European Journal of Operational Research 304 (2023) 411–428

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

Enhanced instance space analysis for the maximum flow problem

Hossein Alipour*, Mario Andrés Muñoz, Kate Smith-Miles

School of Mathematics and Statistics, The University of Melbourne, Parkville, VIC 3010, Australia

ARTICLE INFO

Article history:
Received 14 June 2021
Accepted 9 April 2022
Available online 14 April 2022

Keywords:
Validation of OR computations
Maximum flow
Instance space analysis
Instance selection
Feature selection

ABSTRACT

The Maximum Flow Problem (MFP) is a fundamental network flow theory problem, for which many algorithms, supported by strong theoretical worst-case analyses, have been proposed. However, their practical efficiency depends on the network structure, making it unclear which algorithm is best for a particular instance or a class of MFP. Instance Space Analysis (ISA) is a methodology that provides insights into such per-instance analysis. In this paper, the instance space of MFP is constructed and analysed for the first time. Novel features from the networks are extracted, capturing the performance of MFP algorithms. Additionally, this paper expands the ISA methodology by addressing the issue of how benchmark instances should be selected to reduce bias in the analysis. Using the enhanced ISA methodology with MFP as the case study, this paper demonstrates that the most important features can be detected, and machine learning methods can identify their impact on algorithm performance, whilst reducing the bias caused by over-representation within the selected sample of test instances. The enhanced methodology enables new insights into the performance of state-of-the-art general purpose MFP algorithms, as well as recommendations for the construction of comprehensive and unbiased benchmark test suites for MFP algorithm testing.

1. Introduction

The Maximum Flow Problem (MFP) involves finding the maximum volume of flow of a commodity that could be sent from a source node to a sink node through a network (Ford & Fulkerson, 1956; Harris & Ross, 1955). Being a fundamental problem in network flow optimisation, MFP has many explicit and implicit applications in the modelling of real-world problems (Ahuja, Magnanti, & Orlin, 1993). Hence, it has been heavily researched in the last 60 years, with many algorithms proposed for its solution (Ahuja et al., 1993; Boykov & Kolmogorov, 2004; Goldberg, 2009; Goldberg et al., 2015; Hochbaum, 2001; Orlin, 2013). Performance of these algorithms is theoretically evaluated through worst-case analyses and strong mathematical or algorithmic proofs, which provide unbiased and objective conclusions. In contrast, experimental evaluations are less conclusive, but potentially more relevant for practice than theoretical ones, if they explain the expected performance on particular instance types. In practice, MFP algorithms are tested *competitively*, i.e. running algorithms on a limited set of instances and comparing their average performances (Hooker, 1995). An algorithm is "effective" and "publishable", if it out-performs the others on all or most of the selected instances. Unfortunately, these comparisons often lack of insight (Hooker, 1995). Not only can the possibly arbitrary choice of test instances weakens such comparisons, but also they offer no insight about the expected performance of an algorithm on unseen instances or why algorithms behave differently on different instances. To form a more robust conclusion about the practical performance of an algorithm, *scientific testing* is required instead (Hooker, 1995), i.e., to test hypotheses about how an instance's feature impacts an algorithm's performance.

Instance Space Analysis (ISA) is a methodology developed by Smith-Miles et al. (2007; 2008a; 2008b; 2014), whose primary aim is to overcome the above-mentioned difficulties. ISA offers a scientific testing framework of algorithm performance, based on Rice (1976) *Algorithm Selection Problem*, through a mapping between test instance features and the performance metrics of algorithms. The core output from ISA is a visualisation of the space containing all possible test instances of a problem as a 2*D* plane with a mathematically derived boundary. Within this *instance space*, empirical evidence of algorithm performance can be contrasted to the varying characteristics of a diverse set of test instances. To produce this visualisation, each instance is represented as a high-dimensional vector of measurable features, which capture those characteristics known to affect algorithm performance (Smith-Miles & Lopes, 2012). Then, the 2*D* space is constructed by projecting the instances into a plane, and the boundaries of the instance space are generated using the upper and lower bounds of the features.

* Corresponding author.
E-mail addresses: h.alipour@unimelb.edu.au (H. Alipour), munoz.m@unimelb.edu.au (M.A. Muñoz), smith-miles@unimelb.edu.au (K. Smith-Miles).

https://doi.org/10.1016/j.ejor.2022.04.012
0377-2217/

The projection is constructed by enforcing linear trends on both the instance features and algorithm performances, facilitating the emergence of regions of algorithmic strength, called "*footprints*", and weakness. Therefore, ISA facilitates an objective assessment of the practical capabilities of an algorithm, and enable fair prediction of its likely performance for unseen instances. Assessing the diversity of the existing benchmarks as evidenced by the extent to which they fully occupy the instance space, and using the instance space to guide the generation of new instances with specific characteristics for enriching computational experiments (Smith-Miles & Bowly, 2015) are other benefits of the ISA methodology.

ISA is an iterative process due to its sensitivity to the initial set of benchmark instances, selected algorithms, and choice of features. Eventually, with new instances created to fill sparse areas in the instance space, and new features introduced to explain the algorithms' performances, the process is expected to converge (Smith-Miles, Christiansen, & Muñoz, 2020a). Nevertheless, if inappropriate instances are selected, the results might be misleading. In fact, non-uniform density of instances across the space (under- or over-representation of instances) can potentially create bias in several key steps of the ISA methodology: feature selection, dimensionality reduction, and the performance of machine learning classifiers. Down-sampling (thinning) could be used to avoid over-representation bias from instances with similar features. However, to provide sufficient statistical evidence of algorithm performance, there must be enough benchmarks, i.e., sufficient density, creating a conflict with a down-sampling strategy. Therefore, benchmark instance selection must be done systematically to create a suitable trade-off between bias reduction via down-sampling and the need for sufficient density to provide statistical evidence. We introduce a new instance selection procedure into the ISA methodology to this end. Although we generate specific instances to decrease under-representation bias, our main focus is to decrease over-representation bias by removing redundant benchmarks.

In this paper, we analyse the instance space of MFP for the first time. To this end, we collect and generate a comprehensive set of MFP benchmark instances —within the constraints of our computational environment — and discuss the kinds of benchmarks that should be generated for a more complete analysis. We adopt existing features from the literature, and propose new ones demonstrating their effect on performance. Moreover, we apply state-of-the-art general purpose MFP algorithms on the benchmarks and create a metadata for ISA, defined in Section 3. These algorithms are *highest-level Push-Relabel* (Hi_PR) (Cherkassky & Goldberg, 1997), *pseudoflow highest-label FIFO* (Pseudo_Hi_FIFO) (Chandran & Hochbaum, 2009), *partial augment-relabel* (PAR) (Goldberg, 2008), and *two-level Push-Relabel* (P2R) (Goldberg, 2009). We propose a method for selecting instances from our set, obtaining a more uniform distribution of benchmarks. As result, we reduce the bias created by non-uniform density, while retaining sufficient instances for a thorough exploration of the space. After selecting the metadata, a new instance space for MFP is constructed through ISA, analysing the performances of algorithms based on the important features of MFP. In summary our contributions are:

1. Introducing new features of MFP to capture the behaviour of algorithms on different instance types;
2. Developing a comprehensive set of MFP benchmarks to represent a diverse instance space;
3. The first instance space analysis for MFP;
4. Introducing a new instance selection method for reducing bias in different stages of ISA and ensuring that the insights afforded by ISA are not limited or biased by the available test instances;
5. Introducing a statistical measure to assess the adequacy of features in capturing the algorithm performance;
6. Detecting the fundamental differences in algorithm mechanisms that explain the different behaviours of algorithms in practice;
7. Discovering why features may cause certain behaviours of algorithms;
8. Giving a direction to improve the MFP algorithms using the insights obtained in this work;
9. Identifying gaps and future research opportunities in the scientific testing of MFP algorithms.

The paper continues as follows: Section 2 describes the details of MFP, its algorithms, and the limitations present in their experimental analyses. Section 3 presents an initial instance space aimed to address these limitations, where our new features are introduced and new instances are generated to decrease under-representation bias. Section 4 discusses the bias caused by non-uniform density of instances, introduces our instance selection method, and presents the final results with the insights afforded by the enhanced ISA. and future opportunities are discussed in Section 5 followed by our conclusions in Section 6.

2. Maximum flow problem

2.1. Definitions and notations

Let $G = (V, E, C)$ be a directed network defined on V as the set of $\boldsymbol{n}$ nodes, E as the set of $\boldsymbol{m}$ capacitated arcs with positive capacities, C as the set of $\boldsymbol{m}$ upper bound capacities each of which corresponds to an arc in E, and two distinguished nodes, *source* and *sink*, denoted by s and t respectively. An arc e from nodes u to v is denoted by (u, v). Each arc $(u, v) \in E$ has an associated upper-bound capacity $c_{uv} \in C$. A *path* is defined as the sequence of distinct nodes $\{u_1, u_2, \ldots, u_q\}$ such that, for all $1 \le p \le q-1$, either $(u_p, u_{p+1}) \in E$ or $(u_{p+1}, u_p) \in E$. Moreover, if $(u_p, u_{p+1}) \in E$ for any pair of consecutive nodes u_p and u_{p+1}, the path is called a *directed path*. A *flow* is a function $f : E \mapsto \mathbb{R}^+$ satisfying:

$$\sum_{\{v:(v,u)\in E\}} f_{vu} - \sum_{\{v:(u,v)\in E\}} f_{uv} = 0, \quad \forall u \in V \setminus \{s, t\}, \tag{1a}$$

$$\sum_{\{u:(u,t)\in E\}} f_{ut} = |f|, \tag{1b}$$

$$0 \le f_{uv} \le c_{uv}, \quad \forall (u, v) \in E, c_{uv} \in C, \tag{1c}$$

where f_{uv} is the amount of flow on the arc (u, v) and $|f|$ is the value of f. The aim of **MFP** is to find a flow that maximises $|f|$. In other words, MFP aims to find the maximum amount of flow that could be sent from s to t through directed paths in G such that arc flows satisfy *flow conservation constraints* (Eq. (1a)) at all intermediate nodes, i.e., $V \setminus \{s, t\}$, and satisfy *arc capacity constraints* in Eq. (1c). A flow f satisfying constraints (1a)-(1c) is called a *feasible flow*. Additional relevant concepts are provided in Appendix A.

2.2. Computational studies of MFP algorithms

MFP was formulated by Harris & Ross (1955). Since then, multiple algorithms have been developed to solve it. Initially, the network simplex method of Dantzig (1951), designed for the transportation problem, was used to solve MFP as a special case. Ford & Fulkerson (1956) proposed the first *augmenting path* method with a worst-case complexity in the order of $\mathcal{O}(\boldsymbol{mn}c)$, where c is the largest arc capacity in C. This algorithm tries to find a path from s to t, known as an augmenting path, in each iteration and sends the maximum possible flow through this path, then updates flows in the network and looks for another path in the next iteration

until there is none available. Karzanov (1974) introduced the concept of *preflow*, whereby a flow is allowed to be changed on a single arc instead of the entire augmenting path. Goldberg & Tarjan (1988) combined preflow with the idea of *relabelling operations* to design the *pushrelabel* (incorrectly known as *preflow-push*) algorithm with order $\mathcal{O}(\boldsymbol{n}^3)$ complexity. Hochbaum (2001) introduced the *pseudoflow* algorithm with order $\mathcal{O}(\boldsymbol{mn}\log \boldsymbol{n})$ complexity. More recent implementations of all of these algorithms have resulted in improved worst-case complexity (see Ahuja et al., 1993; Hochbaum & Orlin, 2013). More recently, Orlin (2013) proposed another algorithm which, taken together with the algorithm of King, Rao, & Tarjan (1994), results in order $\mathcal{O}(\boldsymbol{mn})$ —the best worst-case complexity obtained so far.

In practice, algorithms usually behave much better than their worst-case, making this theoretical complexity an unsatisfactory criterion to judge expected performances. As a complement, computational investigations have been conducted to observe the power of the proposed MFP algorithms. Currently, Dinic's (1970) algorithm is considered as the most efficient augmenting path algorithm (Ahuja, Kodialam, Mishra, & Orlin, 1997; Cheung, 1980; Imai, 1983); Hi_PR (Cherkassky & Goldberg, 1997) as one of the most efficient Push-Relabel algorithms (Ahuja et al., 1997; Cherkassky & Goldberg, 1997); Pseudo_Hi_FIFO the most efficient pseudoflow algorithm (Chandran & Hochbaum, 2009); with Push-Relabel algorithms being substantially faster than augmenting path ones (Ahuja et al., 1997; Cherkassky & Goldberg, 1997; Derigs & Meier, 1989), and Pseudo_Hi_FIFO being faster than Hi_PR (Chandran & Hochbaum, 2009). Recently, Goldberg (2008, 2009) introduced two improvements of Hi_PR, i.e. partial augment-relabel (PAR) algorithm and two-level Push-Relabel (P2R) algorithm. Goldberg (2009) concluded that PAR and P2R behave similarly, and both outperform Hi_PR on all tested instances, and the pseudoflow algorithms in most tested instances. Moreover, Dinic's, Hi_PR, PAR, P2R, and Pseudo_Hi_FIFO implementations have also been tested on a real-world family of computer vision instances (Boykov & Kolmogorov, 2004; Fishbain, S., & Mueller, 2016; Goldberg, 2009; Goldberg et al., 2015), where it is shown that PAR, P2R, and Pseudo_Hi_FIFO are competitive against domain-specific algorithms.

2.3. Limitations of MFP algorithm testing

The experimental studies mentioned above have at least one of the following limitations. The first one, unfortunately not confined to MFP studies, is the lack of a systematic instance selection approach to detect bias in the datasets used for experimental analyses. As result, each study uses different benchmark families, often not covering all the ones available in the literature, making a comparison between algorithms unfair. Moreover, the number of instances from each family is often small, resulting on *representation bias* (Suresh & V. Guttag, 2020) due to under-sampling.

The second limitation is that the features that impact the algorithms' performances have not been thoroughly explored. Features such as network size, average node degree, density, connectivity, or shape of the network are highly correlated with performance. However, they remain largely unused for drawing insights into the expected performance of an algorithm on different instance types. Summarising algorithm performance on average across all test instances studied hides valuable insights. In addition, the number of features used in competitive testing are limited to a few in each experiment, making them less informative. Using *analysis of variance*, Sedeño-Noda, González-Sierra, & González-Martín (2000) explored the impact of the number of nodes, number of arcs, the maximum arc capacity, and the instance generator, among other factors, had on CPU time. Although they concluded that some factors impact performance, such as the generator type, they provided no insights into why. Despite this, Sedeño-Noda et al. (2000) acknowledged that more factors must be studied to fully capture the impact that generators have on practical efficiency.

