**−15.22%**	53	447.04	**−15.22%**
157	38	38	2840.51	0.00%	38	57.12	0.00%	38	231.67	0.00%	38	315.09	0.00%
158	48	37	2840.51	22.92%	37	54.38	22.92%	37	228.36	22.92%	37	310.82	22.92%
159	26*	26	1972.66	0.00%	26	94.89	0.00%	25	270.39	3.85%	26	387.05	0.00%
160	36*	36	1508.82	0.00%	35	107.25	2.78%	36	264.82	0.00%	36	392.36	0.00%
AVG			**1660.93**	**4.47%**		**38.35**	**4.66%**		**121.58**	**5.97%**		**198.60**	**−1.18%**
#Impr.							**9**			**9**			**13**
#NBest							**22**			**33**			**49**

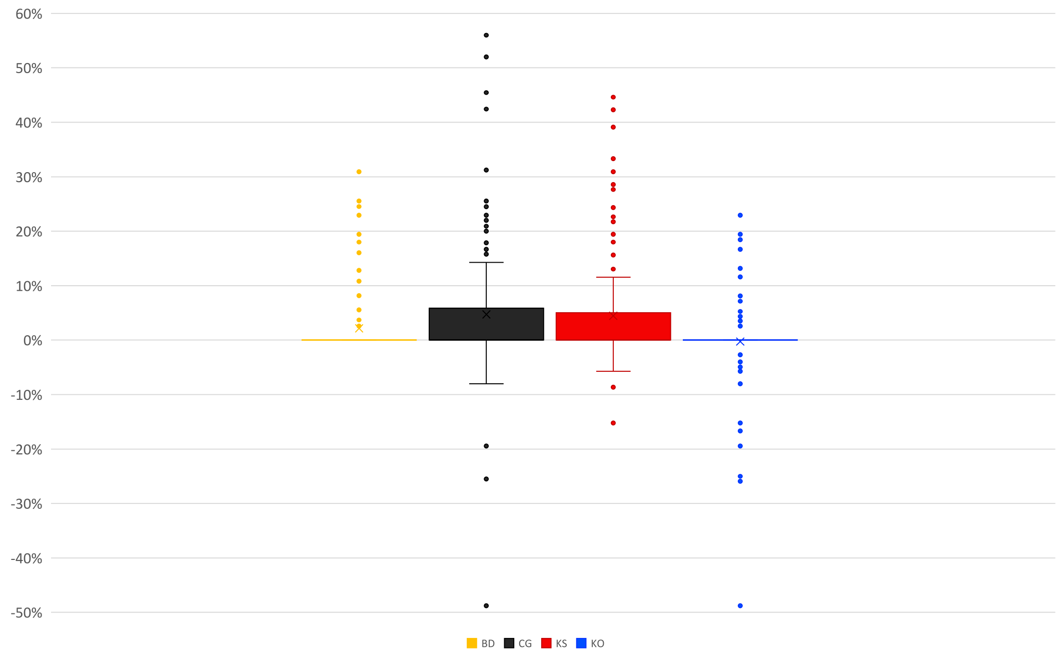


Fig. 9. Box plots of the percentage gaps of each algorithm with respect to the best-known solutions available in the literature.

best/optimal solutions. The pure KS finds, instead, approximately in the first 315 s, 93 out of 160 best/optimal solutions, after which it does not find others. On the other hand, the KO approach identifies or improves 135 best/optimal solutions, in about 540 s, while the BD approach solves 134 out of 160 instances to optimality, in about 2800 s.

4.4. *Statistical and sensitivity analysis of the results*

To further investigate the differences in performance of the four solution approaches, let us observe the box plots reported in Fig. 9, showing the distribution of the percentage gaps obtained by each method on both small and large instances.

The box plots associated with BD and KO are alike in that both their upper and lower whiskers lie on the 0% gap value, indicating low data variation, as for most of the instances (specifically the 85%), both BD and KO identify the best-known solutions available in the literature. The performances of CG and KS, on the other side, are characterized by a higher variation, with the former yielding a gap between 0% and 5.88%, and the latter leading to a gap between 0% and 5.03%, for about 70% of the instances. Although all the box plots have outliers, the ones detected by the KO box plot are mostly equally distributed between the positive and negative sides of the y-axis, while very few negative outliers have been identified for CG and KS. This reflects the fact that KO improves the value of the best-known solution in more cases w.r.t. the other approaches.

The performance of the KO approach has been further analyzed, in terms of efficiency and effectiveness, as a function of the size and characteristics of the instances. As it can be noticed by observing Fig. 10(a), the algorithm requires longer computational times as the number of nodes n, as well as the value of the arc density parameter p, increase. Fig. 10(b) shows, instead, the impact of the conflict density parameter d and the arc density parameter p, indicating that the higher the number of conflicts, the faster the resolution. Concerning the quality of the solutions, Figs. 11(a), 11(b) and 11(c) show the percentage gap trends detected at the variation of the number of nodes n, arc density value p and conflict density value d, respectively. As expected, the quality of the solution is higher, i.e., the average gap values are generally smaller and they often become negative, as the number of nodes and the arc density value increase. On the contrary, the trend is the opposite for the conflict density, with respect to which negative average gap values are observed for $d \leq 40\%$, while higher conflict density values are associated with average gap values between 0% and 1%.

