
# Improving the performance of stochastic local search for maximum vertex weight clique problem using programming by optimization

Chu, Y.; Luo, C.; Hoos, H.H.; You, H.H.

**Citation**
Chu, Y., Luo, C., Hoos, H. H., & You, H. H. (2023). Improving the performance of stochastic local search for maximum vertex weight clique problem using programming by optimization. *Expert Systems With Applications*, *213*(Part B). doi:10.1016/j.eswa.2022.118913

| | |
|---|---|
| Version: | Publisher's Version |
| License: | Licensed under Article 25fa Copyright Act/Law (Amendment Taverne) |
| Downloaded from: | https://hdl.handle.net/1887/4150052 |

**Note:** To cite this publication please use the final published version (if applicable).

Expert Systems With Applications 213 (2023) 118913

Contents lists available at ScienceDirect

Expert Systems With Applications

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

# Improving the performance of stochastic local search for maximum vertex weight clique problem using programming by optimization

Yi Chu [a], Chuan Luo [b,*], Holger H. Hoos [c], Haihang You [d,e]

[a] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[b] School of Software, Beihang University, Beijing, China
[c] Leiden Institute of Advanced Computer Science, Leiden University, Leiden, The Netherlands
[d] State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[e] Zhongguancun Laboratory, Beijing, China

ARTICLE INFO

*Keywords:*
Maximum clique problem
Stochastic local search
Programming by optimization

ABSTRACT

The maximum vertex weight clique problem (MVWCP) is an important generalization of the maximum clique problem (MCP) that has a wide range of real-world applications. In situations where rigorous guarantees regarding the optimality of solutions are not required, MVWCP is usually solved using stochastic local search (SLS) algorithms, which also define the state of the art for solving this problem. However, there is no single SLS algorithm that gives the best performance across all classes of MVWCP instances, and it is challenging to effectively identify the most suitable algorithm for each class of MVWCP instances. In this work, we follow the paradigm of Programming by Optimization (PbO) to develop a new, flexible and highly parametric SLS framework for solving MVWCP, combining, for the first time, a broad range of effective heuristic mechanisms. By automatically configuring this PbO-MWC framework, we achieve substantial advances in the state of the art in solving MVWCP over a broad range of prominent benchmarks, including two derived from real-world applications in transplantation medicine (kidney exchange) and assessment of research excellence.

## 1. Introduction

Given an undirected graph $G$, a clique $C$ is a subset of vertices of $G$ whose induced subgraph is complete. The maximum clique problem (MCP) is to find a clique $C$ of maximum size $|C|$ in a given graph. MCP is one of the most widely known combinatorial optimization problems; it was one of the first problems proven to be NP-hard, with an NP-complete decision variant (Karp, 1972). The maximum vertex weight clique problem (MVWCP) is an important generalization of the MCP, where each vertex is associated with a positive number representing its weight, and the objective is to find a clique with a maximum weight, *i.e.,* a clique $C$ with a maximum total weight over the vertices contained in $C$. The MVWCP has a wide range of practical applications, including computer vision, pattern recognition, robotics (Ballard & Brown, 1982), broadband network design (Park et al., 1996) and wireless telecommunication (Balasundaram & Butenko, 2006).

In light of the importance of the MCP and MVWCP in theory and practice, considerable effort has been expended to develop effective algorithms for these problems. In practice, there are two popular categories of algorithms for solving the MCP and MVWCP: complete algorithms and incomplete algorithms. Complete algorithms are usually based on an approach known as *branch and bound* (see, *e.g.,* Fang et al., 2016; Jiang et al., 2018, 2017; Östergård, 2001; Yamaguchi & Masuda, 2008). These algorithms use sophisticated techniques to determine tight upper bounds and branching strategies, in order to reduce the search space and accelerate the search process. In contrast, incomplete algorithms, which are mostly based on some form of local search, cannot prove the optimality of candidate solutions. However, the best incomplete algorithms are usually able to find high-quality solutions even for large and challenging instances within a reasonable time (see, *e.g.,* Battiti & Protasi, 2001; Cai & Lin, 2016; Fan et al., 2017; Grosso et al., 2004; Pullan, 2008; Pullan & Hoos, 2006; Pullan et al., 2011; Wang et al., 2016; Wu et al., 2012; Zhou et al., 2017).

Among these incomplete algorithms, MN/TS (Wu et al., 2012) shows an advantage on the well-known BHOSLIB benchmark, where instances are generated in the phase-transition area according to Model RB (Xu et al., 2005) and known to be difficult in theory and practice (Xu et al., 2007), while the strong configuration checking (SCC) strategy underlying the LSCC algorithm (Wang et al., 2016) exhibits stronger

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
*E-mail addresses:* chuyi@ios.ac.cn (Y. Chu), chuanluo@buaa.edu.cn (C. Luo), hh@liacs.nl (H.H. Hoos), youhaihang@ict.ac.cn (H. You).

https://doi.org/10.1016/j.eswa.2022.118913
Received 3 May 2021; Received in revised form 21 September 2022; Accepted 24 September 2022
Available online 3 October 2022
0957-4174/© 2022 Elsevier Ltd. All rights reserved.

performance on the prominent DIMACS benchmark (Johnson & Trick, 1996a), which contains instances from applications in various areas. Overall, there is no single algorithm that performs best on all types of MVWCP instances, and selecting the most appropriate algorithm for a given set of instances is challenging.

Designing an effective local search algorithm for MVWCP involves a large number of design choices, such as (1) how to construct the initial solution, (2) which intensification and diversification strategies to use, (3) how to strike a good balance between intensification and diversification, and (4) when to restart the local search process. As noted earlier, to deal effectively with various types of MVWCP instances, different search strategies appear to be required, yet, current state-of-the-art MVWCP solvers are based on rather restrictive choices. In addition, for a given graph, it is quite challenging to identify the most effective combination of algorithmic strategies.

Recently, a novel algorithm design paradigm dubbed *programming by optimization* (*PbO*) (Hoos, 2012) has been proposed to encourage algorithm developers to embrace and exploit rich design spaces incorporating a broad range of algorithmic techniques, to expose choices that may affect performance as parameters, and to use general-purpose automated configuration techniques to instantiate those choices such that performance for specific classes or sets of problems instances is optimized. Through the use of PbO, major improvements have been achieved in the state-of-the-art for solving a broad range of NP-hard problems, including propositional satisfiability (KhudaBukhsh et al., 2016), mixed integer programming (Hutter et al., 2010) and minimum vertex cover (Luo et al., 2019).

In this work, we present what we believe to be the first application of the PbO paradigm to the MVWCP and achieve major improvements in the state-of-the-art in solving this prominent and important NP-hard combinatorial optimization problem. Our main contributions can be summarized as follows:

- We propose a new PbO-based local search framework for MVWCP dubbed PbO-MWC, which is highly configurable and incorporates a broad range of effective techniques.
- We conduct extensive experiments to compare PbO-MWC against five state-of-the-art solvers on four benchmarks, including two benchmarks that have been widely used in the literature (BHOSLIB and DIMACS), and two benchmarks derived from practical applications (kidney exchange and research excellence assessment). Our empirical results indicate that PbO-MWC outperforms its competitors on all four benchmarks. PbO-MWC significantly improves the performance of state-of-the-art solvers for solving MVWCP on three benchmarks (BHSOLIB, kidney exchange and research excellence assessment).
- Based on our extensive empirical analysis, we provide insights into the efficacy of different local search strategies and mechanisms.