The final limitation is that comparing average performance on the studied test instances cannot be used to predict the most appropriate algorithm for an unseen instance. Without understanding the impact of different features on the algorithms' performances over a comprehensive and unbiased set of benchmarks, predicting algorithms' performances for unseen problems is likely to be unreliable.

These limitations demonstrate the need for a scientific testing methodology applied to MFP algorithms. *Instance Space Analysis* (ISA) is a methodology aimed to bridge these limitations. In the following section, we show how ISA can be applied to provide a more rigorous testing of algorithms for MFP, before enhancing the methodology to address the issue of test instance selection.

3. Instance space analysis for the maximum flow problem

3.1. Background and framework

Instance Space Analysis (ISA) draws its foundations from Wolpert and Macready's (1997) *No-Free Lunch theorem*, which state that for a given problem, an algorithm is unlikely to outperform other algorithms on all possible instances; and Rice's (1976) *Algorithm Selection Problem*, which frames the solution as a prediction of the algorithm performance using measurable features of problem instances. Fig. 1 illustrates ISA, with Rice's framework (inside the dashed-line box) having four components:

- The *problem space* $\mathcal{P}$ composed of all possible instances for a given problem;
- The *feature space* $\mathcal{F}$ comprising measurable characteristics of the instances in $\mathcal{P}$;
- The *algorithm space* $\mathcal{A}$ consisting of a set of algorithms to solve the problem;
- The *performance space* $\mathcal{Y}$ containing the mapping of each algorithm in $\mathcal{A}$ to a set of performance metrics.

The collection $\{\mathcal{P}, \mathcal{F}, \mathcal{A}, \mathcal{Y}\}$ is referred to as a problem's *metadata*. The algorithm selection problem is stated as: for a given problem instance $x \in \mathcal{P}$ with a feature vector $\mathbf{f}_x = \mathbf{f}(x)$, find the selection mapping $\mathcal{S}(\mathbf{f}_x)$ into the algorithm space $\mathcal{A}$ such that the selected algorithm $\alpha \in \mathcal{P}$ maximises the performance mapping $y_{\alpha,x} = y(\alpha, x) \in \mathcal{P}$. Through Rices framework, and using regression or other supervised learning methods, the relationship between the features and the algorithm performance can be identified, and used to predict algorithm performances for unseen instances. The blue block in Fig. 1 represents a new instance selection procedure presented in Section 4 to enhance the ISA methodology. ISA is performed using the tools available at the Melbourne Algorithm Test Instance Library with Data Analytics (MATILDA) (Muñoz & Smith-Miles, 2020; Smith-Miles, Muñoz, & Neelofar, 2020b), which allow us to:

1. create a visualisation of the instance space for MFP:
   (a) showing the location of benchmark test instances across the instance space;
   (b) revealing the strengths and weaknesses of MFP algorithms across the instance space;
   (c) summarising the properties of the instances that an algorithm finds easy or hard;
2. calculate objective metrics of algorithmic power via footprint analysis;
3. identify the locations of new test instances that should be generated or acquired at specific locations in the instance space to fill gaps, especially, in the edges of different footprints to ex-

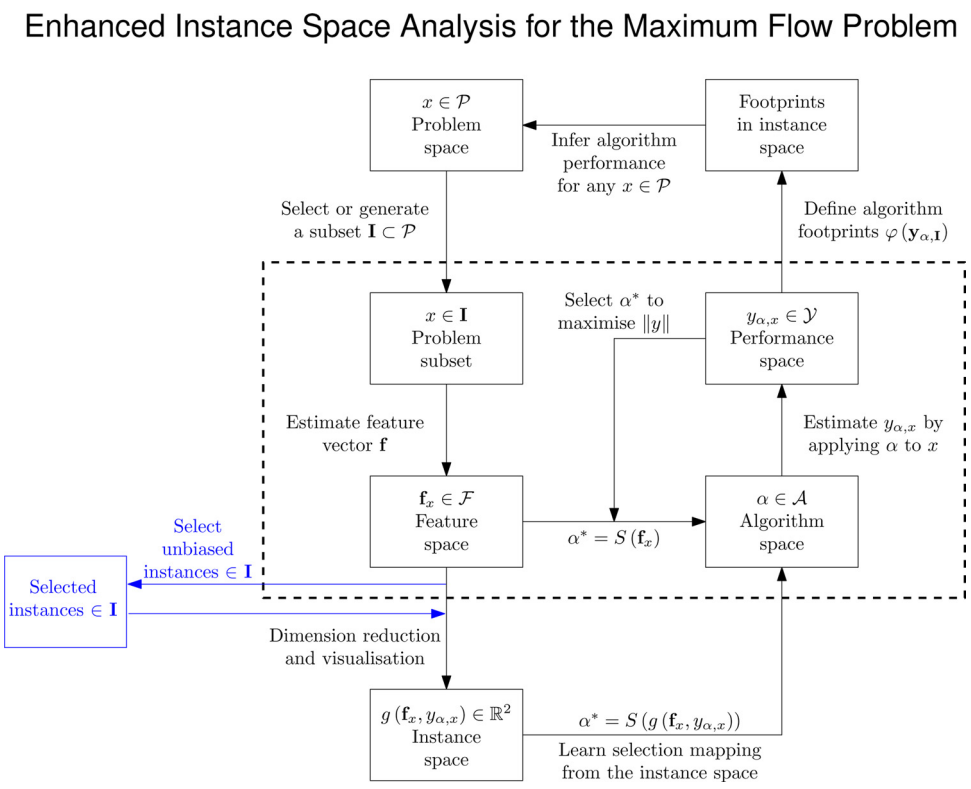


Fig. 1. Enhanced ISA framework: Rice's framework is illustrated within the dashed-line box; ISA framework includes all parts except the blue box; the enhanced ISA presented in this paper augments the ISA framework with the instance selection component in blue. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

plore phase transition boundaries adequately and provide more evidence for machine learning classifiers;
4. recommend the best algorithms for a given instance via automated algorithm selection tools.

MATILDA employs a tailored dimension reduction algorithm (Muñoz, Villanova, Baatar, & Smith-Miles, 2018) that finds an optimal linear transformation from the higher dimensional feature space to the 2*D* instance space, such that maximises the degree of linear trends across the instance space for the distribution of each feature and algorithm performance metric. In this way, the relationships between instance features and algorithm performance can be more easily visualised. Within the instance space, a set of Support vector Machine (SVM) classification models are used, one for each algorithm, based on binary labels of each instance as inducing a good or bad performance by the algorithm, following a user defined performance criteria (see Section 3.2.3). These SVM models are used for performance prediction on unseen instances, and for automated algorithm selection. Finally, the footprints of each algorithm are calculated as the areas of high purity[1] of good performance, and are footprint statistics and locations are used to provide an objective assessment of each algorithm's strengths and weaknesses across the entire instance space. The subroutines employed by MATILDA to create and analyse the instance space are summarised in Appendix B.

In the next sections, we describe the initial metadata, before presenting MATILDA's analysis of the resulting instance space.

3.2. Initial metadata

3.2.1. Benchmark problems

We utilised 16 benchmark generators available in the literature, specified for the general MFP in DIMACS format[2]. These are: AC, AK, Netgen, RMF, Random, TG, and the Washington family of 10 generators (Mesh Graph, Random Level Graph, Random 2-Level Graph, Square Mesh, Basic Line, Exponential Line, Double Exponential, DinicBadCas, GoldBadCase, and Cheryian). This synthetic instances are complemented with a family of real-world instances extracted from computer vision applications[3].

To produce more diverse benchmarks than those commonly studied in the literature, we considered a wider range of values for the controllable parameters in each generator. For example, the number of nodes considered in the literature are in the format $\boldsymbol{n} = 2^{\iota}$ for different values of ι. Here, we used other bases too, i.e., $3, 5, 7, 11, \ldots$. Moreover, the min/max arc capacities are fixed in some generators, while they can impact the performance of MFP algorithms more than other factors in some instances. For example, some huge instances from the AK family can be solved much more quickly than medium sized ones depending on the maximum arc capacities (see Figure E.1 in the Appendix). We manipulated these capacity bounds to produce more diverse networks. Based on the number of controllable parameters and the diversity of the benchmarks, we also generated a different number of benchmarks for each family. Furthermore, for the generators with random parameters, we generated 5 different replications for each set of values of non-random parameters. As a result, we obtained 8508 instances for the initial metadata. The values of the controllable parameters

[1] For an algorithm in an area, *purity* is defined as the number of instances labeled as good for that algorithm over the number of all instances in that area.

[2] Generators and further information are available at http://dimacs.rutgers.edu/pub/netflow/

[3] Available at https://vision.cs.uwaterloo.ca/data/maxflow

for each family fall within the ranges presented in Table E.1 in the Appendix.

3.2.2. Algorithms

In this work, we survey the most successful algorithms for solving the general MFP. Some specific cases, such as unit capacity or bipartite networks and MFPs arising from computer vision problems, can be solved more effectively by specialised algorithms (see Ahuja et al., 1993; Boykov & Kolmogorov, 2004; Goldberg et al., 2015). However, we do not consider them here, since such specialised algorithms either cannot solve the general MFP or are implemented in a language other than C, which is the implementation language of tested algorithms in this work for fairness of CPU comparisons. The most successful general-purpose MFP algorithms in the literature, which belong to the *pseudoflow* and *Push-Relabel* families, are Pseudo_Hi_FIFO (Chandran & Hochbaum, 2009), Hi_PR (Cherkassky & Goldberg, 1997; Goldberg & Tarjan, 1988), PAR (Goldberg, 2008), and P2R (Goldberg, 2009). A short summary for each of these algorithms is provided in Appendix A. We also tested the algorithm of Dinic (1970) but it was uncompetitive with the four other algorithms used here, therefore, we do not include it in this study.

3.2.3. Performance measure

Defining the *goodness* of an algorithm is the key to assessing its performance. In the literature, goodness is often taken as *best performance*: an algorithm is *good* in solving an instance if it outperforms other algorithms on the chosen performance metric. Nevertheless, the definition of goodness is dependent on the performance measure and the underlying problem. For example, in this paper we use CPU time, as it is the most common performance measure for testing and comparing MFP algorithms. However, the best CPU time is unsuitable as a measure because of its noisiness. In addition, small differences in CPU times are irrelevant, as they will soon disappear by advances in computational technology.

An alternative is to consider a range of acceptable performances within a tolerance threshold of the best, which addresses the aforementioned issues about reliability of CPU times. For algorithms $\alpha_1, \alpha_2, \ldots, \alpha_a \in \mathcal{A}$ let $\mathbf{y}_i = \begin{bmatrix} y_{i,1} & \ldots & y_{i,a} \end{bmatrix}^\top$ be the vector of performance values on an instance $x_i \in \mathbf{I}$. The performance of an algorithm in an instance is *good* if it is within $100 \times \epsilon\%$ of the best performing algorithm, where ϵ is a *goodness threshold*. Let $\delta_{i,k} = \mathbf{1}\left(y_{i,k} \leq (1+\epsilon) y_{i,\text{best}}\right)$ be the binary performance of $\alpha_k, k = 1, \ldots, a$ on x_i, where $\mathbf{1}(\cdot)$ denotes the indicator function and $y_{i,\text{best}}$ is the performance of the best algorithm on x_i. The vector of binary performance values on x_i is denoted as $\boldsymbol{\delta}_i = [\delta_{i,1} \ \ldots \ \delta_{i,a}]^\top$. In ISA, we compare algorithms based on such binary labels, capturing if the algorithm's performance was defined as good for any instance. We also regard an instance as *hard* if less than 55% of algorithms achieve a good performance label on it.

3.2.4. Features

Since MFP is concerned with network flows, the capacity of arcs is just as important as the spectral features of the network. In the literature, some features such as *weighted clustering coefficients*, which consider arc capacities, are discussed (Clemente & Grassi, 2018; Opsahl & Panzarasa, 2009); Unfortunately, calculating such features is intractable for large networks, demanding more computing resources than an MFP algorithm. For example, calculating weighted clustering coefficients of a network requires $\Theta(n^4)$, while the practical complexity of Hi_PR is shown to be $\mathcal{O}(\boldsymbol{n}^{1.5})$ (Ahuja et al., 1997). There are many simple spectral network features that can be measured cheaply, but versions that combine the arc capacities with such spectral features are non-existent in the literature. Consequently, our challenge for constructing suitable features for MFPs is to propose tractable features that consider arc capacities as well as network structures. In addition, since multiplying all arc capacities by a fixed scalar does not impact the algorithm's performances, we used the scaled versions of capacities whenever required. *Scaled capacitated density* is inspired by this discussion and defined in Table 1, combines the density with the average capacity of the network to reveal the information about the capability of a network in carrying different amounts of flow; a high/low value of this feature indicates the network can carry a large/small amount of flow.

We also define some features based on our intuition into MFP. For example, compensating the node excesses is the main phase of Pseudo and Push-Relabel algorithms; hence, excesses affect the performance of these algorithms. Because the node excesses are unknown before the algorithm process, we estimate the potential excess for each node and across the network. Firstly, we define the *potential balance* in node u:

$$pb(u) := \sum_{\{v:(v,u)\in E\}} c_{vu} - \sum_{\{v:(u,v)\in E\}} c_{uv}. \tag{2}$$

A positive value of $pb(u)$ is a *potential excess* of node u, $PotExcess(u)$, and a negative value is a *potential deficit* of node u. We denote the *potential excess of a network* by *PotNetExcess*, defined as

$$PotNetExcess := \sum_{u\in V, u\neq s,t} PotExcess(u). \tag{3}$$

Using the notation from Section 2.1, we define 14 features listed in Table 1, of which four are extracted from the literature, and ten are new features. All of these features can be calculated independent of knowing the optimal solutions of the instances. Moreover, this features do not include those that are intractable for the large networks in our metadata, or that require computational effort greater than solving the instance by MFP algorithms. We note that if the aim is to obtain some insights, rather than automated algorithm selection, the last two restrictions could be relaxed. Finally, the mathematical formulation for some features is not presented in Table 1 as they are trivial. However, they can be found in Appendix C.

3.3. Initial results

In this section, we present the initial instance space analysis using MATILDA, specifying all the parameters and computational environments needed for reproducibility. The resulting instance space can be explored online at Alipour, Muñoz, & Smith-Miles (2021).

3.3.1. Experimental setup

We used the University of Melbourne SPARTAN HPC system (Lafayette, Sauter, Vu, & Meade, 2016), with an Intel Xeon Gold 6254 CPU @ 3.10GHz processor, with allocated memory between 1GB to 64GB depending on the size of each instance. The source codes of the benchmark generators, MFP algorithms, and feature extraction (Alipour, 2021) codes are in C, while the MATILDA toolkit (Muñoz & Smith-Miles, 2020) and the instance selection procedure (Alipour, 2021) introduced in this paper are MATLAB scripts. Algorithms tested are Pseudo_Hi_FIFO v3.23, Hi_PR v3.6, PAR v043c, and P2R v045c, all compiled with gcc 4.8.5 using the -o4 optimisation flag.