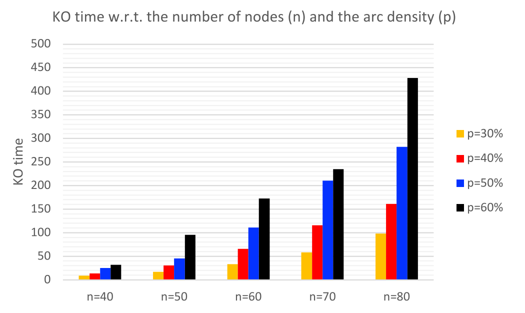
4.5. *Instance space analysis*

We used the Melbourne Algorithm Test Instance Library with Data Analytics (MATILDA) (Smith-Miles et al., 2020) tools to conduct a performance analysis and jointly evaluate the efficiency and effectiveness of the proposed approaches in the instance space of the MFPC. Instance Space Analysis (Smith-Miles & Muñoz, 2023) is a methodology that represents test instances as feature vectors, visualizes the whole instance space, and studies how the instance properties affect the performance of given algorithms. In recent years, such analysis has been successfully performed to evaluate search-based software testing techniques (Neelofar et al., 2022), as well as solution approaches for combinatorial problems such as the curriculum-based course timetabling (De Coster et al., 2022), the sports timetabling (Van Bulck et al., 2024), the two-dimensional bin-packing (Liu et al., 2024), the multi-demand multidimensional knapsack (Scherer et al., 2024), and the maximum flow (Alipour et al., 2023) problems, whose findings represent a starting point for our study on the MFPC.

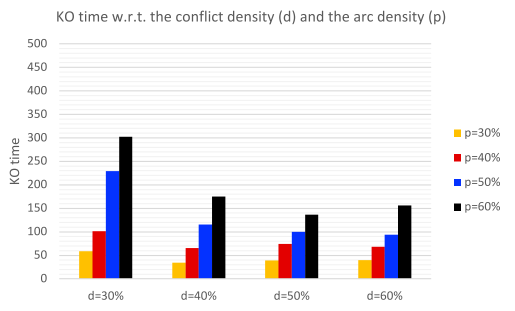
Performance measure. To evaluate the performance of the tested approaches, a problem-specific performance measure must be provided to MATILDA. For the MFPC, such a measure must consider both the solution quality and the computational efficiency of the tested method. Following the idea proposed in De Coster et al. (2022), we devised the following performance measure:

$$\Phi(g,r) = 1.01^{-\left(v\cdot g+\eta\cdot \frac{r}{t_{\max}}\right)}, \tag{5}$$

where g represents the percentage gap from the best-known solution value (obtained considering all the approaches proposed in Şuvak et al. (2020) and in this paper), r is the runtime of the considered algorithm in seconds, $t_{\max}$ is the largest runtime reported among all the tested

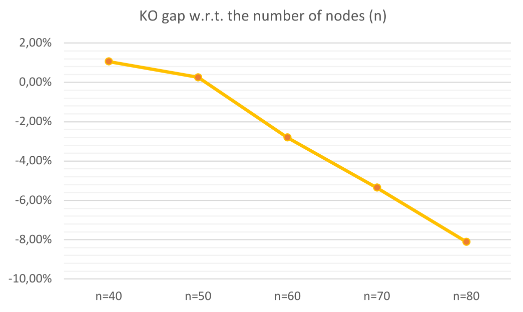


(a) Computational times of the KO algorithm as a function of the number of nodes (x-axis) and the arc density (series color) of the instances.

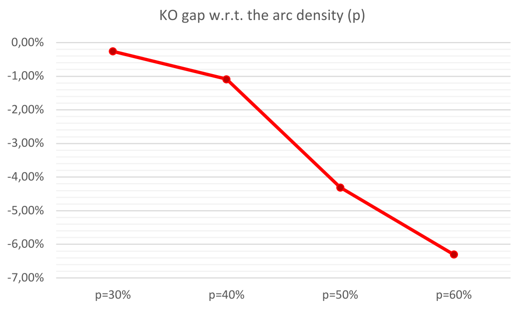


(b) Computational times of the KO algorithm as a function of the conflict density (x-axis) and the arc density (series color) of the instances.

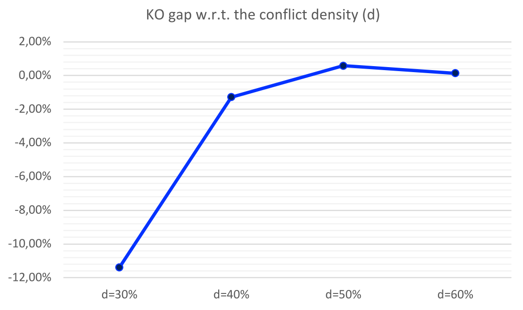
Fig. 10. Computational times charts, showing the impact of the characteristics of the instances on the efficiency of the KO algorithm. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



(a) Percentage gap as a function of the number of nodes (n).