The remainder of this paper is structured as follows. In Section 2, we provide necessary definitions, notations and an introduction to multi-neighborhood search. In Section 3, we give an overview of related work. Then, in Section 4, we present our PbO-MWC framework and provide detailed descriptions of all its core components. In Section 5, we conduct extensive experiments to show the effectiveness of PbO-MWC and analyze the experimental results. Finally, we conclude this paper and give future work in Section 6.

## 2. Preliminaries

Given an undirected graph $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices and $E = \{e_1, e_2, \ldots, e_m\} \subseteq V \times V$ is the set of edges, a clique $C$ is a subset of $V$, such that each pair of vertices in $C$ is connected by an edge. Given an undirected graph $G$, the objective in the maximum clique problem (MCP) is to find a clique $C \subseteq V$ with a maximum number of vertices. Given an undirected vertex-weighted graph $G = (V, E, w)$, where $(V, E)$ is an unweighted graph and $w$ is a weight function that assigns a positive weight $w(v)$ to each vertex $v \in V$, the weight of a clique $C$ is defined as $W(C) := \Sigma_{v \in C} w(v)$. Given a vertex-weighted graph $G = (V, E, w)$, the objective of the maximum vertex weight clique problem (MVWCP) is to find a clique $C \subseteq V$ with maximum weight $W(C)$.

**Algorithm 1:** SLS for MVWCP

```
Input: graph G=(V, E, w), the cutoff time;
Output: C*;
1  initialize C* := ∅, C := ∅;
2  while no termination criteria are met do
3      while V_add(C) ≠ ∅ do
4          v := select a vertex from V_add(C);
5          C := C ∪ {v};
6      if W(C)>W(C*) then
7          C* := C;
8      while (no termination criteria are met) and (no restarting
        criteria are met) do
9          C := a neighboring clique of C;
10         if W(C)>W(C*) then
11             C* := C;
12 return C*;
```

In this paper, we use a pair of vertices $(u, v)$ to represent an edge $e$ in $G = (V, E)$, where $u, v \in V$ are called the two endpoints of $e$. A vertex $u$ is a neighbor of a vertex $v$ in a graph $G = (V, E)$ if there is an edge $(u, v) \in E$. The neighborhood $N(v)$ of a vertex $v \in V$ is its set of all neighbors of $v$, *i.e.,* $N(v) := \{u \in V \mid (u, v) \in E\}$. The degree $d(v)$ of a vertex $v$ is the size of the neighborhood $|N(v)|$.

Stochastic local search (SLS) algorithms for the MVWCP usually build and subsequently modify cliques until termination criteria are met, then return the clique with the maximum weight encountered during the search process. In the process of local search, three kinds of clique move operators are usually performed: $add(u)$, $swap(u, v)$, and $drop(v)$. To formally define these, we introduce three sets of vertices for a given graph $G = (V, E)$ and a clique $C$:

- $V_{add}(C) = \{u \in V \setminus C \mid u \in N(v) \text{ for } \forall v \in C\}$, *i.e.,* the set of vertices, where $u \in V_{add}(C)$ is a neighbor of each $v \in C$, when added to $C$, resulting in a larger clique (In this paper, when $C$ is an empty set, $V_{add}(C) = V$);
- $V_{swap}(C) = \{\langle u, v\rangle \mid u \notin C, v \in C, u \in N(x) \text{ for } \forall x \in C \setminus \{v\}\}$, *i.e.,* the set of vertex pairs $\langle u, v\rangle$, where $u$ is connected to all but one vertex $v$ in $C$, can replace a vertex in $C$ resulting in a new clique of the same size;
- $V_{drop}(C) = \{v \mid v \in C\}$, *i.e.,* the set of all vertices that can be dropped from the given clique, resulting in a new, smaller clique.

For a clique $C$, an $add(u)$ operator moves $C$ to a larger clique $C \cup \{u\}$; a $swap(u, v)$ operator moves $C$ to a same-sized clique $C \setminus \{v\} \cup \{u\}$; a $drop(v)$ operator moves $C$ to a smaller clique $C \setminus \{v\}$.

Two cliques $C$ and $C'$ are called neighbors if $C'$ can be obtained from $C$ by a single $add(u)$, $swap(u, v)$ or $drop(v)$ move operator.

Formally, we define a general framework of SLS algorithms for the MVWCP as shown in Algorithm 1. Starting from an empty set $C$, we construct an initial clique by iteratively selecting a vertex from $V_{add}(C)$ and adding it into $C$, until $V_{add}(C) = \emptyset$ (Lines 3–5); the selection approach is based on different strategies. Then, we iteratively move from one clique to its neighbor until termination criteria are met or some restarting criteria are met (*e.g.,* a fixed number of steps have been iterated) (Lines 8–11).

High-performance SLS algorithms usually balance between two types of strategies: intensification strategies and diversification strategies. In the case of MVWCP, intensification strategies aim to greedily

improve the weight of the clique, *e.g.,* by iteratively moving from current clique to its neighbor with the largest weight. Diversification strategies are used to prevent or overcome search stagnation, usually by moving to a different part of the search space with little or no regard to solution quality (The solution quality here refers to the weight of clique).

Let $C$ be a clique, $W(C)$ denotes the weight of a clique $C$. We use $Ascore(u, C)$ to denote the increment of $W(C)$ after adding $u$ into $C$, *i.e.,* $Ascore(u, C) = W(C \cup \{u\}) - W(C) = w(u)$. We use $Sscore(u, v, C)$ to represent the increment of $W(C)$ after both adding $u$ into $C$ and removing $v$ from $C$, *i.e.,* $Sscore(u, v, C) = W(C\backslash\{v\} \cup \{u\}) - W(C) = w(u) - w(v)$. We use $Dscore(v, C)$ to denote the increment of $W(C)$ after removing $v$ from $C$, *i.e.,* $Dscore(v, C) = W(C\backslash\{v\}) - W(C) = -w(v)$.

Besides, given a graph $G = (V, E, w)$ and a clique $C$, a vertex $v \in V$ has two possible states: inside $C$ or outside $C$. We define the number of steps that has occurred since $v$ last changed its states as the $age$ of $v$, denoted as $age(v)$.

## 3. Related work

In this section, we give an overview of related work, including the existing local search algorithms for MCP and MVWCP, programming by optimization and automated algorithm configuration, which form the basis of the PbO-MWC framework.