All algorithms clock very fast CPU times on tiny instances; therefore, we avoided generating instances with $Order < 500$ or $Size < 2000$. We recorded the times to a precision of three decimal places to achieve good discrimination of the algorithms' performances, but also because higher precision made no substantial difference in the results, and less precision made the performances indistinguishable on small instances. We chose as the performance metric the average CPU time from five repetitions, because this was

Table 1
Set of initial 14 features used in the metadata.

Feature name	Description
Features from the literature: (4 features)	
Order	Number of the nodes in the network;
Size	Number of the arcs in the network;
AvNdDg	Average node degree: $\frac{m}{n}$;
Density	$\frac{m}{n(n-1)}$;
New features: (10 features)	
cvNdDg	Coefficient of variation of node degrees;
cvCap	Coefficient of variation of arc capacities;
PercLoCap	Percentage of capacities smaller than average arc capacity;
PercHiCap	Percentage of capacities greater than average arc capacity;
ScRngCap	Scaled range of arc capacities:= the range of arc capacities over the average arc capacity;
ScAvCap	Scaled average arc capacity := $\frac{AvCap(C)}{MedCap}$, where $AvCap(C)$ and $MedCap$ are average and median of arc capacities in C respectively;
ScAvNdCap	Scaled average node capacity:= $\frac{AvNdCap}{MedCap}$, where $AvNdCap$ is the sum of all arc capacities in C divided by $\boldsymbol{n}$;
ScCapDens	Scaled capacitated density := $ScAvCap \times Density$;
ScPotNetExcess	Scaled potential net excess := $\frac{PotNetExcess}{AvCap(C)}$;
AVScPotNetExcess	Average of scaled potential net excess := $\frac{ScPotNetExcess}{n-2}$;

the minimum that provided some convergence to normal distribution with a reasonable computational cost. Moreover, we included in the performance metric the initialisation time, i.e., where flows are allocated to the arcs to get an initial solution, because it can be significant (Verma & Batra, 2012).

We set the main parameters of the MATILDA toolkit as follows. The performance measure is minimising CPU time with relative performance and $\epsilon = 0.05$. We activated the automatic pre-processing and the feature selection. If the Pearson correlation coefficient exceeds 0.3 between a feature and an algorithm performance, we regard it as one of the top most correlated features with the algorithm performance. We found that the implementation of the SVMs using a Gaussian kernel function with LIBSVM's libraries (Chang & Lin, 2011) leads to a good discrimination of algorithms' performances and yields good accuracy of SVM prediction. Other parameters in MATILDA toolkit are described in Appendix B.

3.3.2. Results

Applying ISA on the initial metadata, denoted by $\mathcal{M}_0$, five features were selected. As such, each instance is represented as a 5D feature vector, visualised as a 2D point in the instance space given by the equation:

$$\begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} = \begin{bmatrix} 0.9272 & -0.2228 \\ 0.2074 & 0.5812 \\ 0.3866 & 0.5068 \\ -0.4044 & -0.5475 \\ -0.2502 & 0.1752 \end{bmatrix}^{\top} \begin{bmatrix} Size \\ AVScPotNetExcess \\ cvCap \\ ScAvCap \\ ScCapDens \end{bmatrix} \tag{4}$$

Fig. 2 shows the distribution of the instances of various generators and the predicted footprints of algorithms in the instance space created using $\mathcal{M}_0$ and Eq. (4). Trends of the features in this instance space are also presented in Fig. 3. The location of each instance in the instance space is determined by its feature values, such that instances with similar features are located near each other even if they are from different families. This demonstrates that the benchmark family is less important than the characteristics of individual instances, especially for evaluating the behaviour of algorithms. For example, according to the portfolio footprints, Pseudo_Hi_FIFO is recommended for most but not all the space; however, its strengths (or weaknesses) are unrelated with particular classes of instances, but with the feature combination that instances share. The same observation can be made for the remaining algorithms. Because of the sparsity of this instance space, which indicates inadequate diversity of these initial instances, the relationship between features and performances is examined later. Another issue is the presence of non-uniform density in the instance space, which is discussed in Section 4. For now, we try to improve the diversity of instances.

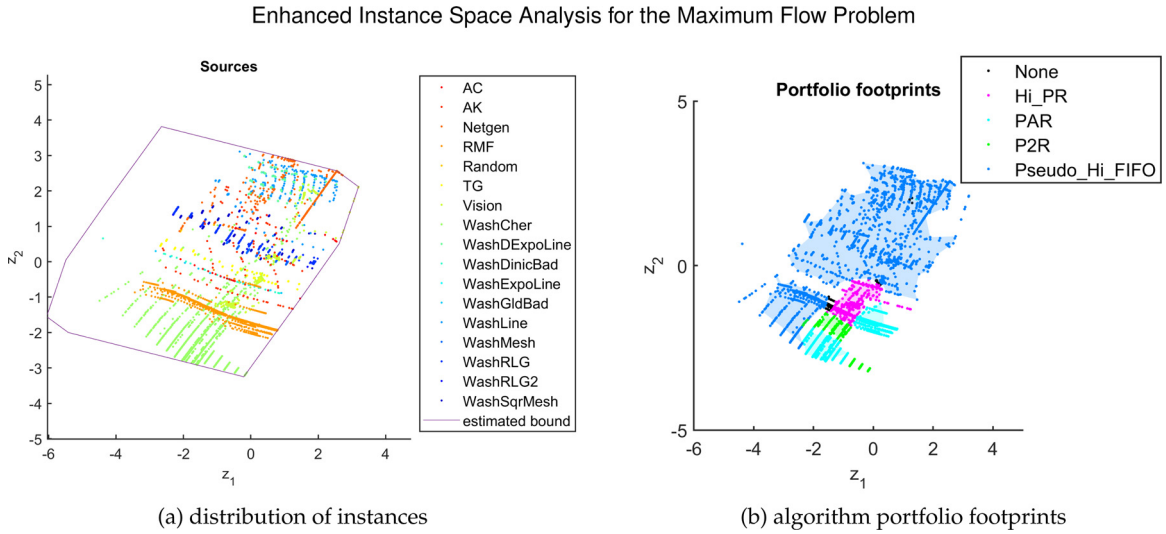
3.4. Generating new instances to fill sparse areas of the instance space

An approach to fill the sparse areas in the instance space is to generate new instances using the available generators by exploring alternative parameters. To this end, we highlight each individual family in the instance space then, based on their nearby location, we target some families that are most promising in filling these holes. Using the distribution pattern of instances from each target family, we detect how to change the values of controllable parameters of that family to generate instances in the targeted areas. Fig. 4 illustrates the distribution of some target families across the instance space (see Figure E.2 in the Appendix for the distribution of other target families). Despite the simplicity of this approach, it might be hard to fill some holes in the projected instance space appropriately because the projection's impact on the density is out of our control. Alternative approaches will be discussed later in Section 5.2.3.

After generating new benchmarks in this way and adding them to $\mathcal{M}_0$, we obtained an augmented metadata set $\mathcal{M}_0'$ with 25,401 instances. Applying ISA on $\mathcal{M}_0'$ with the same parameter setting as before, six features were selected, with the corresponding 6D space reduced to a 2D space using the projection matrix in (5). The updated instance space and the predicted footprints associated with the selected features of $\mathcal{M}_0'$ and Eq. (5) are shown in Fig. 5.

$$\begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} = \begin{bmatrix} -0.1108 & -0.2886 \\ -0.8386 & -0.5103 \\ -0.2889 & 0.3535 \\ -0.2374 & 0.2604 \\ 0.0952 & -0.3641 \\ -0.0029 & 0.3941 \end{bmatrix}^{\top} \begin{bmatrix} Order \\ Size \\ AvNdDg \\ ScPotNetExcess \\ cvNdDg \\ ScCapDens \end{bmatrix}. \tag{5}$$

Fig. 5 illustrates that the newly generated instances dramatically improved the density. However, non-uniform density becomes a major issue due to over-sampled families that create high-density pockets, while other areas remain sparse. Since our assessment of the diversity of the instances might be unreliable in the presence of the bias created by non-uniform density, we must increase the uniformity and reduce the representation bias before adding more benchmarks to $\mathcal{M}_0'$ and filling the remaining sparse areas. In Section 4 we will discuss how the bias created by non-uniform density of instances misleads feature selection, dimensionality reduction, and the performance of machine learning



(a) distribution of instances (b) algorithm portfolio footprints

Fig. 2. Distribution of the instances and predicted algorithm portfolio footprints across the instance space created using $\mathcal{M}_0$ and (4). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

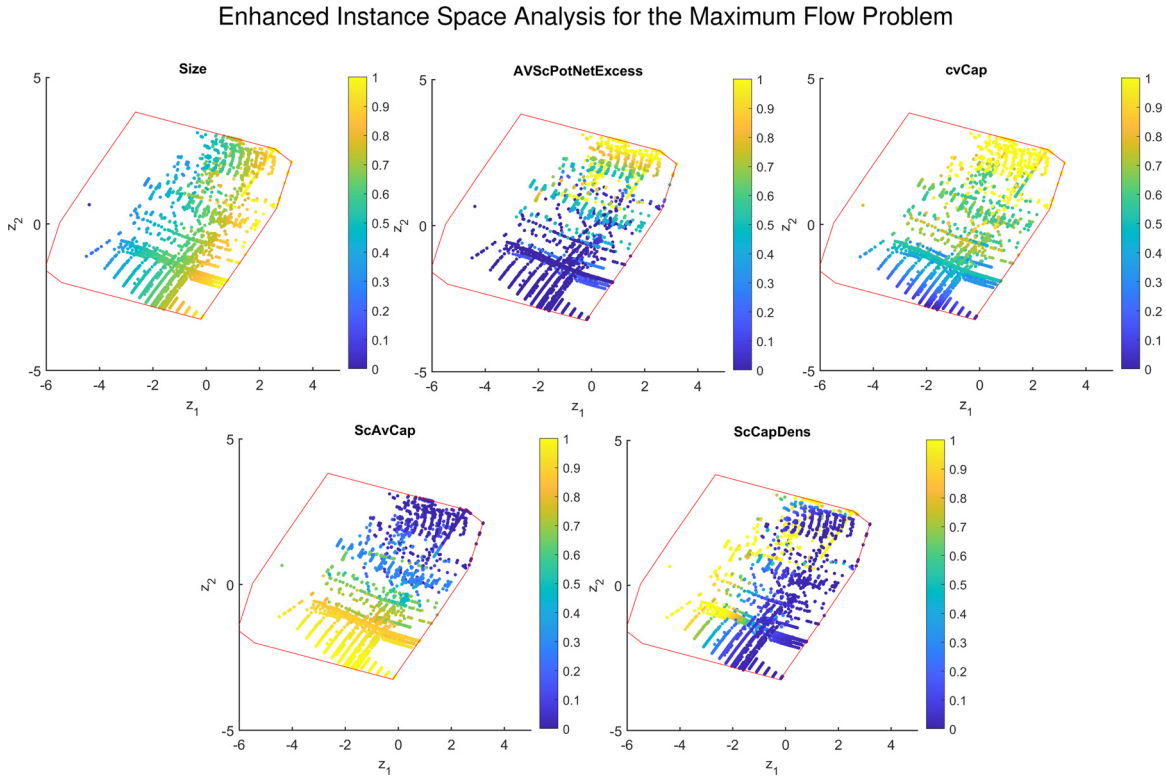
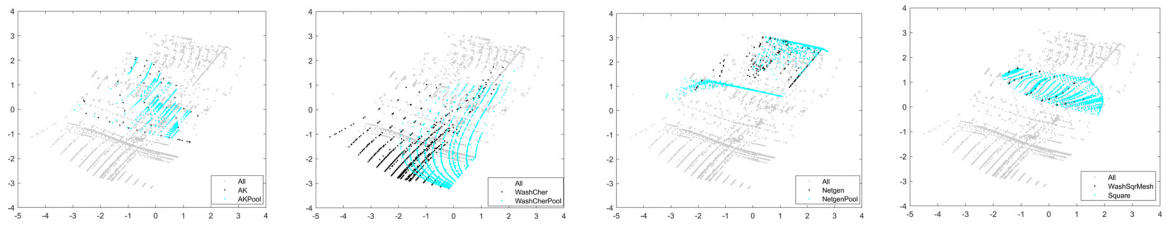
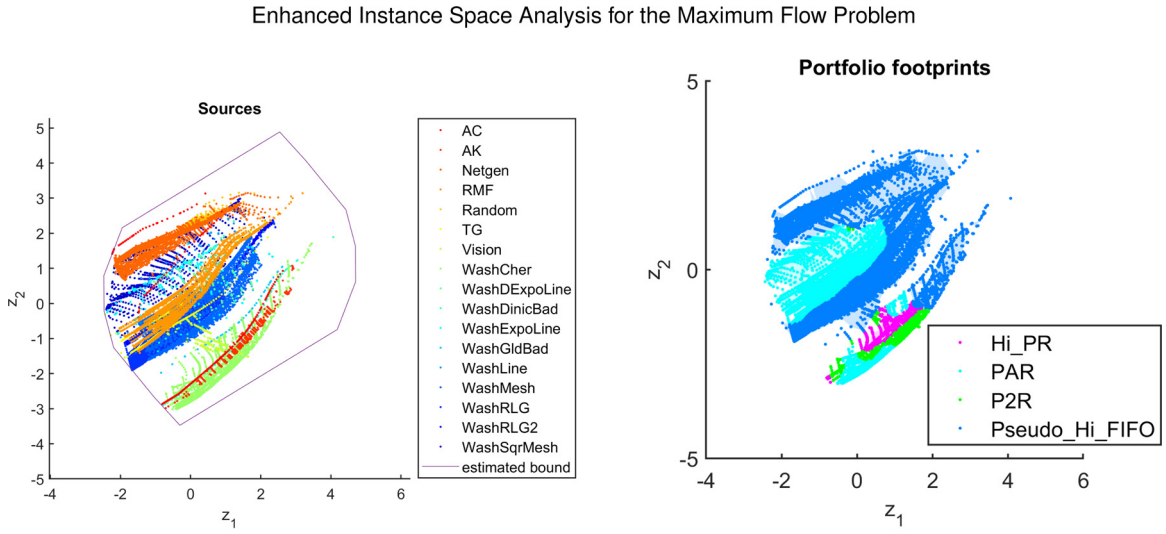


Fig. 3. Distribution of selected features in the initial instance space with a colour scale ranging from scaled minimum (dark blue) to scaled maximum (light yellow) values. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)



(a) AK (b) Cheryian (c) Netgen (d) Square Mesh

Fig. 4. New benchmarks introduced to fill the existing holes; *gray* points are all current instances except those in the target family, *black* points are the current instances from the target family, and *cyan* points are newly generated instances from the target family intended to fill empty and sparse areas. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)



(a) distribution of instances (b) portfolio footprints

Fig. 5. Distribution of the instances and predicted portfolio footprints across the instance space created using $\mathcal{M}_0'$ and Eq. (5). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

classifiers. To handle this issue, we select instances carefully by a novel instance selection procedure introduced in the next section.