(b) Percentage gap as a function of the arc density (p).



(c) Percentage gap as a function of the conflict density (d).

Fig. 11. Percentage gap charts, showing the impact of the characteristics of the instances on the effectiveness of the KO algorithm.

approaches, whereas ν and η are parameters used to balance the impact of the performance gap g and the runtime r. In our analysis, we decided to prioritize the performance gap g and to use the runtime r only as a secondary criterion. To this end, we set $\nu = 100$ and $\eta = 10$. The base 1.01 for the exponential function has been chosen such that the score is scaled down by 1% whenever the exponent increases by 1 unit. As a result, a higher value of the Φ function reflects an overall better performance of the considered algorithm on the tested instance. In our analysis, the Φ function has been used as performance measure with a goodness threshold of 0.05, meaning that an algorithm is considered "good" if its Φ value is within 5% from the one achieved by the best-performing algorithm on a given instance.

Features. Table 5 describes the collection of 18 features that we used to characterize the instance space of the MFPC. Some of them are drawn from the literature, as they were used in Alipour et al. (2023) for the classic maximum flow problem, while others are introduced to capture additional properties of the instance related to conflicts. The selected features incorporate information about arc and conflict density, the variability of node degrees, and the arc capacities. In the reported formula, given a sequence of values X, we denote by $\mu(X)$ and $\sigma(X)$ its associated mean and standard deviation, respectively. We further denote by $U = \langle u_{ij} : (i,j) \in A\rangle$ an arbitrary sequence of the capacities associated with the arcs of G, by $D = \langle |N(i)| : i \in V\rangle$ an analogous sequence of the node degrees, and by $C = \langle |\delta(i,j)| : (i,j) \in A\rangle$ a

Table 5
Features used to characterize the instance space of the MFPC.

Name	Description	Notation
MaxFlow	Maximum flow that can be sent from s to t by neglecting the conflict constraints	
ArcDensity	Number of existing over maximum possible number of arcs	$\frac{\lvert A\rvert}{\lvert V\rvert\times(\lvert V\rvert-1)}$
ConfDensity	Number of existing over maximum possible number of conflicts	$\frac{\sum_{(i,j)\in A}\lvert\delta(i,j)\rvert}{\lvert A\rvert\times(\lvert A\rvert-1)}$
Nodes	Number of nodes	$\lvert V\rvert$
AvNdDg	Average degree of a node	$\frac{\lvert A\rvert}{\lvert V\rvert}$
AvArcCap	Average capacity of an arc	$\mu(U)$
AvNdCap	Average node capacity	$\frac{\sum_{(i,j)\in A} u_{ij}}{\lvert V\rvert}$
AvgArcConf	Average number of conflicts per arc	$\mu(C)$
ScAvCap	Scaled average arc capacity	$\frac{\text{AvArcCap}}{\text{median}(U)}$
ScAvNdCap	Scaled average node capacity	$\frac{\text{AvNdCap}}{\text{median}(U)}$
ScCapDens	Scaled capacitated arc density	$\text{ScAvCap}\times\text{ArcDensity}$
ScRngCap	Scaled range of arc capacities	$\frac{\max(U)-\min(U)}{\text{AvArcCap}}$
cvNdDg	Coefficient of variation associated with the node degrees	$\frac{\sigma(D)}{\mu(D)}$
cvCap	Coefficient of variation associated with the arc capacities	$\frac{\sigma(U)}{\mu(U)}$
cvArcConf	Coefficient of variation associated with the number of conflict per arc	$\frac{\sigma(C)}{\mu(C)}$
PercLoCap	Percentage of arc capacities smaller than or equal to the average arc capacity	$\frac{\lvert\{(i,j)\in A: u_{ij}\le\mu(U)\}\rvert}{\lvert A\rvert}$
PercHiCap	Percentage of arc capacities greater than the average arc capacity	$\frac{\lvert\{(i,j)\in A: u_{ij}>\mu(U)\}\rvert}{\lvert A\rvert}$
PercLoArcConf	Percentage of arcs with less than the average number of conflicts per arc	$\frac{\lvert\{(i,j)\in A: \lvert\delta(i,j)\rvert\le\mu(C)\}\rvert}{\lvert A\rvert}$
PercHiArcConf	Percentage of arc conflicts with at least than the average number of conflicts per arc	$\frac{\lvert\{(i,j)\in A: \lvert\delta(i,j)\rvert>\mu(C)\}\rvert}{\lvert A\rvert}$

sequence containing the number of conflicts involving each arc of G.

Results. The feature selection process produced a set of 10 features, corresponding to the most correlated ones with the algorithms' performance. The derived optimal linear transformation of these features into the 2D instance space is given by the following projection equation computed by MATILDA.