### 3.1. Existing local search algorithms for MCP (MVWCP)

Notable progress on local search algorithms for MCP and MVWCP has been made in recent years. In this subsection, we briefly review the most representative and the state-of-the-art local search algorithms. Reactive local search (RLS) (Battiti & Protasi, 2001), dynamic local search (DLS-MC) (Pullan & Hoos, 2006) and cooperating local search (CLS) (Pullan et al., 2011) are designed for MCP. RLS combines local-neighborhood-search with prohibition-based diversification techniques. Starting from an empty clique, RLS explores the search space by two move operators: $add(u)$ and $drop(v)$. As soon as a vertex is added or dropped, it is put into a tabu list and remains prohibited for the next $T$ iterations. The prohibition length $T$ is adjusted through feedback from the previous history of the search process. RLS performs better than its predecessors on DIMACS benchmark. DLS-MC alternates between clique expansion phase and plateau search phase. The expansion phase performs the $add(u)$ move operator. The plateau search phase performs the $swap(u, v)$ move operator. The selection of vertices is based on vertex penalties that are dynamically adjusted during the search process. DLS-MC exhibits excellent performance on DIMACS benchmark. CLS also alternates between clique expansion phase and plateau search phase. CLS integrates four low-level heuristics that are effective for different instance types. The low-level heuristics differ primarily in their vertex selection methods and also the perturbation mechanisms used to overcome search stagnation. CLS improves the state-of-the-art performance for MCP on BHOSLIB benchmark and achieves the performance that is comparable to state-of-the-art algorithms on DIMACS.

Multi-neighborhood tabu search (MN/TS) (Wu et al., 2012) is designed for MCP and MVWCP. Local search with SCC (LSCC) (Wang et al., 2016) and Restart and random walk in local search (RRWL) (Fan et al., 2017) are both developed on the basis of MN/TS and designed for solving MVWCP. These algorithms alternate between clique construction phase and local search phase. In the local search phase, algorithms are based on a combined neighborhood induced by $add(u)$, $swap(u, v)$ and $drop(v)$ move operators. The general prohibition rule in these three algorithms is only to prohibit removed vertices to be moved back to the clique $C$ during the prohibition period, vertices that are in $C$ can be removed without restriction. In the local search phase, MN/TS adopts tabu strategy. LSCC proposes a new prohibition mechanism named Strong Configuration Checking (SCC) that is based on Configuration Checking (CC). SCC mechanism is more restrictive than CC, a prohibited vertex will be lifted prohibition after adding one of its neighboring vertices to clique by an $add(v)$ move operator. RRWL (Fan et al., 2017) proposes a revisiting-based restart strategy and adopts random walk strategy.

These algorithms do not contain various techniques and no single algorithm can perform well on all types of benchmarks. In addition, there is still room to improve the performance of the above algorithms for solving MVWCP on many hard instances from BHOSLIB benchmark and DIMACS benchmark. They also do not perform well on some benchmarks transformed from real-world problems.

### 3.2. Programming by optimization

PbO approach encourages algorithm developers to greatly expand the design space of algorithms by integrating more algorithmic technologies (Hoos, 2012). Algorithm development following the PbO approach usually involves exposing all design options as configuration parameters and searching for design alternatives to key components. The traditional algorithm configuration method is to test relatively few configurations through some experiments. With the progress of optimization and machine learning, solving the algorithm configuration problem as an optimization problem is a trend of algorithm design (Ansótegui et al., 2009; Hutter et al., 2011, 2009). Up to now, PbO-based approaches have shown effectiveness in many problems, including Boolean satisfiability (Hutter et al., 2017; KhudaBukhsh et al., 2016; Luo et al., 2020), mixed integer programming (Hutter et al., 2010; Xu et al., 2011), AI planning (Vallati et al., 2013) and Minimum Vertex Cover (Luo et al., 2019).

### 3.3. Automated algorithm configuration

The ability of complex heuristic algorithms to solve challenging combinatorial problem instances often critically depends on the use of suitable parameter settings (Hutter et al., 2009). Since it may be difficult to seek performance optimization values for these parameters, in recent years, some work has focused on the automated process of determining optimization parameter configurations, such as ParamILS (Hutter et al., 2009), GGA (Ansótegui et al., 2009), F-RACE (Birattari et al., 2010), SMAC (Hutter et al., 2011) and irace (López-Ibáñez et al., 2016). In these automated algorithm configurations, SMAC is a sequential model-based algorithm configurator, that supports conditional parameters. Since SMAC is one of the best-performing algorithm configuration procedures, we utilized it to configure our PbO-MWC framework and the parametric competitors.

## 4. Parametric stochastic local search for solving MVWCP

In order to explore the performance of configurations in configuration space and avoid premature selection of strategies for different benchmarks, we design a stochastic local search framework named PbO-MWC for solving MVWCP, which exposes various strategies as parameters to the configuration procedure for selection. In the following, we first introduce the top-level design of PbO-MWC, then describe its key components, configuration space and default configuration.

### 4.1. The PbO-MWC framework

The top-level design of PbO-MWC can be described as follows: PbO-MWC iteratively executes two phases until the termination criteria are met: initially, PbO-MWC generates an initialized clique; then PbO-MWC iteratively moves a clique to its neighbor by local search step.

PbO-MWC consists of two phases: construction and search. In the construction phase, the initial clique is generated and regarded as the starting point of the search phase. In the search phase, PbO-MWC iteratively performs local search to move the clique to its neighbor. The pseudo-code of PbO-MWC is outlined in Algorithm 2. There is an

**Algorithm 2:** The PbO-MWC Framework

```
Input: graph G = (V, E, w), the cutoff time;
Output: C*;
1  initialize C* := ∅;
2  while no terminating criteria are met do
3      C := init_construction();
4      if W(C)>W(C*) then
5          C* := C;
6      while no terminating criteria are met do
7          if perform_randomwalk then
8              if with probability randomwalk_prob then
9                  C := random_walk_process(C);
10                 continue;
11         C' := C;
12         C := intensification_process(C);
13         if W(C) ≤ W(C') then
14             if perform_restart then
15                 if with probability restart_prob then
16                     break;
17         else if W(C)>W(C*) then
18             C* := C;
19 return C*;
```

outer loop (Lines 2–18) and an inner loop (Lines 6–18). This framework determines whether the algorithm jumps from the inner loop to the outer loop through a Boolean-valued parameter called *perform_restart*. Local search is restarted by reconstructing the initial clique. If the weight of the current clique does not increase after performing *intensification_process*() and *perform_restart* = *True*, the algorithm restarts the local search process with a probability of *restart_prob* (Lines 12–16).

### 4.2. The construction component

For graph $G = (V, E, w)$, to make the construction effective, PbO-MWC adopts three simple effective construction approaches, resulting in three instantiations of the construction component *init_construction* () (Line 3 in algorithm 2):

(i) Randomized approach: starting with an empty vertex set $C$, randomly select a vertex from $V_{add}(C)$ and add it into $C$ until $V_{add}(C) = \emptyset$.

(ii) Weight-based approach: starting with an empty vertex set $C$, randomly select a vertex from $V$ and add it into $C$, then select the vertex with the largest weight from $V_{add}(C)$ and add it into $C$ until $V_{add}(C) = \emptyset$.

(iii) Degree-based approach: starting with an empty vertex set $C$, randomly select a vertex from $V$ and add it into $C$, then select the vertex with the largest degree from $V_{add}(C)$ and add it into $C$ until $V_{add}(C) = \emptyset$.