4. An instance selection procedure for bias reduction

Non-uniform density appears when many benchmarks have similar features, leading to oversampling in some regions of the instance space, in turn causing representation bias (Suresh & V. Guttag, 2020). Oversampling misleads three key steps in ISA:

- it impacts the estimation of correlation among features and between performance and features, resulting in biased feature selection;
- it affects the estimation of variance of the features, misleading the dimensionality reduction;
- it changes the density and purity across the instance space, biasing the performance of the machine learning classifiers.

Instance selection is a common task within the field of Machine Learning, particularly in the context of fitting classification models, as a means to address oversampling. García, Luengo, & Herrera (2015) provided a comprehensive taxonomy of over 100 approaches for instance selection proposed to date; Ferńandez et al. (2018) investigated over 90 approaches of dealing with imbalanced data sets proposed to date; none of these approaches, nor any other recently reported studies, aims to reduce non-uniform density to reduce over-representation bias to the best of our knowledge. Therefore, we introduce an instance selection procedure aimed to decrease the oversampling bias, which retains only the most informative instances while controlling the information loss and enforcing a more uniform density across the space. While the instance selection procedure has been proposed to mitigate the bias in instance space analysis, the concepts exposed in the methodology are broadly applicable to any machine learning challenge to reduce the representation bias and select an unbiased set of instances. The details of this procedure are described in the following sections, with Section 4.1.1 presenting our method to detect informative instances, and Section 4.1.2 presenting our method to correct the non-uniform density. We also discuss when and how this procedure should be applied in the context of ISA.

4.1. Methodology

4.1.1. Critical and redundant instances

To handle non-uniform density resulted from oversampling, we propose the removal of *redundant* instances. To formalise the definition of redundant, let $\mathbf{f}_i = \begin{bmatrix} \mathrm{f}_{i,1} & \dots & \mathrm{f}_{i,m} \end{bmatrix}^\top$ be the feature vector of $x_i \in \mathbf{I}$. Two instances $\{x_i, x_j\}$ are *similar* if:

$$\left\| \mathbf{f}_i - \mathbf{f}_j \right\|_2 \leq \theta, \tag{6}$$

where θ is a similarity threshold; otherwise they are *dissimilar*. ISA assumes that *for two similar instances, an algorithm is likely to have the same binary performance*. In other words, if (6) is true then:

$$\boldsymbol{\delta}_i = \boldsymbol{\delta}_j \tag{7}$$

must also be true. A pair of instances that fulfil this *similarity assumption* carry the same information; hence, one of them is *redundant* and could cause oversampling bias (Suresh & V. Guttag, 2020). However, we might find a pair of similar instances for which (7) is false, i.e., $\boldsymbol{\delta}_i \neq \boldsymbol{\delta}_j$. Then, one of them *violates the similarity assumption*, implying that the current set of features cannot properly explain algorithm performance and perhaps a new feature is required. Although violations could be addressed by removing one of the instances, this would result in valuable information being lost. We define the set of instances in violation of the similarity assumption (named ViSA) and denote the set by $\mathcal{V}_\theta$. The set of dissimilar instances is denoted by $\mathcal{D}_\theta$. Dissimilar and ViSA instances form the *critical* set of instances, denoted by $\mathcal{C}_\theta := \mathcal{D}_\theta \cup \mathcal{V}_\theta$. Redundant instances are denoted by $\mathcal{R}_\theta := \mathbf{I} \setminus \mathcal{C}_\theta$. The ratio between the cardinalities of $\mathcal{V}_\theta$ and $\mathcal{C}_\theta$ is called *ViSA ratio*, and it is calculated as follows:

$$r_\theta = \frac{|\mathcal{V}_\theta|}{|\mathcal{C}_\theta|} = 1 - \frac{|\mathcal{D}_\theta|}{|\mathcal{C}_\theta|}, \tag{8}$$

where $|.|$ is cardinality. Note that $0 \leq r_\theta < 1$, with values of r_θ close to zero indicating that the current set of features describes algorithm performance well, whereas values close to one indicate a poor set of features. Hence, we propose r_θ as a criterion to assess the adequacy of a feature set.

4.1.2. Instance selection algorithm

Algorithm 1 selects critical instances from a given metadata set based on a given similarity threshold θ. It uses as inputs a list,

Algorithm 1: Instance selection algorithm .

Result: A filtered metadata including $\mathcal{C}_\theta$, $\mathcal{V}_\theta$, and $\mathcal{D}_\theta$ associated with a given similarity threshold θ;
Initialisation: given a list $\mathcal{L}$ of instances and a similarity threshold θ, set $\mathcal{C}_\theta = \emptyset, \mathcal{V}_\theta = \emptyset, \mathcal{D}_\theta = \emptyset$, and $\mathrm{Id}_i = 1, \forall x_i \in \mathcal{L}$;
while $\mathcal{L} \neq \emptyset$ **do**
  select an instance x_i from $\mathcal{L}$ in FIFO order;
  for *any* $x_j \in \mathcal{L}, x_j \neq x_i$ **do**
    if $\left\|\mathbf{f_i} - \mathbf{f_j}\right\|_2 \leq \theta$ **then**
      if $\boldsymbol{\delta}_i = \boldsymbol{\delta}_j$ **then**
        $\mathcal{L} \leftarrow \mathcal{L}/\{x_j\}$;
      else
        $\mathrm{Id}_j \leftarrow 0$;
      end
    end
  end
  if $Id_i = 1$ **then**
    $\mathcal{D}_\theta \leftarrow \mathcal{D}_\theta \cup \{x_i\}$;
  else
    $\mathcal{V}_\theta \leftarrow \mathcal{V}_\theta \cup \{x_i\}$;
  end
  $\mathcal{L} \leftarrow \mathcal{L}/\{x_i\}$;
end
$\mathcal{C}_\theta \leftarrow \mathcal{V}_\theta \cup \mathcal{D}_\theta$;
return $\mathcal{C}_\theta, \mathcal{V}_\theta, \mathcal{D}_\theta$;

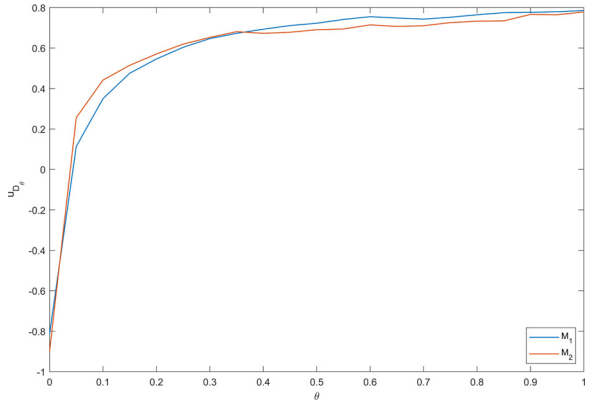


Fig. 6. The uniformity of $\mathcal{D}_\theta$ versus θ in $\mathcal{M}_1$ and $\mathcal{M}_2$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

denoted as $\mathcal{L}$, of all instances in $\mathbf{I}$, vectors of features and binary performance labels, $\mathbf{f}_i$ and $\boldsymbol{\delta}_i$ respectively, for each $x_i \in \mathbf{I}$. As a first step, the algorithm initialises $\mathcal{C}_\theta$, $\mathcal{V}_\theta$ and $\mathcal{D}_\theta$ as empty sets. Then, it creates an identifier $\mathrm{Id} \in \{0, 1\}$ for each instance to separate $\mathcal{V}_\theta$ from $\mathcal{D}_\theta$: x_i is a candidate to belong to $\mathcal{D}_\theta$ if $\mathrm{Id}_i = 1$ and it is a candidate to belong to $\mathcal{V}_\theta$ if $\mathrm{Id}_i = 0$. Id is initialised as 1 for all $x_i \in \mathbf{I}$, so that initially all instances are assumed dissimilar until shown otherwise. In each iteration, the algorithm selects an instance x_i from $\mathcal{L}$ in FIFO order and compares it to all other instances in $\mathcal{L}$. If an instance x_j is similar to x_i and $\boldsymbol{\delta}_i = \boldsymbol{\delta}_j$, then, x_j it is removed from $\mathcal{L}$. If x_j is similar to x_i but $\boldsymbol{\delta}_i \neq \boldsymbol{\delta}_j$, violating the similarity assumption, then Id_j is set to 0. When all instances in $\mathcal{L}$ have been compared with x_i, the algorithm removes x_i from $\mathcal{L}$ and adds it to $\mathcal{V}_\theta$ if $\mathrm{Id}_i = 0$, otherwise, adds it to $\mathcal{D}_\theta$. In the next iteration, it selects another instance from the front of $\mathcal{L}$ and repeats the above comparison process. The algorithm terminates when $\mathcal{L}$ is empty, which occurs after $|\mathbf{I}|(|\mathbf{I}| - 1)/2$ comparisons. The critical set of instances $\mathcal{C}_\theta$ is identified as the union of $\mathcal{V}_\theta$ and $\mathcal{D}_\theta$. Notice that if an instance gets $\mathrm{Id}_j = 0$ in the current iteration, it might be detected as redundant later during the comparison process. In this case, it will be removed from $\mathcal{L}$ but not be added to $\mathcal{V}_\theta$. A MATLAB implementation of this algorithm can be found in Alipour (2021).

The value of θ in Algorithm 1 depends on the trade-off between good uniformity and diverse sampling. In the following, we discuss how to assess the uniformity of instances and determine θ to achieve a balance between good uniformity and diverse sampling. To avoid missing important information, we assess the similarity and uniformity of instances only in the high-dimensional space of features but not in the reduced 2-dimensional space. This point will be discussed further in Section 4.1.3. Moreover, since the locations of ViSA instances are potentially unreliable in the feature space, as the features must be inadequate if they are similar but achieving different performance, we do not regard $\mathcal{V}_\theta$ in assessing non-uniform density. Based on the *point-to-point* comparison strategy used in Algorithm 1, we use the simple point-to-point uniformity measure based on the *coefficient of variation* of the *nearest neighbour distances* to assess the uniformity of $\mathcal{D}_\theta$ in the space of features. For an instance $x_i \in \mathcal{D}_\theta$, the nearest neighbour distance, denoted by $d^{NN}(x_i)$, is defined as $d^{NN}(x_i) = \min \left\|\mathbf{f}_i - \mathbf{f}_j\right\|_2, \forall x_j \in \mathcal{D}_\theta, x_j \neq x_i$. Let $\mathcal{D}_\theta^{NN} = \left\{d^{NN}(x_i) \mid x_i \in \mathcal{D}_\theta\right\}$. The *coefficient of variation of the nearest neighbour distances* for $\mathcal{D}_\theta$ is denoted by cv^{NN}, that is,

$$cv^{NN} = \frac{\sigma_{\mathcal{D}_\theta^{NN}}}{\mu_{\mathcal{D}_\theta^{NN}}}.$$

The uniformity of $\mathcal{D}_\theta$ is measured by $\mathrm{u}_{\mathcal{D}_\theta} = 1 - cv^{NN}$ (Gunzburger & Burkardt, 2004) as a reliable uniformity measures (Ong, Kuang, & Ooi, 2012). Notice that $\mathrm{u}_{\mathcal{D}_\theta} \leq 1$, where values near 1 indicate high uniformity of $\mathcal{D}_\theta$. If $|\mathcal{D}_\theta| = 1$, we take $\mathrm{u}_{\mathcal{D}_\theta} = 1$. Since $\mathrm{u}_{\mathcal{D}_\theta} \in \left[1 - cv^{NN}_{\max}, 1\right]$, where $cv^{NN}_{\max}$ is the maximum allowable cv^{NN} in $\mathcal{D}_\theta$, knowing whether $\mathrm{u}_{\mathcal{D}_\theta}$ is suitable depends on the value of $cv^{NN}_{\max}$, which is a little vague. Standardising $\mathrm{u}_{\mathcal{D}_\theta}$ using $\left(\mathrm{u}_{\mathcal{D}_\theta} - \left(1 - cv^{NN}_{\max}\right)\right)/\left(1 - \left(1 - cv^{NN}_{\max}\right)\right)$, makes $\mathrm{u}_{\mathcal{D}_\theta} \in [0, 1]$ easier to interpret.

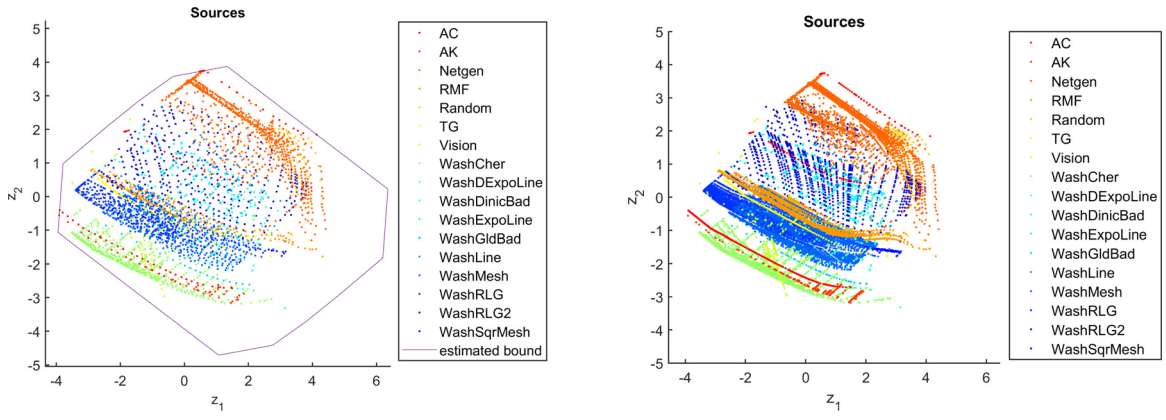
In detecting a good value for θ, if we focus on achieving a high uniformity of $\mathcal{D}_\theta$, the ideal θ might be large, resulting in a small cardinality of $\mathcal{D}_\theta$ in which all instances have the same distances from their nearest neighbours; however this is likely to result in insufficient diversity of instances in the instance space. On the other hand, if we focus on achieving an acceptable diversity of instances, the ideal θ might be small resulting in the maximum possible benchmarks retained in $\mathcal{D}_\theta$, but likely, with low uniformity of $\mathcal{D}_\theta$ and a persistence of oversampling bias. The following rule determines the value of θ to achieve a balance between these conflicting goals. Notice that, if the value of θ is large enough, Algorithm 1 will return $\mathcal{D}_\theta$ with perfect uniformity; in the worst case, just one or a few instances are retained in $\mathcal{D}_\theta$ resulting in $\mathrm{u}_{\mathcal{D}_\theta} = 1$. This means that using some large values of θ, the algorithm can return $\mathcal{D}_\theta$ that satisfies any predetermined uniformity condition such as $\mathrm{u}_{\mathcal{D}_\theta} \geq 0.8$. Now, given an acceptable uniformity threshold u^0, we apply Algorithm 1 on the metadata with different similarity thresholds $\theta_1 < \theta_2 < \cdots < \theta_q$ and regard the following rule.