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 0.1806 & -0.0043 \\ 0.2812 & 0.3205 \\ 0.2327 & 0.0229 \\ 0.1081 & -0.3217 \\ -0.2415 & -0.06 \\ 0.1021 & -0.1807 \\ 0.3589 & 0.2168 \\ 0.2206 & -0.1617 \\ -0.1821 & 0.1858 \\ -0.3904 & 0.3613 \end{bmatrix}^T \begin{bmatrix} \text{Nodes} \\ \text{AvNdDg} \\ \text{MaxFlow} \\ \text{ConfDensity} \\ \text{cvNdDg} \\ \text{PercLoCap} \\ \text{PercHiCap} \\ \text{ScCapDens} \\ \text{AvgArcConf} \\ \text{cvArcConf} \end{bmatrix} \tag{6}$$

The performance of the trained Support Vector Machines (SVM) models in the resulting instance space is summarized in Table 6: the first column reports the tested algorithms; the second column indicates the probability of each algorithm being labeled as "good" for a specific instance; while the last three columns represent quality measures (i.e., accuracy, precision and recall) of the SVM models trained for each algorithm.

The BD approach is predicted to have a "good" rate of 0.738, with an accuracy of 88.8%, a precision of 92.4%, and a recall of 92.4%. This means the prediction is precise, with a slightly lower recall than the others. CG algorithm instead, has a predicted "good" rate of 0.694, with an accuracy of 69.4%, a precision of 69.4%, and a recall of 100%. While having moderate accuracy and precision values, the high recall value testifies that the SVM model successfully recognizes all the instances for which CG behaves well. The KS approach is predicted to have a "good" rate of 0.731, with an accuracy of 75.0%, a precision of 76.2%, and a recall of 95.7%. Finally, with an accuracy of 96.3%, a precision of 96.3%, and a recall of 100%, the KO algorithm is reliably predicted to be the best performing one, having a "good" rate of 0.963.

Table 6
Statistical results of SVM models.

Algorithm	Pr (Good)	Accuracy	Precision	Recall
BDw	0.738	88.8%	92.4%	92.4%
CG	0.694	69.4%	69.4%	100%
KS	0.731	75.0%	76.2%	95.7%
KO	0.963	96.3%	96.3%	100%

Fig. 12 shows the distribution of the following four selected features, allowing us to observe interesting performance trends of the tested algorithms: Nodes, AvNdDg, PercHiCap, and AvgArcConf. Each point represents an instance, while its color reflects the value of the considered feature for the related instances, normalized to the feature's value range. In particular, minimal values of each feature are shown as blue, while maximal values are shown as yellow. As an example, in Fig. 12(b), the instances with the largest AvNdDg values are located on the first and fourth quadrants (corresponding to $z_1 \geq 0$), while the ones with the smallest values on the remaining quadrants (corresponding to $z_1 \leq 0$). Fig. 13 shows the performance ("good" or "bad") of each analyzed algorithm on each point of the instance space. Considering both Figs. 12 and 13, the BD approach generally exhibits bad performance on instances with numerous nodes, as well as in the presence of many arcs having a capacity greater than the average one, and for instances with high average number of conflicts per arc, corresponding to the points in the upper area of the 2D instance space. Although identifying a bad-performance area for both KS and CG approaches is more challenging, the most difficult instances seem to be mainly located in the central region of the instance space, corresponding to a high average number of conflicts per arc. Finally, the KO approach generally performs well, exhibiting poor performance in the fewest possible instances.

5. Conclusions

In this paper, we introduced three heuristic approaches to solve the Maximum Flow Problem with Conflicts. The first one is a greedy algorithm enhanced according to the Carousel Greedy strategy and the second one is a Kernel Search algorithm. The third algorithm, named