### 4.3. The random walk component

PbO-MWC utilizes a Boolean-valued parameter called *perform_randomwalk* to determine whether to try to perform a random walk search or not. If *perform_randomwalk* = *True*, then PbO-MWC performs random walk search with a probability of *randomwalk_prob*, where *randomwalk_prob* is a real parameter. This procedure is outlined in Algorithm 3.

### 4.4. The intensification component

The Intensification procedure is outlined in algorithm 4. The *intensification_process*() moves a clique to its neighbor with the maximum weight and follows a prohibition mechanism. It first selects a vertex $v \in V_{add}(C)$ with the largest *Ascore* and $v$ is not forbidden by the prohibition mechanism (Line 1), and selects a vertex pair $\langle u, u' \rangle \in V_{swap}(C)$ with the largest *Sscore* and $u$ is not forbidden by the prohibition mechanism (Line 2–5). If an $add(v)$ move operator is possible (*i.e.,* there exist a vertex in $V_{add}(C)$ that is not forbidden by the prohibition mechanism), it selects the move operator with the largest weight (Line 6–10). On the contrary, it selects a vertex $x \in V_{drop}(C)$ with the largest *Dscore*, it picks the $drop(x)$ move operator if there is no $swap(u, u')$ move operator or *Dscore*>*Sscore* (Line 13–15), otherwise it picks the $swap(u, u')$ move operator(Line 16–17).

**Algorithm 3:** The *random_walk_process* Procedure

```
Input: C;
Output: C;
1  prob := a random integer between 0 and 99;
2  if prob<33 and V_add(C) ≠ ∅ then
3      v := a vertex randomly selected from V_add(C);
4      C := C ∪ {v};
5  else if prob<67 and V_swap(C) ≠ ∅ then
6      ⟨u, v⟩ := a vertex pair randomly selected from V_swap(C);
7      C := C ∪ {u}\{v};
8  else
9      v := a vertex randomly selected from V_drop(C);
10     C := C\{v};
11 return C;
```

**Algorithm 4:** The *intensification_process* Procedure

```
Input: C;
Output: C;
1  v := a vertex with the largest Ascore in V_add(C) and v is not
   forbidden by the prohibition mechanism, breaking ties by a
   breakingTiesRule; if there is no such a vertex, return −1;
2  if perform_BMS then
3      ⟨u, u'⟩ := a vertex pair with the largest Sscore in V_swap(C)
         with BMS strategy and u is not forbidden by the
         prohibition mechanism, breaking ties by a
         breakingTiesRule; if there is no such a vertex pair, return
         ⟨−1, −1⟩;
4  else
5      ⟨u, u'⟩ := a vertex pair with the largest Sscore in V_swap(C)
         and u is not forbidden by the prohibition mechanism,
         breaking ties by a breakingTiesRule; if there is no such a
         vertex pair, return ⟨−1, −1⟩;
6  if v ≠ −1 then
7      if ⟨u, u'⟩ = ⟨−1, −1⟩ or Ascore>Sscore then
8          C := C ∪ {v};
9      else
10         C := C ∪ {u}\{u'};
11 else
12     x := a vertex with the largest Dscore in V_drop(C);
13     if ⟨u, u'⟩ = ⟨−1, −1⟩ or Dscore>Sscore then
14         x := a vertex selected by a selectDropVertexRule from
             V_drop(C);
15         C := C\{x};
16     else
17         C := C ∪ {u}\{u'};
18 return C;
```

**Table 1**
The configuration space of PbO-MWC.

| Parameters | Depended conditions | Parameter type | Value domain | Default value |
|---|---|---|---|---|
| *perform_BMS* | – | Boolean-valued | {*True*, *False*} | *True* |
| *bms_num* | *perform_BMS* = 1 | Integer | [1, 100] | 50 |
| *breaking_ties* | – | Categorical | {0, 1} | 0 |
| *init_construction* | – | Categorical | {0, 1, 2} | 0 |
| *drop_vertex* | – | Categorical | {0, 1, 2} | 0 |
| *randomdrop_prob* | *drop_vertex* = 1 | Categorical | {0.1, 0.2, …, 0.9} | 0.2 |
| *perform_restart* | – | Boolean-valued | {*True*, *False*} | *False* |
| *restart_prob* | *perform_restart* = 1 | Real | [0.0000001, 0.0001] | 0.000001 |
| *perform_randomwalk* | – | Boolean-valued | {*True*, *False*} | *True* |
| *randomwalk_prob* | *perform_randomwalk* = 1 | Real | [0.00001, 0.1] | 0.0001 |
| *tabu_type* | – | Categorical | {0, 1, 2} | 1 |
| *tabu_tenure* | *tabu_type* = 1 or 2 | Integer | [1, 100] | 7 |

In the intensification procedure, local search algorithms correspond to efforts of revisiting promising regions of the search space (Hoos & Stützle, 2004). In this process, algorithms may easily encounter the cycling phenomenon, *i.e.,* returning to a candidate solution that has been visited recently. The cycling problem is an inherent problem of local search as the method does not allow our algorithms to memorize all previously visited candidate solutions. To deal with this severe issue, two fundamental prohibition mechanisms have been proposed to combine with local search : tabu mechanism (Glover, 1989) and configuration checking (CC) (Cai et al., 2011). Tabu mechanism was proposed by Glover (1989), the tabu mechanism forbids reversing the recent changes, where the "strength" of prohibition is controlled by a parameter called *tabu_tenure* (*tt*). Configuration Checking (CC) was proposed by Cai et al. (2011), CC forbids a vertex to be added back into the candidate set until its circumstance information (also called configuration) has been changed. This paper adopts two prohibition mechanisms: one is based on the tabu mechanism and the other is derived from CC. AThe brief descriptions are as follows.

*Tabu-based prohibition mechanism.* The MN/TS algorithm proposes a prohibition rule: a vertex $v$ that leaves the current clique $C$ (by a $swap(u, v)$ or $drop(v)$ move operator) is forbidden to move back to $C$ for the next *tt* iterations. Any vertex in the clique $C$ can be removed from $C$ without restriction (Wu et al., 2012).

For the $swap(u, v)$ move operator, a vertex pair $\langle u, v\rangle \in V_{swap}(C)$, when $v$ is removed from $C$, $v$ is prohibited to be moved back to $C$ for the next *tt_swap* iterations, *tt_swap=random*($|V_{swap}(C)|$) + $T_1$, *random*($|V_{swap}(C)|$) represents a random integer in the range of $[1, |V_{swap}(C)|]$.

For the $drop(v)$ move operator, a vertex $v \in V_{drop}(C)$, when $v$ is removed from $C$, $v$ is prohibited to be moved back to $C$ for the next *tt_drop* iterations, *tt_drop* = $T_1$.

As suggested in the literature (Wu et al., 2012), the default value of $T_1$ is set to 7 in our work.

*CC-based prohibition mechanism.* The LSCC algorithm proposes a prohibition rule called strong configuration checking (SCC) for MVWCP (Wang et al., 2016). The SCC strategy is implemented with a Boolean array named *confChange*, where *confChange*[v] = 1 means $v$ is allowed to be added to the current clique $C$ and *confChange*[v] = 0 means $v$ is forbidden to be added to $C$.