Threshold rule: the preferred value of $\theta = \theta_h, h = 1, \ldots, q$ is the smallest one such that $\mathrm{u}_{\mathcal{D}_\theta} \geq \mathrm{u}^0$.

Because the value of θ is proportional to $1/\mathcal{D}_\theta$, this rule enables us to retain as many instances as possible while satisfying the user-defined minimum uniformity requirement, i.e., $\mathrm{u}_{\mathcal{D}_\theta} \geq \mathrm{u}^0$.

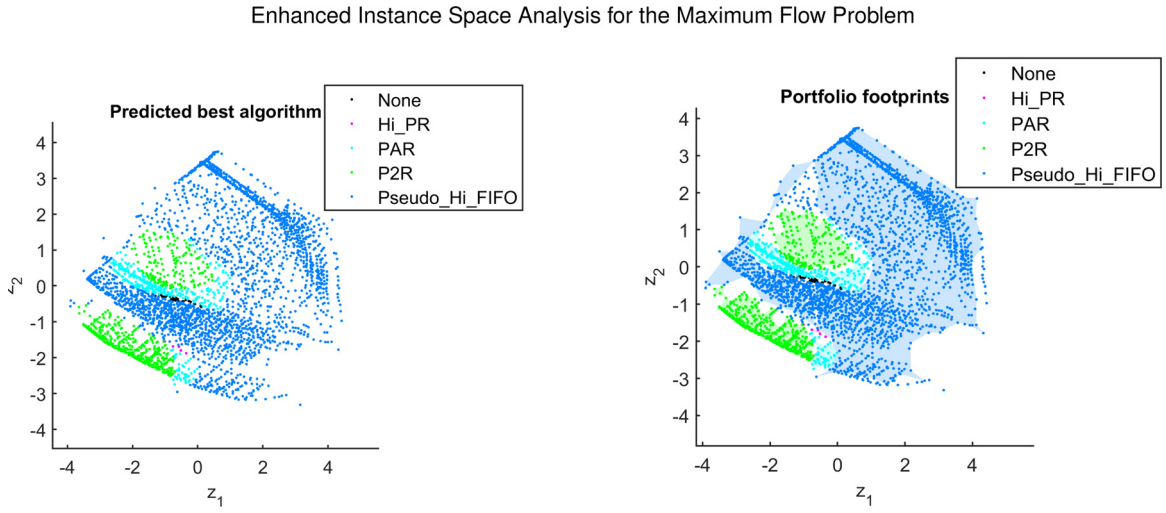
4.1.3. When should the instance selection procedure be applied?

Applying the instance selection procedure before feature selection is crucial. It decreases the bias of non-uniform density on the feature selection and gives the same chance to all features in the metadata to show their importance. The selected instances are then used by ISA to detect the appropriate features and preclude the redundant features. However, precluding the redundant features by ISA changes the dimension of the feature space, implying



(a) Instance distribution of $\mathcal{M}_2$ (b) Instance distribution of $\mathcal{M}_0''$

Fig. 7. Instance spaces created using $\mathcal{M}_2$ and $\mathcal{M}_0''$ with (9). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)



(a) Predicted best algorithm performance (b) Portfolio footprints

Fig. 8. Predicted best algorithm performance and portfolio footprint areas. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

that the previous assessments of the uniformity and the critical instances are no longer valid in the new lower dimensional feature space. Therefore, the instance selection process must be applied again on the induced metadata, which is the original metadata without the precluded features. As a benefit, this decreases the bias of the presence of redundant features on the instance selection. Applying the instance selection process after feature selection, decreases the bias of non-uniform density in the lower dimensional feature space associated with the induced metadata. As a result, the bias in dimensionality reduction used to create the final instance space and the bias in the performance of machine learning classifiers will be decreased at this phase. If the original metadata includes no redundant features, that is, all features are selected through the feature selection process, then, applying the instance selection process before and after feature selection will result in the same $\mathcal{C}_\theta$. In this case, the second phase of applying the instance selection process will be unnecessary.

Some critical instances in the induced metadata might become similar in the 2D space due to information lost during the dimensionality reduction. In this case, non-uniform density might happen in the 2D space as the consequence of the absence of this information. To avoid losing more information, applying the instance selection process is not carried out in the 2D space, but just in the higher dimension feature space where instance similarity is assessed.

4.2. Results

Taking $\mathrm{u}^0 = 0.5$ as an acceptable uniformity threshold and applying Algorithm 1 on $\mathcal{M}_0'$, we obtained $\mathcal{M}_1$. Six features were selected: *Order*, *Size*, *AvNdDg*, *AVScPotNetExcess*, *cvNdDg*, and *ScCapDens*. We kept these features and removed all others from $\mathcal{M}_0'$ to obtain the induced metadata $\mathcal{M}_0''$. Applying Algorithm 1 on $\mathcal{M}_0''$ resulted in $\mathcal{M}_2$. We then applied ISA without the feature selection on $\mathcal{M}_2$.

Fig. 6 shows the uniformity of $\mathcal{D}_\theta$ for different values of θ in $\mathcal{M}_1$ and $\mathcal{M}_2$. Table 2 reports the changes in the number of instances and the uniformity in each phase of applying Algorithm 1, as well as the similarity thresholds obtained by the Threshold rule in each phase using $\mathrm{u}^0 = 0.5$. The ViSA ratio associated with each of the selected set of instances is reported too. According to the Threshold rule of Section 4.1.2, the preferred similarity thresholds are $\theta = 0.20$ and $\theta = 0.15$ to obtain $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively.

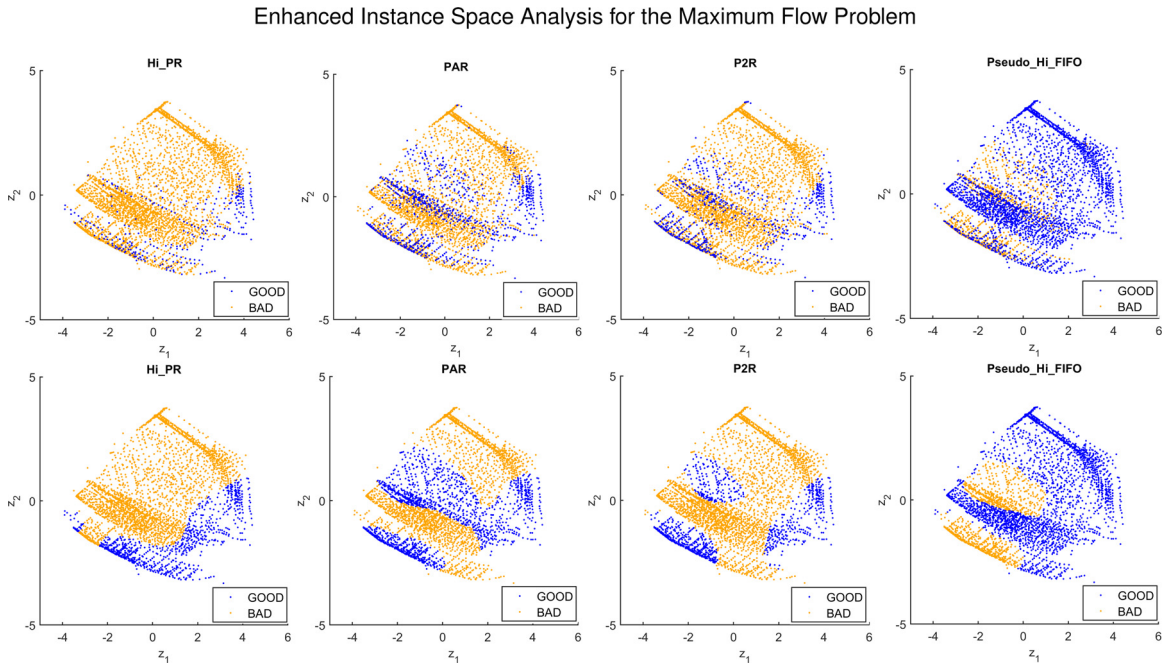


Fig. 9. Actual binary performances of each algorithm on each instance (top row) and their SVM predictions (bottom row); the blue points are instances for which the algorithm is labeled as *good* and the orange points are instances where the algorithm is labeled as *bad*.. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 2
Changes to the uniformity and sample size resulting from applying the instance selection procedure on different metadata sets..

Metadata	θ	$u_{\mathcal{D}_\theta}$	$\lvert\mathcal{D}_\theta\rvert$	$\lvert\mathcal{C}_\theta\rvert$	r_θ	Dimension
$\mathcal{M}_0'$	-	-0.83	25,401	25,401	-	14D
$\mathcal{M}_1$	0.20	0.55	3073	4633	0.34	14D
$\mathcal{M}_0''$	-	-0.95	25,401	25,401	-	6D
$\mathcal{M}_2$	0.15	0.51	2430	3857	0.37	6D

The instance space shown in Fig. is constructed using $\mathcal{M}_2$ and the optimised projection equation given in (9). By contrast, in Fig., $\mathcal{M}_0''$ is projected into the same instance space using the projection Eq. (9). In combination, Fig. 7 illustrates how efficiently Algorithm 1 decreases non-uniform density across the instance space, reducing oversampling and its potential for bias.

$$\begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} = \begin{bmatrix} -0.4208 & -0.0738 \\ -1.0599 & 0.7700 \\ 0.0458 & 0.5908 \\ 0.4662 & 0.3763 \\ -0.3384 & -0.2591 \\ 0.4274 & 0.2908 \end{bmatrix}^\top \begin{bmatrix} Order \\ Size \\ AvNdDg \\ AVScPotNetExcess \\ cvNdDg \\ ScCapDens \end{bmatrix} \tag{9}$$

According to Fig., the empirical boundaries in the bottom and right sides of the instance space are far from any generated instances. As we will see later in Fig. 11, these areas belong to the tiny networks, which are not worth considering as the running times of all algorithms are near-zero, making all algorithms' performances indistinguishable. However, the gaps to the boundaries on other sides of the instance space should be reduced by generating more instances. Supporting this, Fig. 8 shows SVM prediction of the best algorithm for each instance and prediction of the algorithms' footprints, where some small holes exist in the predicted footprints due to insufficient evidence of performance. These results show that more instances must be generated still to obtain a complete instance space. Unfortunately, because of the large storage and computing time requirements, we could not generate and test more of the largest benchmarks. Nevertheless, Figs. 7 and 8 illustrate an acceptable density across the instance space with a good uniformity, meaning that there is a balance between uniformity and diverse sampling. This allows us to explore the instance space with more confidence than the initial metadata ($\mathcal{M}_0$) or the augmented metadata ($\mathcal{M}_0'$), with their inadequate instances and oversampling bias, could afford. Comparing these results with those in Section 3.4, we see the benefits of addressing non-uniform density issues as a priority before attempting to fill sparse areas, so the assessment of sufficiency of the instances is not biased by the non-uniformity. Notice that it is hard to compare different instance spaces resulting from different features, projection matrices, or different set of instances to obtain further insights about the algorithms' performances. Here, we used such a comparison just to see the improvement of diversity and uniformity of instances and their impacts on the feature selection, dimensionality reduction, and footprint detection.

Table 3
Statistical results of SVM models: the first column consists of the name of the algorithms. The second column shows the probability that an algorithm is labeled as good for a given instance. The remaining three column show the quality of SMV models for each algorithm.

Algorithm	$Pr(Good)$	Accuracy	Precision	Recall
Hi_PR	0.081	81.9%	28.2%	80.4%
PAR	0.237	70.2%	42.5%	72.9%
P2R	0.172	80.4%	45.2%	65.5%
Pseudo_Hi_FIFO	0.752	81.5%	89.1%	85.8%

In Fig. 9, the top row shows the actual binary performance and the bottom row provides the SVM prediction of the binary performance for each algorithm in the instance space. These results show that the SVMs have learned to distinguish good and bad performances well.

Statistical results of the SVM models' learning within the final instance space are provided in Table 3. In the presence of numerous instances violating the similarity assumption (ViSA instances), distinguishing the algorithm's labels precisely is challeng-

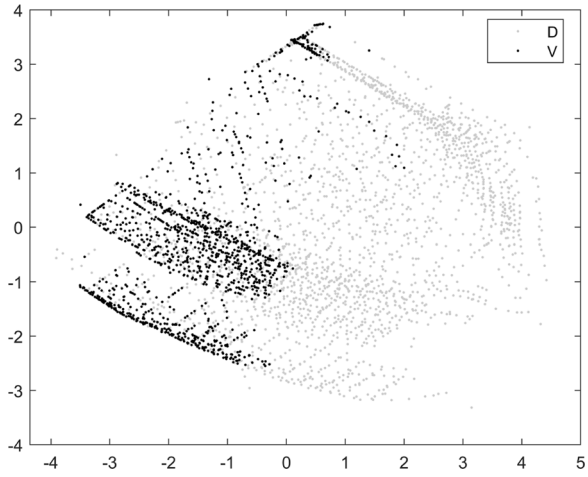


Fig. 10. Locations of $\mathcal{V}_{0.15}$ (black) and $\mathcal{D}_{0.15}$ (gray) in the instance spaces created using $\mathcal{M}_2$ and $\mathcal{M}_0''$ with (9). The black instances violate the similarity assumption, meaning that the performances of algorithms on these instances might not be explained accurately by the current set of features..

ing. Fig. 10 highlights $\mathcal{V}_{0.15}$ and $\mathcal{D}_{0.15}$ in $\mathcal{M}_2$. We note that the regions with a significant number of ViSA instances also correspond to the regions where the SVM models perform less precisely. The low precision of the Push-Relabel family in Table 3 supports this relationship, and the belief that the current features are probably insufficient to accurately describe the relationship to performance for instances in these regions that are mostly occupied by the Push-Relabel algorithms' footprints. The quantity $r_{0.15} = 0.37$ in Table 2 also acknowledges that the set of the selected features remains imperfect. However, this does not prevent us from making predictions for such instances; as we can see in the top row of Fig. 9, the similar behaviours of PAR and P2R in such areas is one of the main reasons for this phenomenon, meaning that we cannot have a preference between PAR and P2R but each of them is preferable than Pseudo_Hi_FIFO and Hi_PR for instances in the relevant regions. Except for the low SVM precision for the Push-Relabel family, other metrics (i.e. accuracy and recall) have acceptable values for all four algorithms. Consequently, the SVM models are reliable enough to discuss other aspects of the final instance space.