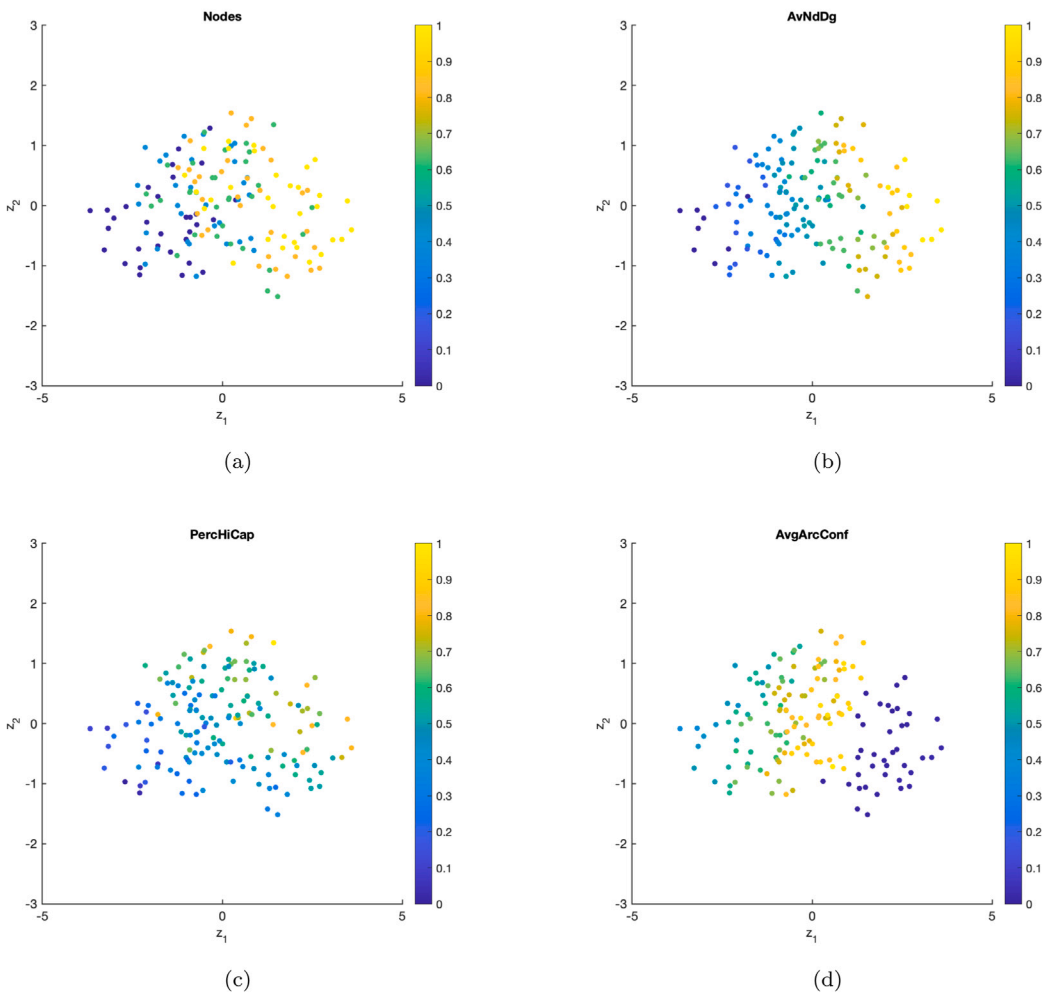


Fig. 12. Distribution of the Nodes (a), AvNdDg (b), PercHiCap (c), and AvgArcConf (d) features' values in the 2D instance space resulting from (6). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Kernousel, is obtained by merging the Carousel Greedy and the Kernel Search approaches. The idea behind Kernousel is to combine the optimality guarantee provided by the Kernel Search for restricted problems with the knowledge gained by the Carousel Greedy algorithm during its exploration of the solution space to obtain a sorting and partitioning of the variables, into the kernel set and the buckets, that makes the Kernel Search more effective. Moreover, the use of Carousel Greedy, as an alternative approach to the classical linear relaxation solution of the problem, makes Kernousel suitable for optimization problems where the information retrieved from the solution of the linear relaxation does not produce good predictors on the most promising variables.

The proposed methods have been tested on benchmark instances and compared with the best-known solutions from the literature. The computational results show that Kernousel is much more effective than the singular approaches applied separately and this occurs also by providing the same time limit to the three algorithms. Finally, the comparison with the best solutions available in the literature shows that the average gap of Kernousel is lower than 1% and there are several instances where it returns a new best-known solution value.

CRediT authorship contribution statement

F. Carrabs: Writing – review & editing, Writing – original draft, Visualization, Supervision, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **R. Cerulli:** Writing – review & editing, Writing – original draft, Supervision, Project administration, Methodology, Investigation, Formal analysis, Conceptualization. **R. Mansini:** Writing – review & editing, Writing – original draft, Supervision, Project administration, Methodology, Investigation, Formal analysis, Conceptualization. **D. Serra:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **C. Sorgente:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization.

Funding

Not applicable.

Ethics approval

This article does not contain any studies with human participants or animals performed by any of the authors.

Declaration of competing interest

Authors declare no conflicts of interest or competing interests.