(1) Initially, for each vertex $v$, *confChange*[v] = 1.

(2) When $v$ is added into $C$, *confChange*[u] = 1 for all $u \in N(v)$.

(3) When $v$ is removed from $C$, *confChange*[v] = 0.

(4) When performing a $swap(u, v)$ move operator, where $v$ is removed from $C$, *confChange*[v] = 0.

*Our new prohibition mechanism TabuCC.* In this intensification component, we propose a new prohibition mechanism named TabuCC, which is inspired by tabu strategy and SCC strategy. MVWCP aims to find a clique with the maximum weight, therefore, intuitively, if a vertex $v$ is added to the clique, then its neighbors should also be encouraged to add to the clique (Wang et al., 2016). Based on this idea, we propose a prohibition mechanism. The TabuCC mechanism is worked as follows:

(1) For the $swap(u, v)$ move operator, a vertex pair $\langle u, v\rangle \in V_{swap}(C)$, when $v$ is removed from $C$, $v$ is prohibited to be moved back to $C$ for the next *tt_swap* iterations, *tt_swap=random*($|V_{swap}(C)|$) + $T_1$, *random*($|V_{swap}(C)|$) represents a random integer in the range of $[1, |V_{swap}(C)|]$.

(2) For the $drop(v)$ move operator, a vertex $v \in V_{drop}(C)$, when $v$ is removed from $C$, $v$ is prohibited to be moved back to $C$ for the next *tt_drop* iterations, *tt_drop* = $T_1$.

(3) For the $add(u)$ move operator, a vertex $u \in V_{add}(C)$, when $u$ is added into $C$, for each vertex $v \in N(u)$, lift the prohibition of $v$.

The intensification component provides three prohibition mechanisms for selection, including the tabu mechanism proposed in MN/TS, SCC proposed in LSCC and TabuCC proposed above. The tabu mechanism and TabuCC involve a parameter called $T_1$ (a tabu tenure), which is exposed to the configurator for selection.

For selecting a vertex pair from $V_{swap}(C)$, *intensification_process*() procedure applies a fast and effective strategy named Best from Multiple Selection (BMS), which strikes a balance between quality and complexity and can bring diversity to the search process (Cai, 2015). The BMS strategy can efficiently select a good element from a source set $S$, it randomly selects *bms_num* elements (*bms_num* is an integer parameter) from $S$, and then returns the best one from the *bms_num* elements. The activation of BMS strategy depends on a Boolean-valued parameter *perform_BMS*. If *perform_BMS* = *True*, the BMS strategy is activated. When BMS strategy is activated, the parameter *bms_num* of BMS strategy will be activated.

There are three *selectDropVertexRule* in this component: (i) Random selection; (ii) Weight-based selection; (iii) Perform random selection with a probability of *randomdrop_prob*, otherwise perform weight-based selection. This component includes two *breakingTiesRules*: (i) Breaking ties randomly; (ii) Breaking ties in favor of the vertex with the largest *age*.

### 4.5. Configuration space and default configuration

PbO-MWC is a parametric local search framework and can be configured to various high-performance local search algorithms. We have introduced the top level of our algorithm framework, all its components and the parameters in Sections 4.1–4.4. In Table 1, we give an overview of the full configuration space of PbO-MWC, including all strategies and parameters, as well as the conditions under which strategies and parameters are activated. The default configuration settings of PbO-MWC are shown in Table 2. The default configuration settings of PbO-MWC mostly follow the MN/TS solver, *i.e., breaking_ties* = 0, *init_construction* = 0, *drop_vertex* = 0, *tabu_type* = 1 with *tabu_tenure* = 7. The PbO-MWC framework does not employ the fixed-step restart strategy used by MN/TS; it utilizes a probabilistic restart strategy, and the default *perform_restart* is set to *False*. Since *perform_restart* is set to *False*, a random walk strategy is employed to increase diversification. According to the paper that proposes the BMS strategy, a good balance between quality and complexity can be achieved when *bms_num* = 50 (Cai, 2015).

**Table 2**
The default configuration of PbO-MWC.

| Instantiation | Default configuration |
|---|---|
| Default | *perform_BMS* = *True*, *bms_num* = 50, *breaking_ties* = 0, *init_construction* = 0, *drop_vertex* = 0, *perform_restart* = *False*, *perform_randomwalk* = *True*, *randomwalk_prob* = 1.0E−4, *tabu_type* = 1, *tabu_tenure* = 7 |

### 4.6. A discussion of the PbO-MWC framework

The PbO-MWC framework is developed by following the programming by optimization paradigm, which consists of four components that are discussed in Sections 4.1–4.4, namely, the construction, random walk, intensification, and restart components. Every component includes several strategy options. A strategy option is represented by an abstract function. Each abstract function corresponds to several concrete functions. A concrete function stands for an alternative strategy. PbO-MWC considers abstract functions as parameters and also treats concrete functions as parameter values. By utilizing an automatic algorithm configuration (such as SMAC), our PbO-MWC framework can be instantiated as an algorithm for solving each benchmark, *i.e.,* assigning a concrete function to each abstract function.

Following are the strategy options contained in each component of the PbO-MWC framework:

- The construction component contains one abstract function which corresponds to three concrete functions: *init_construction* = 0 represents the randomized approach, *init_construction* = 1 represents the weight-based approach, and *init_construction* = 2 represents the degree-based approach (Algorithm 2, line 3).
- The random walk component contains one abstract function which corresponds to two concrete functions: The algorithm performs a random walk with a probability of *randomwalk_prob* if *perform_randomwalk* is *True*; otherwise, the random walk process is omitted (Algorithm 2, lines 7–10).
- The intensification component contains four abstract functions: (1) The algorithm utilizes the BMS strategy if *perform_BMS* is *True*; otherwise, the algorithm would traverse all variables in $V_{swap}(C)$ (Algorithm 4, lines 2–5). (2) *breaking_ties* = 0 represents a random tie-breaking rule, and *breaking_ties* = 1 represents a tie-breaking rule that favors the vertex with the largest *age* (Algorithm 4, lines 3 and 5). (3) *tabu_type* points to the prohibition mechanism adopted by the algorithm: *tabu_type* = 0 represents the SCC strategy, *tabu_type* = 1 represents the Tabu strategy, and *tabu_type* = 2 represents the TabuCC strategy (Algorithm 4, lines 3 and 5). (4) *drop_vertex* = 0 represents weight-based selection; *drop_vertex* = 1 represents random selection with a probability of *randomdrop_prob*, and otherwise perform weight-based selection; *drop_vertex* = 2 represents random selection (Algorithm 4, line 14).
- The restart component contains one abstract function: the algorithm adopts the restart process if *perform_restart* is *True* (Algorithm 2, line 14).