Fig. 11 illustrates the trends of the selected features across the space. Comparing these trends with the predicted footprints in Fig. 8, we can see how features impact the algorithms' performances beyond the name of their generator. The size of the footprints in Fig. 8 also reveals that Pseudo_Hi_FIFO is the most powerful algorithm in general, P2R is the next strongest algorithm, PAR is slightly weaker than P2R, and Hi_PR is the weakest algorithm in the portfolio. Table 4 summarises the statistical significance testing for the impact of the features on the good/bad performance of algorithms. Taking the groups 1 and 2 as the sets of instances in which an algorithm is labeled as good and bad respectively, we obtained the *p*-values from the *two-sample t-test* for each pair of feature-algorithm, where the mean of the feature values in group 1 is compared with that in group 2; corresponding to each such *p*-value smaller than 0.05, we calculated the 95% *confidence interval* (CI) using the *one-sample t-test* for the feature values in group 1. These results enable selecting suitable algorithms for unseen problems. For example, Hi_PR is the preferable algorithm for instances with $AvNdDg \in [4.7558 \quad 6.1397]$ or $cvNdDg \in [90.0514 \quad 187.2852]$; if $Size \in 10^7 \times [1.4520 \quad 1.8567]$, $AvNdDg \in [642.9823 \quad 825.7418]$, $AvScPotNetExcess \in [109.5541 \quad 150.4747]$, or $ScCapDens \in [0.0215 \quad 0.0298]$, Pseudo_Hi_FIFO is the best option; beyond these bounds, we might invoke combinations of multiple feature values to distinguish the performances of algorithms (see Section 4.3.2). The large *p*-values of *Order* for Push-Relabel algorithms also demonstrate that the analyses limited to this feature, such as some asymptotic analyses, prohibit getting deep insights into the algorithms' behaviours.

The distribution of the percentage of good algorithms on each instance across the space is depicted in Fig. 12, indicating the loca-

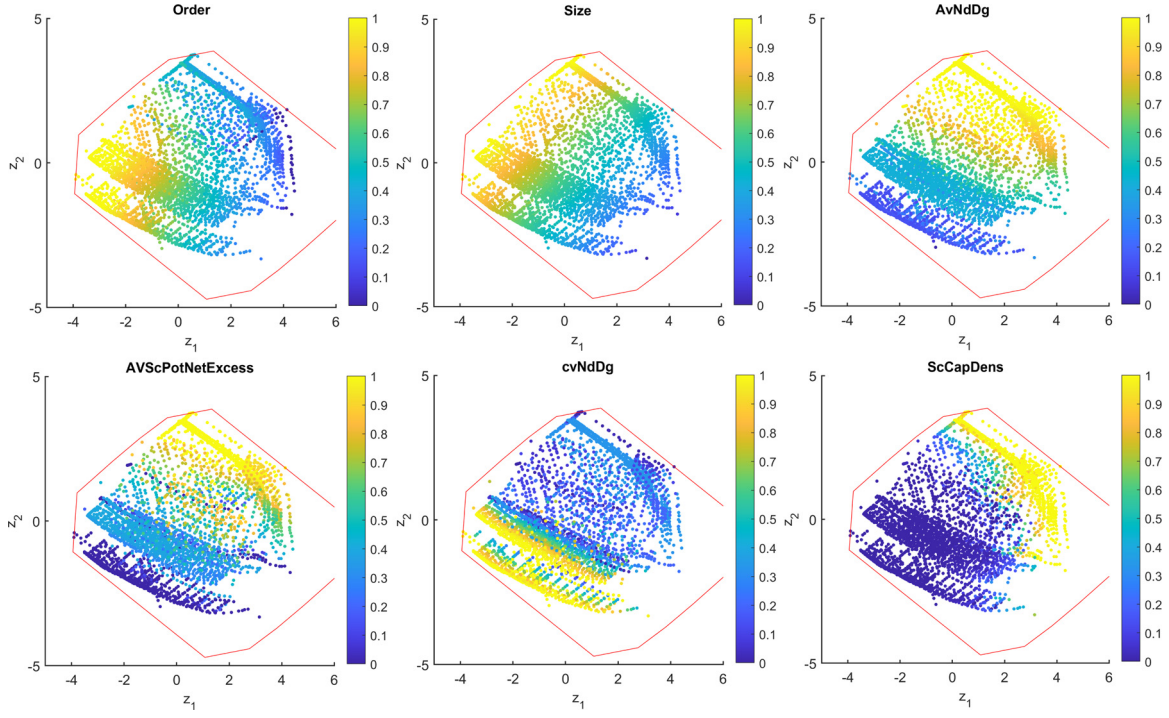


Fig. 11. Distribution of the selected features in the final instance space with a colour scale ranging from scaled minimum (dark blue) to scaled maximum (light yellow) values. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 4
Statistical significance testing for the impact of the features on the good/bad performance of algorithms, *p*-values (top rows) and 95% CIs (bottom rows) of each feature for good performance of each algorithm.

Feature	Hi_PR	PAR	P2R	Pseudo_Hi_FIFO
Order	0.2175	0.7101	0.7098	$3.9712e-10$
	-	-	-	$10^6 \times [1.3779 \quad 1.9798]$
Size	$5.6488e-04$	$1.7018e-08$	$8.7593e-06$	$2.7700e-04$
	$10^6 \times [1.8974 \quad 8.6659]$	$10^6 \times [4.9612 \quad 8.1441]$	$10^6 \times [4.5965 \quad 9.0809]$	$10^7 \times [1.4520 \quad 1.8567]$
AvNdDg	$3.1590e-06$	$1.0164e-16$	$7.2315e-09$	$2.4369e-15$
	[4.7558 6.1397]	[7.6440 68.2496]	[31.8288 199.0035]	[642.9823 825.7418]
AVScPotNetExcess	$2.6246e-04$	$3.4048e-09$	0.0068	$2.6045e-06$
	[0.7775 1.1073]	[0.8083 31.1063]	[13.9620 97.5733]	[109.5541 150.4747]
cvNdDg	$2.3784e-40$	0.0016	0.0330	$9.3394e-09$
	[90.0514 187.2852]	[10.5733 18.7062]	[13.2920 20.8401]	[16.6497 24.3152]
ScCapDens	0.0041	$5.0676e-07$	0.0032	$2.2573e-08$
	[0.0035 0.0057]	[0.0035 0.0080]	[0.0055 0.0142]	[0.0215 0.0298]

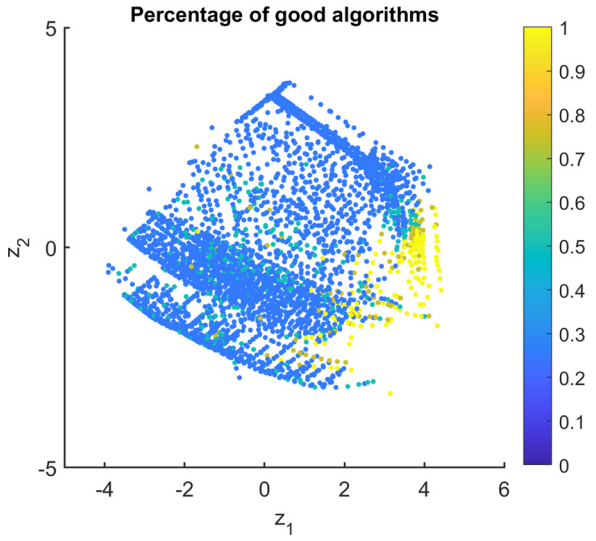


Fig. 12. Distribution of the percentage of good algorithms with a colour scale ranging from scaled minimum (dark blue) to scaled maximum (light yellow) values. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

tions of easy (bottom right) and hard instances for the algorithms. Fig. 13 shows the scaled performance for each algorithm, where the scaling is based on the range of the algorithm's performances. According to our choice of the performance measure and the definition of the performance goodness, Fig. 12 shows that most instances are hard for MFP algorithms; some small instances (networks with small *Size* and *Order* according to Fig. 11) are the only easy instances for all MFP algorithms. Because these algorithms are extremely fast on the small instances, their CPU times are regarded as 0 (smaller than 0.001) indicating that all algorithms are known to be good on such instances. Fig. 13 illustrates this fact. It is believed that most of the benchmark families examined here are easy for MFP algorithms (Ahuja et al., 1997; Goldberg, 2009), which at first glance it is in contradiction with our findings. Since, the term "hard" has not been clarified by research claiming that such instances are easy, we cannot discuss such claims. Nevertheless, different definitions of "hardness" yield different conclusions.

4.3. Further insights into the algorithms' behaviours

In the previous section, we explored how key features appear to impact the behaviours of algorithms. We now explain why these features may cause certain behaviours. To this end, we first describe differences in the underlying mechanism adopted by each algorithm and show that arc/path finding strategies are responsible for the major differences in algorithms behaviours. Then we explain why algorithms exhibit certain behaviours in the instance space by interpreting the relationship between the features and the arc/path searching strategies.

4.3.1. The key difference in the behaviours of MFP algorithms

The four algorithms in our study utilise a variety of strategies for finding the maximum flow (see Appendix A). These strategies are:

(1) using preflows or pseudoflows,
(2) pushing flows through admissible arcs,
(3) relabelling, and
(4) finding admissible arcs/paths.

This raises the question: Which of these strategies has the greatest impact on the behaviour of an algorithm for particular instances?

Push-Relabel algorithms work with preflows, while Pseudo_Hi_FIFO works with pseudoflows. Chandran & Hochbaum (2009) demonstrated that Push-Relabel algorithms can be adopted to work with any kind of initial pseudoflows, but this does not make Push-Relabel algorithms behave like Pseudo_Hi_FIFO, meaning that using preflows or pseudoflows cannot cause a major difference between these algorithms.

Except for P2R, all other algorithms push the maximum possible flows through admissible arcs. If the pushing strategies are responsible for large differences among all algorithms, then Hi_PR, PAR, and Pseudo_Hi_FIFO must behave similarly, but this is not the case. While pushing strategies cause some differences between PAR and P2R, they cannot describe the major differences among all four algorithms.

The relabelling strategies do not seem to cause a major difference either. For example, PAR and P2R use different relabelling strategies, while behaving similarly. Chandran & Hochbaum (2009) claim that the main reason for the substantial differences among Push-Relabel and Pseudoflow algorithms is whether they allow flows to be pushed through arcs (u, v) with $l(u) = l(v)$ (See Appendix A). We see that Hi_PR, PAR, and P2R do not allow flows to be pushed through such arcs, thus, if this is the main reason for the substantial difference between the Pseudoflow and the Push-Relabel algorithms, all algorithms from the Push-Relabel family would be expected to behave similarly, but this is not the case. The global relabeling heuristics used in Push-Relabel algorithms can cause some substantial differences if they are combined with efficient path finding strategies. Nevertheless, these heuristics are not responsible for major differences individually, otherwise, all Push-Relabel algorithms must behave similarly as thy utilise similar heuristics.

Although differences in the above strategies can cause some different behaviours, the main strategies that make significant differ-

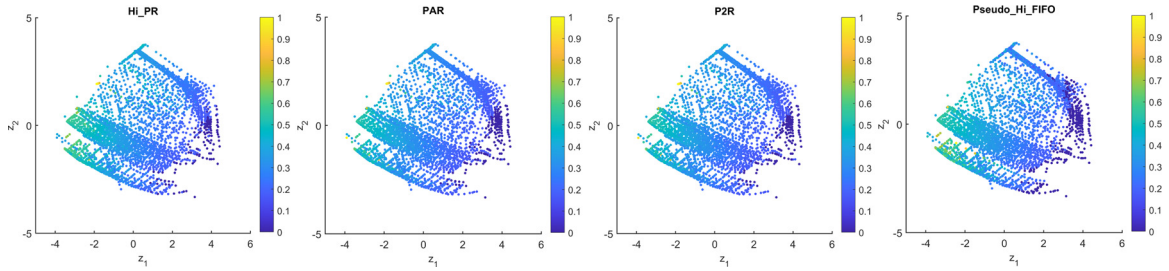


Fig. 13. Scaled algorithm performance for each algorithm relative to its own performance range with a colour scale ranging from scaled minimum (dark blue) to scaled maximum (light yellow) values.. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

ences appears to be the way of finding admissible arcs/paths. Except for Hi_PR, which looks only for the admissible arcs adjacent to a fixed active node using breadth-first-search BFS at each iteration, the three other algorithms look for paths stemming from an active node using depth-first-search DFS at each iteration. This explains why Hi_PR performs quite differently from the three other algorithms. Though the last three algorithms rely on paths, they use different approaches for finding admissible paths. Pseudo_Hi_FIFO uses a forest of the current arcs and finds admissible paths from the root of a strong component to the root of weak components in each iteration using the information of the components. On the other hand, in each iteration, PAR and P2R use only the information of the current arcs and the current nodes explored within the iteration in determining admissible paths. That is, the strategies of PAR and P2R use less information at the cost of having a blind search for the admissible paths. This explains the dramatic difference of Pseudo_Hi_FIFO's behaviour from that of PAR and P2R, and why PAR and P2R behave similarly. In summary, these insights and statistical footprint analysis have provided support for the hypothesis that arc/path finding strategies make a critical difference in explaining the key differences in behaviour of these four algorithms; whether they use BFS or DFS to find admissible arcs/paths, and which strategy they use to find admissible paths if they rely on DFS. The role of the instance features on arc/path finding strategies must be explored.