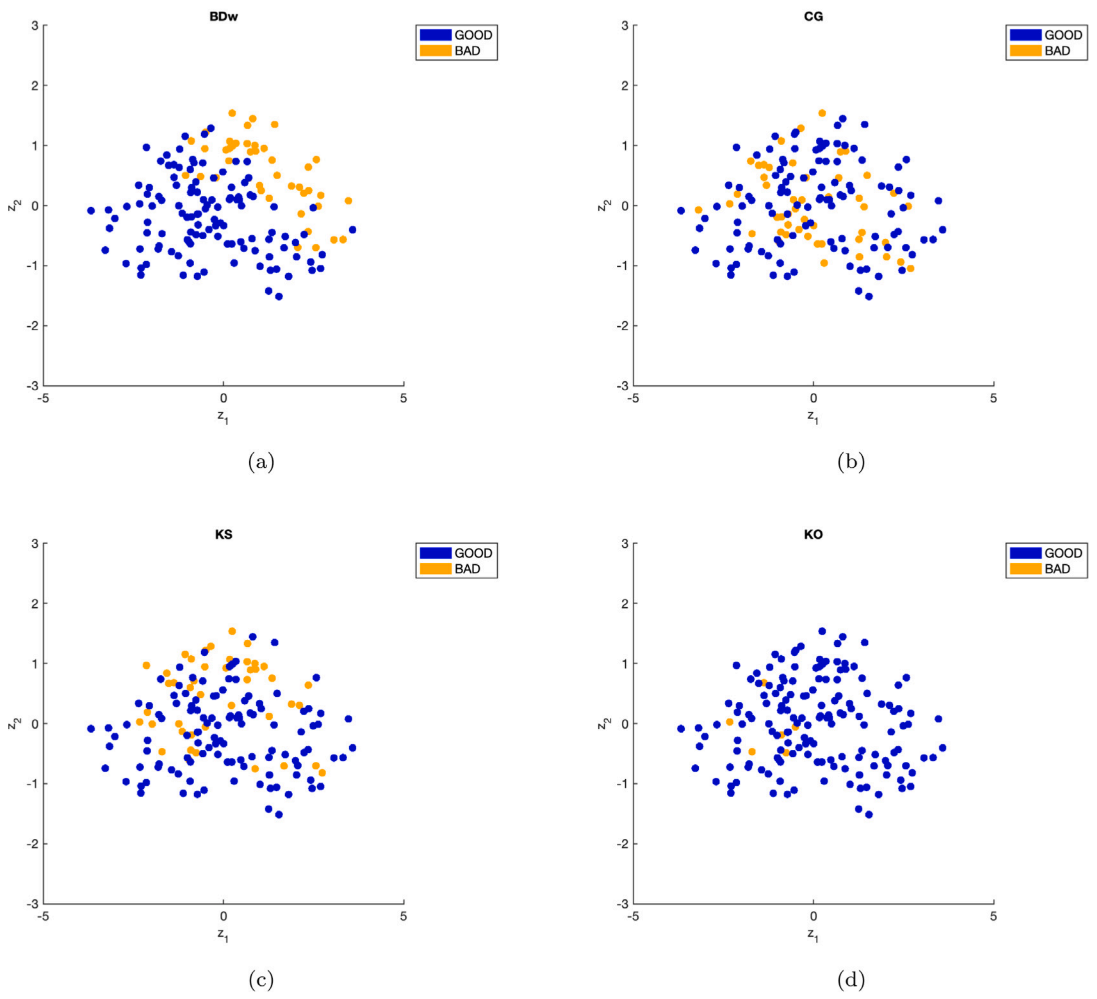


Fig. 13. Binary performance of the BD (a), CG (b), KS (c) and KO (d) approaches on each point of the 2D instance space resulting from (6).

Table A.7
Mathematical symbols and notations.

Notation	Definition
$G = (V, A)$	Capacitated and directed flow network
u_{ij}	Maximum amount of flow that can be sent from node i to node j
$\delta(i, j)$	Set of arcs in conflict with $(i, j) \in A$
$D_{\max}$	Maximum number of conflicts involving an arc of G
$C_{\max}$	Maximum capacity associated with an arc of G
$\mathcal{F}$	Non negative variable representing the amount of flow sent through the network
f_{ij}	Continuous variable indicating the flow carried out be the arc $(i, j) \in A$
x_{ij}	Integer variable indicating whether the arc $(i, j) \in A$ carries some flow
$\hat{f}$	Given flow, i.e., realization of the f variables
$G_{\hat{f}}$	Residual graph associated with flow $\hat{f}$
$r_{ij}^{(\hat{f})}$	Residual capacity associated with arc (i, j) in $G_{\hat{f}}$
S	Solution composed of a sequence of augmenting paths
Δ	Vector of augmenting capacities associated with the paths in S
$z(S)$	Value of solution S
$\mathcal{P}$	Collection of all the paths computed by CG
Λ	Kernel set
λ	Exact (for KS) and minimum (for KO) kernel size
B	Collection of buckets
n_B	Number of buckets
γ	Bucket size
$L_{>}$	Sorted list of pairs of f_{ij} and x_{ij} variables s.t. $f_{ij} > 0$
$L_{=}$	Sorted list of pairs of f_{ij} and x_{ij} variables s.t. $f_{ij} = 0$
L	Sorted list of pairs of variables obtained by concatenating $L_{>}$ and $L_{=}$
ρ_{ij}	Number of conflicts among (i, j) and the arcs associated with variables in $L_{>}$
τ_{ij}	Number of times the arc (i, j) appears in a path generated by CG
$\mathcal{F}(f^*, x^*)$	Value of the best incumbent integer solution

Appendix. Mathematical notation

Table A.7 reports a list of the notations used throughout the paper.

References

Ahuja, R., Magnanti, T., & Orlin, J. (1993). *Network Flows, theory, algorithms, and applications*. Prentice-Hall.