The PbO-MWC framework includes strategy options and parameter options designed to solve each benchmark efficiently. In the PbO-MWC framework, an algorithm is formed by assigning a concrete function to each abstract function. For example, in an instantiated concrete algorithm for solving the BHOSLIB benchmark, the main strategy and parameter options are as follows:

In the construction component: the weight-based approach is used (*i.e., init_construction* = 1). The algorithm contains the random walk strategy (*i.e., perform_randomwalk* = *True*, and *randomwalk_prob* = 0.09733547356349166). In the intensification component: the algorithm traverses all variables in $V_{swap}(C)$ (*i.e., perform_BMS* = *False*); the tie-breaking rule used in this component is to tend to the vertex with the largest $age$ (*i.e., breaking_ties* = 1); the prohibition mechanism used is the Tabu strategy (*i.e., tabu_type* = 1, and *tabu_tenure* = *5*); using the weight-based drop vertex selection rule (*i.e., drop_vertex* = 0). In the restart component: the algorithm adopts the probabilistic restart strategy (*i.e., perform_restart* = *True*, and *restart_prob* = 5.016696977394702 E−5).

## 5. Experimental evaluations

To evaluate the efficiency of our proposed PbO-MWC framework and explore the potential of the configuration space, we conduct extensive experiments to compare PbO-MWC against five state-of-the-art solvers on a broad range of MVWCP benchmarks. First, we describe the benchmarks and the competitors. Second, we introduce the configuration protocols used to automatically configure PbO-MWC and its competitors. Third, we describe the experimental setup. Then, we present the experimental results and give some suggestions about which strategies work well on which benchmark. Finally, we conduct experiments to evaluate the effectiveness of different strategies on each benchmark.

### 5.1. The benchmarks

The set of 40 BHOSLIB instances arose from the SAT'04 Competition. The BHOSLIB instances were translated from hard random SAT instances. DIMACS benchmark set was established for the Second DIMACS Implementation Challenge. This set comprises 80 instances from a variety of real-world applications (Johnson & Trick, 1996b). The BHOSLIB and DIMACS benchmarks have been widely used in the recent literature to test new MCP and MVWCP solvers (Fan et al., 2017; Fang et al., 2016; Li et al., 2017; Pullan, 2008; Pullan et al., 2011; Richter et al., 2007; Wang et al., 2016; Wu et al., 2012). The original graphs are unweighted, we obtain the weighted graphs by associating a weight $w(v_i) = i \ mod \ 200+1$ to each vertex $v_i$. This method was initially proposed in Pullan (2008) and has been used as a common method for generating weighted graphs from unweighted graphs to evaluate the solvers for solving MCP and MWCP (Fan et al., 2017; Fang et al., 2016; Jiang et al., 2018, 2017; Wang et al., 2016; Wu et al., 2012).

Besides the above two benchmarks, we evaluate the performance of all the solvers on two real-world application benchmarks. Kidney Exchange Scheme (KES) exists in several countries to increase the number of transplants from living donors to patients with end-stage renal disease. A donor–patient pair contains a patient and a person who is willing to donate to that patient but unable to do so due to a non-compatible problem. Each feasible exchange gives a score that reflects its desirability. Typically, administrators perform matching operations at fixed intervals, with the goal of maximizing the sum of exchange scores. McCreesh et al. (2017) proposed that this optimization problem may be solved by reduction to MVWCP, where each vertex is an exchange (consists of two donor–patient pairs), whose weight is its score. Two exchanges are adjacent if and only if they have no participants in common. The clique stands for a maximally desirable set of donor–patient exchange. Research Excellence Framework (REF) is a system for assessing the quality of research in higher education institutions. In each assessment unit, each staff would submit six publications, from which the organization chooses four. Cooperating authors within the same assessment unit cannot submit a shared publication. MVWCP helps the assessment units find a way to maximize the submission, where each vertex is a choice of four publications from six publications. The clique stands for the set of publications that an assessment unit can provide to the authority. The generation process of these two

**Table 3**
The optimized configurations of PbO-MWC for all benchmarks.

| Benchmark/ Instance family | Optimized configuration |
|---|---|
| BHOSLIB | *perform_BMS* = *False*, *breaking_ties* = 1, *init_construction* = 1, *drop_vertex* = 0, *perform_restart* = *True*, *perform_randomwalk* = *True*, *restart_prob* = 5.016696977394702E−5, *randomwalk_prob* = 0.09733547356349166, *tabu_type* = 1, *tabu_tenure* = 5 |
| DIMACS (MANN family) | *perform_BMS* = *False*, *breaking_ties* = 1, *init_construction* = 1, *drop_vertex* = 1, *perform_restart* = *False*, *perform_randomwalk* = *True*, *randomdrop_prob* = 0.1, *randomwalk_prob* = 0.0021339029487367554, *tabu_type* = 0 |
| DIMACS (except MANN family) | *perform_BMS* = *False*, *breaking_ties* = 1, *init_construction* = 0, *drop_vertex* = 0, *perform_restart* = *True*, *perform_randomwalk* = *True*, *restart_prob* = 3.459685410644107E−5, *randomwalk_prob* = 0.00994485968433248, *tabu_type* = 1, *tabu_tenure* = 8 |
| KES | *perform_BMS* = *True*, *bms_num* = 6, *breaking_ties* = 1, *init_construction* = 0, *drop_vertex* = 2, *perform_restart* = *True*, *perform_randomwalk* = *False*, *restart_prob* = 2.7775287025690946E−5, *tabu_type* = 1, *tabu_tenure* = 30 |
| REF | *perform_BMS* = *True*, *bms_num* = 16, *breaking_ties* = 1, *init_construction* = 0, *drop_vertex* = 1, *perform_restart* = *True*, *perform_randomwalk* = *False*, *randomdrop_prob* = 0.4, *restart_prob* = 9.44211698679448E−6, *tabu_type* = 2, *tabu_tenure* = 8 |

benchmarks is described in McCreesh et al. (2017). The KES benchmark contains 120 instances and the REF benchmark contains 129 instances.[1] In this work, we ignore the instances in which PBO-MWC and all competitors can obtain the best solution within 10 s, so in our experiments we identify and adopt 43 hard instances in the KES benchmark and 24 hard instances in the REF benchmark. We would like to note that the experimental results of all 120 instances in the KES benchmark and all 129 instances in the REF benchmark are shown in the appendix.[2]

### 5.2. Competitors

In this paper, we compare PbO-MWC (the implementation is available online.[2]) against five state-of-the-art solvers, including four SLS solvers and one complete solver:

**SLS solvers:**

MN/TS (Wu et al., 2012) is a high-performance SLS solver based on multi-neighborhood search and tabu mechanism. In our experiments, we used the version of MN/TS made available by its authors.[3]

LSCC and LSCC+BMS (Wang et al., 2016) are two efficient SLS solver that performs well on BHOSLIB and DIMACS. We used the versions of these two solvers that are available online.[4]

RRWL (Fan et al., 2017) is an efficient SLS solver without parameters. We used the version of RRWL made available by its authors.[5]

**Complete solver:**

TSM-MWC (Jiang et al., 2018) is a state-of-the-art complete solver for MVWCP in both small/medium and massive real-world graphs. The source code is available online.[6]

### 5.3. Configuration protocol

In this work, we made use of SMAC (version: 2.10.03) to automatically configure our PbO-MWC framework and the competitors with parameters. In this subsection, we describe the protocol used for PbO-MWC and its competitors. We extracted a training set for each benchmark. For BHOSLIB, the *frb45* family is chosen to be the training set. For DIMACS and REF, we randomly chose an instance from each family. For KES, we randomly chose 10 instances from the benchmark.