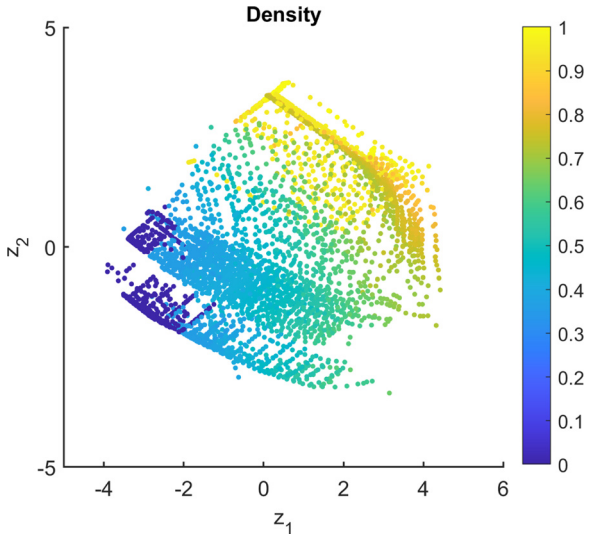


Fig. 14. Distribution of the network density in the instance space with a colour scale ranging from scaled minimum (dark blue) to scaled maximum (light yellow) values.. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

4.3.2. Impact of instance features on the algorithm behaviour

Now, we present some intuitive explanations of the relationship between the features and the admissible arc/path finding strategies. To begin with, we expect that looking for admissible arcs/paths demands less effort on sparse networks than on dense ones. Moreover, if the values of node excesses are low, an algorithm does not need to push back as much flow; thus, its inefficient trial-and-error search for admissible arcs/paths decreases. In this case, searching for admissible arcs/paths might require less effort than searching in networks with high values of node excesses. We call a network as *symmetric/pseudo-symmetric* if its nodes and arcs have the same/similar degrees (i.e. incident arcs) and capacities respectively, otherwise, it is *non-symmetric*. We expect that searching for admissible arcs/paths in any part of a symmetric or pseudo-symmetric network will lead to the similar result, implying that there is no substantial difference between using simple or rich information there. Consequently, in a sparse network with minimal *AvScPotNetExcess* or in a sparse symmetric/pseudo-symmetric network, we expect that a simple searching strategy is more effective than searching with high-information because in the latter, extra operations are required to obtain and store information that is unnecessary. From now onwards, we call the search strategies of Push-Relabel algorithms as *constrained search* because they limit the path length and use simpler information, and the Pseudo_Hi_FIFO's search strategy as *unconstrained search*. Notice that although the Push-Relabel algorithms benefit from the rich information of the global relabeling heuristics periodically, they do not carry/obtain this rich information during all iterations in the first phase. Therefore, they are freed of the extra operations and remain constrained during most iterations. In the following, we first compare the unconstrained search of Pseudo_Hi_FIFO with the constrained searches of the Push-Relabel algorithms, then compare the constrained searches with each other.

We refer to the scaled values of the features as *tiny, small, medium, large*, and *huge* if they fall within the intervals [0 0.2], [0.2 0.4], [0.4 0.6], [0.6 0.8], and [0.8 1] respectively. Fig. 14 illustrates the distribution of the network density, which can be obtained using *Order* and *AvNdDg*. Figs. 14 and 8 reveal that the constrained searches are preferable only on some sparse networks, where $Density \in [0 \quad 0.4]$, and Pseudo_Hi_FIFO's footprint occupies all parts of the instance space with dense networks, where $Density \in (0.4 \quad 1)$. In the bottom of the instance space, the networks are sparse and *AvScPotNetExcess* is tiny (see Fig. 11), thus constrained searches are more effective than the unconstrained search as discussed. This describes why the footprints of PAR and P2R occupy the corresponding region. If a network is sparse and *AvScPotNetExcess* is mall but not tiny, we do not expect a good performance of the constrained searches; this coincides with the Pseudo_Hi_FIFO's footprint in the area with these properties.

The region around the center to the left of the instance space is occupied by the footprints of PAR and P2R. Networks there have small *Density* and tiny *ScCapDens*, meaning that the average arc capacity of each network is tiny or small in comparison

with its median capacity according to the definitions of *ScCapDens* and *ScAvCap* (see Table 1). This implies that the normalised distribution of arc capacities is skewed to the left with a long tail of small capacities, meaning that most arcs have small capacities in each network. Moreover, *cvNdDg* is small or tiny for networks in this area, implying that nodes have almost equal incident arcs in each network. Therefore, it is most likely that these networks are symmetric/pseudo-symmetric; thus, the constrained searches of PAR and P2R are more effective there.

PAR and P2R use the similar constrained searches, while Hi_PR uses a different constrained search. We first compare these algorithms with simple augmenting-path based algorithms (APBAs), which use DFS to find the augmenting paths. Simple APBAs perform many unnecessary non-saturating push operations (see Ahuja et al., 1993); working with preflows mitigates this drawback at the cost of a weaker path search strategy, which is the case with Hi_PR. Therefore, Hi_PR performs well only on networks where the path search is not essential, which is the case with some sparse networks with $AvNdDg \in$ [4.7558 6.1397] according to Table 4. In such networks, some nodes can have large degrees, leading to large *cvNdDg* (see Table 4 and Fig. 11), but most node degrees must be less than 5 to have $AvNdDg \in$ [4.7558 6.1397]. This implies that the number of paths in such networks are small, thus path search strategies must be avoided as they are unnecessary and just apply additional computational costs; this coincides with the small area occupied by the Hi_PR's footprint. On the other hand, PAR and P2R utilise preflows and use partial path searches instead of complete path searches. Therefore, PAR and P2R avoid non-saturating pushes like Hi_PR, but perform the path search better than Hi_PR, making them outperform Hi_PR almost everywhere that a constrained search is required. Moreover, the policy of avoiding unnecessary node activation helps P2R avoid some node excesses in networks with large *AvScPotNetExcess*, making P2R better than PAR in such networks. This policy also enables decreasing arc scans and node relabels (see Appendix D), which is especially important in networks with large and huge sizes, making P2R stronger than PAR there. Although these explain why PAR and P2R are successful in specific regions, their overall performances are not well distinguishable (see Fig. 9 and Table 3) because of their similar searching strategies.

4.3.3. Insights from the fundamental operation counts

In this section, we utilise the fundamental operation counts to compare the subtle differences of MFP algorithms. Here, we present only a summary of the insights and provide the full discussion in Appendix D. The common fundamental operations among the algorithms studied here are *arc scans, node relabels*, and *flow pushes* (through single arcs), which consists of *saturating* and *non-saturating* pushes.

The arc scanning strategy in Pseudo_Hi_FIFO avoids re-scanning arcs efficiently such that it has the least arc scans among the tested algorithms. The monotonicity property (Appendix A.2) also enables Pseudo_Hi_FIFO to avoid unnecessary relabels efficiently, making it competitive with P2R in performing the least relabels. The policy of avoiding some node activation in P2R enables it to perform less arc scans and node relabels than Hi_PR and PAR. This policy together with the global relabeling heuristic makes P2R competitive with Pseudo_Hi_FIFO in minimising node relabels. Optimisations applied on the global relabeling heuristic used in PAR, make its relabeling strategy more effective than that of Hi_PR. The global relabeling heuristic and the short path used in PAR to push flows in each iteration, enables PAR to perform the least non-saturating pushes, and thus the least overall pushes among algorithms. Non-saturating pushes are the major part of the overall pushes performed by the MFP algorithms, hence, they are more important than saturating pushes in practice.

Table 5
Linear regressions of CPU times in terms of the fundamental operation counts, where s, p, and r denote the number of scans, pushes, and relabels respectively.

Algorithm	Linear regression	Adjusted R^2
Hi_PR	$CPU = 0.9221 \times s + 0.0587 \times p$	0.9431
PAR	$CPU = 0.9367 \times s + 0.0381 \times p$	0.9384
P2R	$CPU = 0.8096 \times s + 0.1316 \times p$	0.8478
Pseudo_Hi_FIFO	$CPU = 0.3854 \times s + 0.5749 \times p + 0.0533 \times r$	0.9015

4.3.4. Impacts of the fundamental operation counts on CPU times

Table 5 presents the coefficients and adjusted R^2 for the linear regression of CPU times in terms of the fundamental operation counts (data is normalised and mean-centered). According to Table 5, the greatest proportion of the CPU time in Push-Relabel algorithms is spent on arc scans, a small proportion is spent on pushes, while the time spent on relabeling is negligible. In Pseudo_Hi_FIFO, arc scans and pushes take the greatest proportion of the CPU time, with the proportion of push operations being higher than that of arc scans, and relabeling takes a small proportion of the CPU time. These insights identify potential opportunities for improving MFP algorithms using strategies that can reduce the number of scans in the Push-Relabel algorithms, and reduce the number of pushes and scans in Pseudo_Hi_FIFO.

Interpreting the correlation between the individual fundamental operation counts and the features can be difficult because each operation count is not just impacted by the problem characteristics, but is also impacted by other operations. To handle this issues, all fundamental operations must be regarded in a unit performance measure reflecting their counterpart in the overall performance; and the impact of the features must be investigated on this overall performance. We discussed the CPU time as such a measure in this work, while other options will be discussed in Section 5.2.1.

4.4. Summary of new insights obtained through the enhanced instance space analysis

The following insights would have been hard or impossible to observe with traditional reporting of which algorithm is best on average across the initial set of literature benchmarks.

- Instances. The instances used in this study are diverse enough to compare the performance of MFP algorithms fairly, but there is still scope to extend the benchmarks into a more comprehensive set of instances. Moreover, the set of instances selected by our instance selection algorithm has an acceptable uniformity, thus the conclusions about the algorithms are not unduly influenced by representation bias. Among these instances, tiny and small networks are quickly solved by all tested algorithms, but most of the remaining instances are more challenging, eliciting poor performance from at least half of the tested algorithms. As such, the test instances selected in this paper serve well as a comprehensive, unbiased, and challenging set of benchmarks for studying MFP algorithm performance.
- Features. The performance of MFP algorithms can be well explained by six key features: *Order*, *Size*, *AvNdDg*, *AVScPotNetExcess*, *cvNdDg*, and *ScCapDens*. Although the value of r_θ indicates the potential for additional features to add value to the analyses, the ability of the selected features to capture the behaviours of algorithms illustrates their adequacy for practical analyses of MFP algorithms, and enables insights into how instance features affect algorithm performance as described in Section 4.3.2 and summarized below.
- Algorithms. Based on the size of each algorithm's footprints, we can objectively state that Pseudo_Hi_FIFO is the strongest

algorithm, P2R is slightly stronger than PAR, and Hi_PR is the weakest algorithm across the broadest instance space of MFP. However, the insights offered by ISA, as shown in Table 4 and the footprints in Fig. 8, enable us to draw more nuanced conclusions at a per-instance level. We can summarise in general terms the suitability of different algorithms for various problem instance characteristics as follows. For a sparse symmetric/pseudo-symmetric network, or a sparse network with tiny potential excesses on its nodes, the Push-Relabel algorithms are preferred. For the remaining networks, that is, when a network is non-symmetric, dense, or it is sparse but the potential excesses on its nodes are small on average, Pseudo_Hi_FIFO is the best option. From the territory of the Push-Relabel family, when the size of the network is large or huge, P2R is desirable, otherwise PAR is better, though there is not a significant difference between PAR and P2R in general.

ISA also offers the opportunity to gain insights, not only into which algorithm has strengths or weaknesses for different types of instances, but also to explore why the underlying mechanism employed by the algorithm may or may not be effective. The behaviour of the algorithms in the instance space revealed that arc/path finding strategies are mostly responsible for the different behaviours of the algorithms in practice, while other polices cause subtle and minor differences. Moreover, among the arc/path finding strategies in MFP algorithms, DFS is more effective than BFS. Analysing the fundamental operation counts of the algorithms through the instance space also illustrated that Pseudo_Hi_FIFO has the most effective arc scanning strategy, Pseudo_Hi_FIFO and P2R have the most effective relabeling strategies, and PAR has the best flow pushing strategy. Finally, Table 5 suggests that improving the arc scanning strategies in Push-Relabel algorithms may increase the efficiency of these algorithms, while further improvements of arcs scanning and flow pushing strategies may make Pseudo_Hi_FIFO even more powerful.

5. Discussion and future opportunities

In this section, we discuss the limitations of the instance selection procedure and the metadata that may still impact this analysis. We also discuss the future opportunities that are revealed through our analysis.

5.1. Limitation of the instance selection procedure

In the instance selecting procedure, detecting the redundancy of similar instances is done using the binary performance vector $\boldsymbol{\delta}_i$ on $x_i \in \mathbf{I}$. Arguably, $\boldsymbol{\delta}_i$ is a rough estimation, and the actual performance $\mathbf{y}_i$ could be used instead. Unfortunately, the explicit relationship between $\mathbf{f}_i$ and $\mathbf{y}_i$ for all $x_i \in \mathbf{I}$ is unknown, therefore the performance bounds under any feature perturbation cannot be determined, nor can a fair similarity threshold be established. This is why we used only $\boldsymbol{\delta}_i$ to determine redundant instances.

Between two instances satisfying (6) and (7), Algorithm 1 regards both instances as the same and removes one of them based on the way they are sorted in $\mathcal{L}$. This makes $\mathcal{C}_\theta$ potentially sensitive to the order of instances in $\mathcal{L}$, although we did not observe a significant sensitivity in practice. To overcome this potential limitation, however, the following strategies could be explored in future enhancements to the methodology to consider preferences for which instance to retain:

- Between the realistic and synthetic instances, keep the realistic one;
- Regard the variance of the algorithm performance and keep the instance that yields more variance;
- For each instance, consider the average distance from its several nearest neighbours, then keep the instance with the bigger average distance from its nearest neighbours.

Table 6
Adjusted R^2 for the linear regressions of CPU times in terms of the six selected features versus two features *Size* and *Order*.

Algorithm	Six features	Two features
Hi_PR	0.7985	0.7637
PAR	0.8235	0.7596
P2R	0.8273	0.7680
Pseudo_Hi_FIFO	0.7986	0.7665

5.2. Limitations of the metadata

The collected metadata for this study has naturally influenced the analysis, and we now discuss how future work could augment the metadata to increase the possibility of additional insights.

5.2.1. Performance measure

The common measure for the computational performance of MFP algorithms in the literature is CPU time. Nevertheless, CPU time has two major drawbacks. First, it tells us which algorithm is faster but not why. Second, it depends greatly on the particular details of the computational environment, thus CPU times are often difficult to replicate, which is contrary to the spirit of scientific testing. Without considering different kinds of performance measures, our knowledge about the capabilities of MFP algorithms will be shallow. The quality of the solution, generality of application, memory usage, operation counts, etc. are other measures that should be regarded in future practical analyses.

Ahuja & Orlin (1996) proposed a method for performance measurement based on representative operation counts (ROC) and tested it on MFP algorithms (Ahuja et al., 1997). This approach provides both valuable insights into an algorithm's efficiency, and powerful tools to more fairly compare algorithms in practice. However, ROC has two weaknesses. Firstly, it is based only on the order, size, and average node degree, thus ignores other important features. Secondly, ROC is designed for asymptotic analyses and cannot be used on per-instance analysis, as in ISA. Therefore, ROC cannot be used for our purpose, although it could be adapted to consider more features and extend its suitability for per-instance analyses.