Alipour, H., Muñoz, M. A., & Smith-Miles, K. (2023). Enhanced instance space analysis for the maximum flow problem. *European Journal of Operational Research*, *304*(2), 411–428.

Angelelli, E., Mansini, R., & Speranza, M. G. (2010). Kernel search: A general heuristic for the multi-dimensional knapsack problem. *Computers & Operations Research*, *37*(11), 2017–2026.

Angelelli, E., Mansini, R., & Speranza, M. G. (2012). Kernel search: A new heuristic framework for portfolio selection. *Computational Optimization and Applications*, *51*(1), 345–361.

Capobianco, G., D'Ambrosio, C., Pavone, L., Raiconi, A., Vitale, G., & Sebastiano, F. (2022). A hybrid metaheuristic for the knapsack problem with forfeits. *Soft Computing*, *26*(2), 749–762.

Capua, R., Frota, Y., Ochi, L. S., & Vidal, T. (2018). A study on exponential-size neighborhoods for the bin packing problem with conflicts. *Journal of Heuristics*, *24*(4), 667–695.

Carrabs, F., Cerrone, C., Cerulli, R., & Golden, B. (2020). An adaptive heuristic approach to compute upper and lower bounds for the close-enough traveling salesman problem. *INFORMS Journal on Computing*, *32*(4), 1030–1048.

Carrabs, F., Cerrone, C., D'Ambrosio, C., & Raiconi, A. (2017). Column generation embedding carousel greedy for the maximum network lifetime problem with interference constraints. In *Optimization and decision science: methodologies and applications* (pp. 151–159). Cham: Springer International Publishing.

Carrabs, F., Cerrone, C., & Pentangelo, R. (2019). A multiethnic genetic approach for the minimum conflict weighted spanning tree problem. *Networks*, *74*(2), 134–147.

Carrabs, F., Cerulli, R., Mansini, R., Moreschini, L., & Serra, D. (2024). Solving the set covering problem with conflicts on sets: A new parallel GRASP. *Computers & Operations Research*, *166*, Article 106620.

Carrabs, F., Cerulli, R., Pentangelo, R., & Raiconi, A. (2021). Minimum spanning tree with conflicting edge pairs: a branch-and-cut approach. *Annals of Operations Research*, *298*(1–2), 65–78.

Carrabs, F., & Gaudioso, M. (2021). A Lagrangian approach for the minimum spanning tree problem with conflicting edge pairs. *Networks*, *78*(1), 32–45.

Carvalho, D. M., & Nascimento, M. C. V. (2018). A kernel search to the multi-plant capacitated lot sizing problem with setup carry-over. *Computers & Operations Research*, *100*, 43–53.

Cerrone, C., Cerulli, R., & Golden, B. (2017). Carousel greedy: A generalized greedy algorithm with applications in optimization. *Computers & Operations Research*, *85*, 97–112.

Cerrone, C., D'Ambrosio, C., & Raiconi, A. (2019). Heuristics for the strong generalized minimum label spanning tree problem. *Networks*, *74*(2), 148–160.

Cerrone, C., Gentili, M., D'Ambrosio, C., & Cerulli, R. (2018). An efficient and simple approach to solve a distribution problem. In *New trends in emerging complex real life problems: ODS, taormina, Italy, September 10–13, 2018* (pp. 151–159). Cham: Springer International Publishing.

Cerulli, R., D'Ambrosio, C., Iossa, A., & Palmieri, F. (2022). Maximum network lifetime problem with time slots and coverage constraints: heuristic approaches. *Journal of Supercomputing*, *78*(1), 1330–1355.

Cerulli, R., D'Ambrosio, C., Raiconi, A., & Vitale, G. (2020). The knapsack problem with forfeits. *Lecture Notes in Computer Science*, *12176*, 263–272.

Cerulli, R., Guerriero, F., Scalzo, E., & Sorgente, C. (2023). Shortest paths with exclusive-disjunction arc pairs conflicts. *Computers & Operations Research*, *152*, Article 106158.

Colombi, M., Corberán, Á., Mansini, R., Plana, I., & Sanchis, J. M. (2017). The directed profitable rural postman problem with incompatibility constraints. *European Journal of Operational Research*, *261*(2), 549–562.

Coniglio, S., Furini, F., & San Segundo, P. (2021). A new combinatorial branch-and-bound algorithm for the knapsack problem with conflicts. *European Journal of Operational Research*, *289*(2), 435–455.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to algorithms, 4th edition*. The MIT Press.

Darmann, A., Pferschy, U., Schauer, J., & Woeginger, G. J. (2011). Paths, trees and matchings under disjunctive constraints. *Discrete Applied Mathematics*, *159*(16), 1726–1735.

De Coster, A., Musliu, N., Schaerf, A., Schoisswohl, J., & Smith-Miles, K. (2022). Algorithm selection and instance space analysis for curriculum-based course timetabling. *Journal of Scheduling*, *25*(1), 35–58.