For some training sets, SMAC could not find a configuration for some solvers, with the configuration, the solver could reach the known optimal solution at least once within a cutoff time for each run. In order to configure all the solvers in a uniform protocol, we define a new solution quality named *NewSQ*: *NewSQ* = 0-(*solution quality*) + (*solution time*)/1000, The '*solution quality*' represents the maximum weight of the clique found by the solver, and the '*solution time*' represents the running time when the maximum weight clique is found. We used SMAC to minimize *NewSQ*.

Throughout the configuration process, we allowed a 2-day time budget and a cutoff time of 300 CPU seconds for each solver run. For each training set, we performed 25 SMAC runs and obtained 25 optimized configurations. The configuration with the minimum *NewSQ* value on the training set was chosen as the final optimized configuration. The optimized configurations of PbO-MWC for each benchmark are shown in Table 3.

### 5.4. Experimental setup

All the experiments were carried out on a workstation under the operating system CentOS (version: 7.6.1810), with Intel(R) Xeon(R) CPU E5-2620 2.10 GHz CPU, 20MB L3 cache and 128 GB RAM. All solvers were configured using SMAC with the same configuration protocol, except for RRWL and TSM-MWC, which have no parameters. Each local search solver was executed 100 runs on each instance with seeds from 1 to 100. The deterministic algorithm TSM-MWC was performed 1 run on each instance. The cutoff time for each run was set to 3600 s. The best solution-quality known so far is indicated by *solBest*. We report the number of successful runs (reaches *solBest* within cutoff time), denoted by #*Suc*, and the average running time of finding the final solution on each instance, denoted by $t_{avg}$. For one solver on solving one instance, the number of successful runs divided by the number of total runs is expressed as *successRate*, *i.e.*, *successRate* = #*Suc*/100. As TSM-MWC was performed one run on each instance, the *successRate* of TSM-MWC on an instance is 0 or 1. Since the main difference between the results of solvers on DIMACS is the solution-quality, for each solver on each instance of DIMACS, on the 100 runs, we report the maximum weight ($w_{max}$) and average weight ($w_{avg}$) of the cliques found by each solver. For TSM-MWC, we report the weight of the final clique found, denoted by $w_{sol}$, and the time of the final clique found, denoted by *time*. The time unit that is not specified in the results, in **CPU seconds**. In Tables 4–8, the experimental results in **bold** indicate the best performance for an instance or a benchmark.

[1] https://github.com/jamestrimble/max-weight-clique-instances
[2] https://github.com/filyouzicha/PbO-MWC
[3] http://www.info.univ-angers.fr/%7ehao/clique.html
[4] http://ai.nenu.edu.cn/wangyy/Yiyuandata/Download/mwcp.tar.gz
[5] https://github.com/Fan-Yi/Restart-and-Random-Walk-in-Local-Search-for-MVWC-in-Large-Sparse-Graphs
[6] https://home.mis.u-picardie.fr/%7ecli/EnglishPage.html

**Table 4**
Experimental results on BHOSLIB benchmark. For all instances, each solver was performed with its optimized configuration trained on 5 instances in the upper part. For the deterministic algorithm TSM-MWC, $\#Suc = 100$ indicates that TSM-MWC can obtain the best solution within the cutoff time, and $\#Suc = 0$ indicates that TSM-MWC cannot obtain the best solution within the cutoff time. The *time* denotes the time of the final solution found by TSM-MWC.