5.2.2. Features

In this paper, we have introduced some new network features to reveal the relationship between MFP algorithms' performances and problem characteristics. Among the six features selected via the automated feature selection process in MATILDA, three of them are our newly introduced features. As observed in Section 4.3.2, these features play a crucial role in describing why the algorithms behave in specific ways. We also observed that though *Density* is an important feature, since it can be obtained from two selected features *Order* and *AvNdDg*, it is a redundant feature in the final set, and the feature selection procedure of ISA removed it correctly. This illustrates the richness of the information carried by the six selected features; the acceptable values of adjusted R^2 for the linear regressions of the CPU times in terms of these six futures, presented in Table 6 acknowledge this fact too. The lower values of adjusted R^2 for the linear regressions with just two features *Size* and *Order* also illustrate the necessity of using all six features and possibly other new features, as discussed in Section 4.2, in the practical analyses of MFP algorithms.

As noted earlier, because of computational limitations and practicalities, we have considered only simple features that can be calculated faster than finding the optimal solutions of the corresponding instances. Nevertheless, it will be useful for future work to consider more complicated features to obtain deeper insights into MFP algorithms.

5.2.3. Benchmarks

Generating specific instances to fill sparse areas in the initial instance space, and selecting a less-biased set of instances with the instance selection algorithm, we obtained an instance space with an acceptable density distribution. Nevertheless, it is obvious that more benchmarks are required in order to generate a perfect instance space for MFP, especially to handle any non-uniform density resulting from under-representation of the sample. However, it may be difficult to fill some sparse areas in the instance space using the existing generators, since their reach through parameter exploration is quite limited according to the efforts demonstrated in this paper. Instead, we may be required to invoke other approaches to generate new benchmarks with controllable features. A methodology is proposed in Bowly, Smith-Miles, Baatar, & Mittelmann (2020) to generate benchmarks with controllable features for general linear programming problems. More generally, a methodology has been proposed for evolving benchmarks that occupy target regions of the instance space using evolutionary programming (Smith-Miles & Bowly, 2015). Utilising these methodologies for generating specific benchmarks for MFP is a promising direction for future work.

New hard instances that enforce algorithms to reach their upper bound complexities are also interesting. It is observed that the number of operations required by MFP algorithms to solve instances from the current families are much less than their expected worst-cases (Ahuja et al., 1997). Buzdalov & Shalyto (2015) conducted a study to bridge this gap using a genetic algorithm. Their approach engages MFP algorithms to assess the quality of each new generation of instances, making it impractical to generate medium or large instances. Nevertheless, their idea is worth exploring to generate more challenging instances.

Without realistic instances relevant to many real-world applications of MFP, judging the practical performance of MFP algorithms will be difficult. Except for the computer vision family of instances, other benchmarks are synthetic, meaning that collating more real-world instances is crucial moving forward.

The benchmark generators proposed for MFP were developed before the appearance of the pseudoflow algorithms, with the aim of challenging the Push-Relabel or augmenting path algorithms. Hence, it is likely that they are not challenging enough for the pseudoflow algorithms. The excellent performance of Pseudo_Hi_FIFO on most instances included in our metadata supports this concern. Accordingly, developing benchmark generators that can challenge the pseudoflow algorithms is necessary in the future.

Achieving a high uniformity of instances across the instance space is a vital goal when generating new benchmarks. Since Algorithm 1 improves the uniformity by removing redundant instances, it cannot be used to achieve a high uniformity if there are inadequate instances in the metadata. In future work, we propose to adapt the instance generation methodology in Smith-Miles & Bowly (2015) to combine with Algorithm 1 and achieve an instance generation and selection methodology that fills sparse areas and achieves a a high uniform density of benchmarks across the instance space efficiently.

Finally, the instance diversity across the instance space is assessed visually in ISA, which is slightly subjective. Splitting the instance space into equal bins and counting the number of instances within each bin is a good idea for devising a statistical measure of assessing the sample diversity in the future work.

6. Conclusions

In this paper, we bridged some major gaps in the practical analyses of MFP algorithms. The lack of scientific testing of MFP algorithms was addressed by the first instance space analysis of MFP. We also introduced some new features of MFP instances to achieve the goal of generating new insights into how the characteristics of MFP instances determine the effectiveness of MFP algorithms. To achieve these aims, we have also collected and generated the most comprehensive set of instances from 17 different families, revealing for the first time their locations in an instance space so that their diversity as a benchmark suite can be assessed and improved to guide future studies of MFP.

We also enhanced the ISA methodology by introducing a new instance selection procedure to improve the uniform density of instances across the instance space, thereby reducing representation bias that can affect subsequent analysis. This procedure has four main advantages. Firstly, it increases the uniformity of the instance space, enabling less bias in feature selection, dimensionality reduction, and the performance of machine learning classifiers used in ISA. Secondly, the instance selection procedure avoids losing important information by regarding the trade-off between the uniformity and the information carried by the instances. Thirdly, it decreases the number of instances remarkably, which ensures computational efficiency, especially in replicating the experiment. While the instance selection procedure does not utilise this advantage, as it considers all existing benchmarks, it returns a filtered metadata set that includes only the most critical instances, retaining the most significant information within the metadata with reduced bias. This enables further experiments to be conducted just with the selected instances and provides a method to identify if any new instances are likely to be redundant. Finally, the new instance selection procedure enables strong tools to be devised for posterior analyses, such as detecting the adequacy of the current features or explaining the precision of machine learning classifiers.

Our analysis has provided new insights into MFP and its algorithms. We have scrutinised the suitability of MFP benchmarks; proposed and assessed features that explain MFP algorithm performance; visualised and understood the strengths and weaknesses of the algorithms across the instance space; explored the hard instances for the tested algorithms; detected that the arc/path finding strategies are responsible for the major differences of the algorithm's behaviours; discovered why features may cause specific behaviours of algorithms; revealed the efficient strategies in minimising the fundamental operations of MFP algorithms; specified the potential areas for further improvements of MFP algorithms and their scientific testing, and provided a method for automated algorithm selection for unseen instances. The enhanced ISA methodology, that includes our new instance selection procedure, has enabled such insights that are hard or impossible from traditional reporting of which algorithm is best on average across a set of benchmarks. We hope that this new per-instance approach of analysing experimental performance via instance space analysis will inform future developments in MFP algorithms and other related topics.

Finally, we note that our instance selection procedure can be extended to problems beyond MFP, and is not tied to ISA: it can be exploited in any experimental analysis to determine a critical minimum set of instances to achieve uniformity across a feature space and thereby reduce representation bias.

Acknowledgements

Funding for this research was provided by the Australian Research Council through grant FL140100012, and the University of Melbourne through a Melbourne Research Scholarship awarded to H. Alipour. This research was supported by The University of Melbourne's Research Computing Services and the Petascale Campus Initiative.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.ejor.2022.04.012

References

Ahuja, R. K., Kodialam, M., Mishra, A. K., & Orlin, J. B. (1997). Computational investigations of maximum flow algorithms. *European Journal of Operational Research, 97*, 509–542.

Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network flows: Theory, algorithms, and applications*. Englewood Cliffs, NJ: Prentice Hall.

Ahuja, R. K., & Orlin, B. J. (1996). Use of representative operation counts in computational testing of algorithms. *INFORMS Journal on Computing, 8*(3), pp.318–330.

Alipour, H. (2021). Halipour8463/ISA_MFP: Feb 2021 release. https://doi.org/10.5281/zenodo.4922867.

Alipour, H., Muñoz, M. A., & Smith-Miles, K. (2021). Instance space analysis for the maximum flow problem: metadata and source codes. https://matilda.unimelb.edu.au/matilda/problems/opt/mfp-mfp. https://doi.org/10.6084/m9.figshare.14761836.v2

Bowly, S., Smith-Miles, K., Baatar, D., & Mittelmann, H. (2020). Generation techniques for linear programming instances with controllable properties. *Mathematical Programming Computation, 12*, 389–415.

Boykov, Y., & Kolmogorov, V. (2004). An experimental comparison of min–cut/max-flow algorithms for energy minimization in computer vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 26*(9), 1124–1137.

Buzdalov, M., & Shalyto, A. (2015). Hard test generation for augmenting path maximum flow algorithms using genetic algorithms: Revisited. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 2121–2128).

Chandran, B., & Hochbaum, D. (2009). A computational study of the pseudoflow and push-relabel algorithms for the maximum flow problem. *Operations Research, 27*(2), 358–376.

Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology, 2*(3), 1–27.

Cherkassky, B. V., & Goldberg, A. V. (1997). On implementing push-relabel method for the maximum flow problem. *Algorithmica, 19*(4), 390–410.

Cheung, T. (1980). Computational comparison of eight methods for the maximum network flow problem. *ACM Transactions on Mathematical Software. Association for Computing Machinery, 6*(1), 1–16.

Clemente, G. P., & Grassi, R. (2018). Directed clustering in weighted networks: A new perspective. *Chaos, Solitons, and Fractals, 107*, 26–38.

Dantzig, G. B. (1951). Application of the simplex method to a transportation problem. *Activity Analysis of Production and Allocation* (pp. 359–373). John Wiley and Sons, NY.

Derigs, U., & Meier, W. (1989). Implementing Goldberg's max-flow algorithm: A computational investigation. *ZOR - Zeitschrift für Operations Research, 33*, 383–403.

Dinic, E. A. (1970). Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Mathematics - Docladi, 11*, 1277–1280.

Ferńandez, A., García, S., Galar, M., Prati, R. C., Krawczyk, B., & Herrera, F. (2018). *Learning from imbalanced data sets*. Cham: Springer.

Fishbain, B., S, H. D., & Mueller, S. (2016). A competitive study of the pseudoflow algorithm for the minimum s-t cut problem in vision. *Journal of Real-Time Image Processing, 11*, 589–609.

Ford, L. R., & Fulkerson, D. R. (1956). Maximal flow through a network. *Canadian Journal of Mathematics, 8*, 399–404.

García, S., Luengo, J., & Herrera, F. (2015). Instance selection. *Intelligent Systems Reference Library, Vol. 72. Springer, Cham.*.

Goldberg, A. V. (2008). The partial augment–relabel algorithm for the maximum flow problem. In *Algorithms - ESA 2008* (pp. 466–477). Berlin: Springer.

Goldberg, A. V. (2009). Two-level push-relabel algorithm for the maximum flow problem. *Springer, Heidelberg, AAIM, LNCS, 5564*(09), 212–225.

Goldberg, A. V., Hed, S., Kaplan, H., Kohli, P., Tarjan, R. E., & Werneck, R. F. (2015). Faster and more dynamic maximum flow by incremental breadth-first search. *In Algorithms - ESA, Berlin, Germany: Springer, pp. 619–630*.

Goldberg, A. V., & Tarjan, R. E. (1988). A new approach to the maximum flow problem. *Journal of the ACM, 35*(4), 921–940.

Gunzburger, M., & Burkardt, J. (2004). Uniformity measures for point samples in hypercubes. *Technical Report*. Florida State University.

Harris, T. E., & Ross, F. S. (1955). Fundamentals of a method for evaluating rail net capacities. *Technical Report*. RAND Corporation.

Hochbaum, D. S. (2001). A new-old algorithm for minimum-cut and maximum-flow in closure graphs. *Networks, 37*(4), 171–193.

Hochbaum, D. S., & Orlin, J. B. (2013). Simplifications and speedups of the pseudoflow algorithm. *Networks, 61.1*, 40–57.

Hooker, J. N. (1995). Testing heuristics: We have it all wrong. *Journal of Heuristics, 1*, 33–42.

Imai, H. (1983). On the practical efficiency of various maximum flow algorithms. *Journal of the Operations Research Society of Japan, 26*, 61–82.

Karzanov, A. V. (1974). Determining the maximal flow in a network by the method of preflows. *Soviet Mathematics Dokladi, 15*, 434–437.

King, V., Rao, S., & Tarjan, R. (1994). A faster deterministic maximum flow algorithm. *Journal of Algorithms, 23*, 447–474.

Lafayette, L., Sauter, G., Vu, L., & Meade, B. (2016). Spartan performance and flexibility: An HPC-Cloud chimera. *OpenStack Summit, Barcelona*. https://doi.org/10.4225/49/58ead90dceaaa.

Muñoz, M. A., Villanova, L., Baatar, D., & Smith-Miles, K. (2018). Instance spaces for machine learning classification. *Machine Learning, 107*(1), 109–147.

Muñoz, M. A., & Smith-Miles, K. (2020). Instance space analysis: A toolkit for the assessment of algorithmic power. 10.5281/zenodo.4484108

Ong, M. S., Kuang, Y. C., & Ooi, M. P. L. (2012). Statistical measures of two dimensional point set uniformity. *Computational Statistics & Data Analysis, 56*, 2159–2181.

Opsahl, T., & Panzarasa, P. (2009). Clustering in weighted networks. *Social Networks, 31*, 155–163.

Orlin, J. B. (2013). Max flows in $O(nm)$ time, or better. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing* (pp. 765–774). New York: ACM Press.

Rice, J. R. (1976). The algorithm selection problem. *Advanced Computer, 15*, 65–118.

Sedeño-Noda, A., González-Sierra, M. A., & González-Martín, C. (2000). An algorithmic study of the maximum flow problem: A comparative statistical analysis. *Top, 8*(1), 135–162. https://doi.org/10.1007/BF02564832.

Smith-Miles, K., Christiansen, J., & Muñoz, M. A. (2020a). Revisiting "where are the hard knapsack problems?" via instance space analysis. *Computers & Operations Research, 128*, 105184.

Smith-Miles, K., Muñoz, M. A., & Neelofar (2020b). Matilda: Melbourne Algorithm Test Instance Library with Data Analytics. Available at https://matilda.unimelb.edu.au.

Smith-Miles, K. A. (2007). Generalising meta-learning concepts: from machine learning to meta-heuristics. In *Proceedings of the 7th Meta-heuristics International Conference, Montreal*.

Smith-Miles, K. A. (2008a). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys, 41*(6), 1–25.

Smith-Miles, K. A. (2008b). Towards insightful algorithm selection for optimisation using meta-learning concepts. *IEEE International Joint Conference on Neural Network*, 4118–4124.

Smith-Miles, K. A., Battar, D., Wreford, B., & Lewis, R. (2014). Towards objective measures of algorithm performance across instance space. *Computers & Operations Research, 45*, 12–24.

Smith-Miles, K. A., & Bowly, S. (2015). Generating new test instances by evolving in instance space. *Computers & Operations Research, 63*, 102–113.

Smith-Miles, K. A., & Lopes, L. B. (2012). Measuring instance difficulty for combinatorial optimization problems. *Computers & Operations Research, 39*, 875–889.

Suresh, H., & V. Guttag, J. (2020). A framework for understanding unintended consequences of machine learning. https://arxiv.org/abs/1901.10002.

Verma, T., & Batra, D. (2012). Maxflow revisited: An empirical comparison of maxflow algorithms for dense vision problems. *BMVC*, 1–12.

Wolpert, D., & Macready, W. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation, 1*(1), 67–82.