Filippi, C., Guastaroba, G., Huerta-Muñoz, D. L., & Speranza, M. G. (2021). A kernel search heuristic for a fair facility location problem. *Computers & Operations Research*, *132*, Article 105292.

Gendreau, M., Manerba, D., & Mansini, R. (2016). The multi-vehicle traveling purchaser problem with pairwise incompatibility constraints and unitary demands: A branch-and-price approach. *European Journal of Operational Research*, *248*(1), 59–71.

Gobbi, A., Manerba, D., Mansini, R., & Zanotti, R. (2023). Hybridizing adaptive large neighborhood search with kernel search: a new solution approach for the nurse routing problem with incompatible services and minimum demand. *International Transactions in Operational Research*, *30*(1), 8–38.

Guastaroba, G., Savelsbergh, M., & Speranza, M. G. (2017). Adaptive kernel search: A heuristic for solving mixed integer linear programs. *European Journal of Operational Research*, *263*(3), 789–804.

Guastaroba, G., & Speranza, M. G. (2012). Kernel search: An application to the index tracking problem. *European Journal of Operational Research*, *217*(1), 54–68.

Guastaroba, G., & Speranza, M. G. (2014). A heuristic for BILP problems: The single source capacitated facility location problem. *European Journal of Operational Research*, *238*(2), 438–450.

Hanafi, S., Mansini, R., & Zanotti, R. (2020). The multi-visit team orienteering problem with precedence constraints. *European Journal of Operational Research*, *282*(2), 515–529.

Kirschstein, T., & Meisel, F. (2019). A multi-period multi-commodity lot-sizing problem with supplier selection, storage selection and discounts for the process industry. *European Journal of Operational Research*, *279*(2), 393–406.

Lamanna, L., Mansini, R., & Zanotti, R. (2022). A two-phase kernel search variant for the multidimensional multiple-choice knapsack problem. *European Journal of Operational Research*, *297*(1), 53–65.

Li, J., Lan, Y., Chen, F., Han, X., & Blazewicz, J. (2021). A fast algorithm for knapsack problem with conflict graph. *Asia-Pacific Journal of Operational Research*, *38*(6).

Liu, C., Smith-Miles, K., Wauters, T., & Costa, A. M. (2024). Instance space analysis for 2D bin packing mathematical models. *European Journal of Operational Research*, *315*(2), 484–498.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, *3*, 43–58.

Mansini, R., Zanella, M., & Zanotti, R. (2023). Optimizing a complex multi-objective personnel scheduling problem jointly complying with requests from customers and staff. *Omega*, *114*, Article 102722.

Mansini, R., & Zanotti, R. (2022). Two-phase kernel search: An application to facility location problems with incompatibilities. In *ICORES* (pp. 105–111).

Neelofar, N., Smith-Miles, K., Munoz, M. A., & Aleti, A. (2022). Instance space analysis of search-based software testing. *IEEE Transactions on Software Engineering*, *49*(4), 2642–2660.

Öncan, T., Zhang, R., & Punnen, A. P. (2013). The minimum cost perfect matching problem with conflict pair constraints. *Computers & Operations Research*, *40*(4), 920–930.

Pferschy, U., & Schauer, J. (2009). The knapsack problem with conflict graphs. *Journal of Graph Algorithms and Applications*, *13*(2), 233–249.

Pferschy, U., & Schauer, J. (2013). The maximum flow problem with disjunctive constraints. *Journal of Combinatorial Optimization*, *26*(1), 109–119.

Sadykov, R., & Vanderbeck, F. (2013). Bin packing with conflicts: A generic branch-and-price algorithm. *INFORMS Journal on Computing*, *25*(2), 244–255.

Saffari, S., & Fathi, Y. (2022). Set covering problem with conflict constraints. *Computers & Operations Research*, *143*.

Scherer, M. E., Hill, R. R., Lunday, B. J., Cox, B. A., & White, E. D. (2024). Applying instance space analysis for metaheuristic selection to the 0–1 multidemand multidimensional knapsack problem. *Computers & Operations Research*, *170*, Article 106747.

Smith-Miles, K., & Muñoz, M. A. (2023). Instance space analysis for algorithm testing: Methodology and software tools. *ACM Computing Surveys*, *55*(12), 1–31.

Smith-Miles, K., Muñoz, M. A., & Neelofar, N. (2020). MATILDA: Melbourne algorithm test instance library with data analytics. Available at https://matilda.unimelb.edu.au/matilda/ (Accessed 15 January 2025).

Şuvak, Z., Altınel, İ. K., & Aras, N. (2020). Exact solution algorithms for the maximum flow problem with additional conflict constraints. *European Journal of Operational Research*, *287*(2), 410–437.

Van Bulck, D., Goossens, D., Clarner, J.-P., Dimitsas, A., Fonseca, G. H., Lamas-Fernandez, C., Lester, M. M., Pedersen, J., Phillips, A. E., & Rosati, R. M. (2024). Which algorithm to select in sports timetabling? *European Journal of Operational Research*, *318*(2), 575–591.