| Graph | *solBest* | PbO-MWC $\#Suc(t_{avg})$ | MN/TS $\#Suc(t_{avg})$ | LSCC $\#Suc(t_{avg})$ | LSCC+BMS $\#Suc(t_{avg})$ | RRWL $\#Suc(t_{avg})$ | TSM-MWC $\#Suc(time)$ |
|---|---|---|---|---|---|---|---|
| frb45-21-1 | 4760 | **100 (14.459)** | **100** (68.809) | 46 (1349.447) | 42 (1414.568) | 51 (1085.535) | 0 (1029.450) |
| frb45-21-2 | 4784 | **100 (1.759)** | **100** (14.099) | 78 (1326.001) | 70 (1352.767) | 57 (1032.973) | 0 (2733.750) |
| frb45-21-3 | 4765 | **100 (2.565)** | **100** (22.378) | 53 (1607.364) | 51 (1638.062) | 64 (1288.928) | 0 (1563.770) |
| frb45-21-4 | 4799 | **100 (1.245)** | **100** (49.663) | 67 (1442.893) | 59 (1510.165) | 85 (1139.340) | 0 (1329.360) |
| frb45-21-5 | 4779 | **100 (2.294)** | **100** (5.323) | **100** (301.269) | **100** (338.229) | 95 (393.415) | 0 (1772.870) |
| frb30-15-3 | 2995 | 100 (**0.202**) | 100 (0.635) | 100 (13.648) | 100 (14.819) | 100 (4.423) | 100 (1562.590) |
| frb35-17-1 | 3650 | **100 (0.808)** | **100** (4.672) | **100** (91.153) | **100** (107.303) | **100** (38.994) | 0 (3354.780) |
| frb35-17-2 | 3738 | **100 (3.187)** | **100** (28.094) | **100** (151.364) | **100** (173.864) | **100** (196.664) | 0 (3276.140) |
| frb35-17-3 | 3716 | **100 (0.378)** | **100** (4.365) | **100** (42.298) | **100** (49.195) | **100** (29.877) | 0 (2267.320) |
| frb35-17-4 | 3683 | **100 (0.429)** | **100** (4.245) | **100** (328.516) | **100** (406.264) | **100** (136.796) | 0 (3440.670) |
| frb35-17-5 | 3686 | **100 (0.548)** | **100** (1.058) | **100** (20.187) | **100** (22.571) | **100** (19.197) | 0 (2461.840) |
| frb40-19-1 | 4063 | **100 (5.377)** | **100** (11.478) | **100** (341.834) | **100** (438.977) | **100** (497.250) | 0 (3344.290) |
| frb40-19-2 | 4112 | **100 (2.137)** | **100** (23.773) | 99 (656.502) | 99 (747.700) | 98 (879.786) | 0 (3584.080) |
| frb40-19-3 | 4115 | **100 (9.149)** | **100** (72.378) | 97 (913.408) | 94 (1017.261) | 92 (898.788) | 0 (2150.580) |
| frb40-19-4 | 4136 | **100 (1.121)** | **100** (65.162) | 97 (850.168) | 96 (921.804) | 95 (823.619) | 0 (1722.160) |
| frb40-19-5 | 4118 | **100 (2.928)** | **100** (16.579) | **100** (434.633) | **100** (514.314) | 97 (317.954) | 0 (1008.240) |
| frb50-23-1 | 5494 | **100 (119.462)** | 77 (1532.625) | 3 (1643.836) | 1 (1741.232) | 3 (1375.311) | 0 (3040.930) |
| frb50-23-2 | 5462 | **100 (92.994)** | 71 (1154.720) | 6 (1577.357) | 5 (1652.677) | 10 (1372.736) | 0 (319.980) |
| frb50-23-3 | 5486 | **100 (17.533)** | **100** (47.560) | 17 (1703.953) | 11 (1619.835) | 17 (1326.604) | 0 (2949.470) |
| frb50-23-4 | 5454 | **99** (1035.311) | 58 (919.783) | 0 (1792.514) | 0 (1729.724) | 1 (1256.577) | 0 (3560.910) |
| frb50-23-5 | 5498 | **100 (5.399)** | **100** (26.746) | 70 (1578.170) | 62 (1653.499) | 60 (1284.259) | 0 (2748.320) |
| frb53-24-1 | 5670 | **100 (33.530)** | **100** (398.036) | 7 (1666.790) | 3 (1788.058) | 7 (1424.357) | 0 (562.150) |
| frb53-24-2 | 5707 | **100 (98.211)** | 54 (1681.134) | 2 (1756.783) | 3 (1668.641) | 2 (1245.848) | 0 (3453.760) |
| frb53-24-3 | 5655 | **100** (711.933) | 29 (666.191) | 1 (1836.278) | 1 (1960.289) | 0 (1511.852) | 0 (3433.890) |
| frb53-24-4 | 5714 | **100 (77.797)** | 30 (1051.233) | 0 (1730.691) | 0 (1801.601) | 0 (1536.212) | 0 (2215.230) |
| frb53-24-5 | 5659 | **100 (286.630)** | 42 (1664.354) | 0 (1641.510) | 0 (1635.425) | 1 (1314.714) | 0 (1657.180) |
| frb56-25-1 | 5916 | **100 (35.216)** | 97 (935.194) | 1 (1996.991) | 1 (1779.035) | 2 (1459.226) | 0 (3456.430) |
| frb56-25-2 | 5886 | **100 (37.982)** | 35 (1727.162) | 1 (1770.091) | 0 (1798.876) | 1 (1495.152) | 0 (1467.520) |
| frb56-25-3 | 5859 | **98 (706.840)** | 20 (1498.373) | 0 (1868.919) | 0 (1799.464) | 0 (1461.475) | 0 (2619.290) |
| frb56-25-4 | 5892 | **99 (678.962)** | 30 (1719.638) | 0 (1765.438) | 0 (1862.983) | 0 (1434.981) | 0 (2525.100) |
| frb56-25-5 | 5853 | **94 (1022.826)** | 12 (1594.286) | 0 (1625.700) | 0 (1730.110) | 0 (1599.427) | 0 (2425.130) |
| frb59-26-1 | 6591 | **100 (33.607)** | **100** (464.091) | 0 (1693.629) | 0 (1737.817) | 1 (1367.452) | 0 (2930.000) |
| frb59-26-2 | 6645 | **100 (111.328)** | 99 (632.977) | 2 (1875.558) | 0 (1843.966) | 1 (1470.946) | 0 (2124.560) |
| frb59-26-3 | 6608 | **100 (74.141)** | 34 (1534.772) | 0 (1991.458) | 0 (1930.665) | 0 (1513.631) | 0 (1590.650) |
| frb59-26-4 | 6592 | **100** (92.463) | 95 (988.683) | 1 (1678.177) | 1 (1727.378) | 1 (1502.068) | 0 (23.280) |
| frb59-26-5 | 6584 | **100 (279.545)** | 48 (1429.216) | 0 (1652.560) | 0 (1797.136) | 0 (1536.050) | 0 (2806.550) |

We also report the average PAR10 (*i.e.,* running time, if the solver cannot get the *solBest* in a given cutoff time, it counts the running time as 10 times the cutoff time.) of each solver on each benchmark, denoted by *avgPAR10*.

We do not report the instances that all the local search solvers reach a 100% *successRate* with $t_{avg}$<10 s. We would like to note that the experimental results of all instances are shown in the appendix.[2]

### 5.5. Experimental results

#### 5.5.1. Results on BHOSLIB benchmark

Table 4 presents the comparative results of PbO-MWC and its competitors on BHOSLIB benchmark. From Table 4, we can clearly see that our PbO-MWC algorithm stands out as the best solver on the training set and test set. **On training set**, PbO-MWC achieves a 100% *successRate* on all 5 instances. MN/TS, the second best solver also achieves a 100% *successRate* on all training instances, but in terms of running time, the average time of PbO-MWC is 4.464 and the average time of MN/TS is 32.054. The *successRate* of LSCC, LSCC+BMS and RRWL is much lower than 100%. TSM-MWC cannot reach the best solution known on any training instance. **On all 31 test instances**, PbO-MWC achieves a 100% *successRate* for 27 of them, while the figure is 15, 8, 8, 7 and 1 for MN/TS, LSCC, LSCC+BMS, RRWL and TSM-MWC, respectively. On the four instances where PbO-MWC cannot reach the 100% *successRate*, the *successRate* of PbO-MWC is 99%, 98%, 99% and 94% respectively, which is much higher than that of its competitors. The total $\#Suc$ of PbO-MWC is 3090, while the figure is 2331, 1264, 1177, 1189 and 100 for its competitors respectively. In addition, PbO-MWC finds best solutions with the shortest average time on 28 of 31 instances. The average time of PbO-MWC on 31 test instances is 179.935, the figure is 706.426, 1248.068, 1279.758, 1042.975 and 2389.775 for MN/TS, LSCC, LSCC+BMS, RRWL and TSM-MWC respectively.

#### 5.5.2. Results on DIMACS benchmark

Table 5 shows the comparative results of PbO-MWC and its competitors on DIMACS benchmark. From Table 5, PbO-MWC provides a performance advantage in terms of solution-quality and running time. **On training set**, PbO-MWC achieves the best $w_{avg}$ with the shortest average time on 6 of 10 instances. **On all 11 test instances**, PbO-MWC gets the best $w_{max}$ and $w_{avg}$ for all of them. In terms of running time, PbO-MWC has an obvious advantage on most instances, except for three instances where the complete solver has an advantage.

#### 5.5.3. Results on Kidney Exchange Scheme (KES) benchmark

The comparative results of PbO-MWC and its competitors on KES benchmark are illustrated in Table 6. From Table 6, our PbO-MWC algorithm performs much better than its competitors. **On training set**, PbO-MWC is the only solver that achieves a 100% *successRate* with the shortest $t_{avg}$ on all 10 instances. **On all 28 test instances**, PbO-MWC reaches a 100% *successRate* for all of them, while the figure is 11, 17, 13, 14 and 12 for MN/TS, LSCC, LSCC+BMS, RRWL and TSM-MWC, respectively. The total $\#Suc$ of PbO-MWC is 2800, while the figure is 1373, 2217, 2110, 1984 and 1200 for its competitors respectively. In terms of running time, on 22 of the 28 instances, PbO-MWC can achieve *solBest* with $t_{avg}$<10 s. The average time of PbO-MWC on 28 test instances is 7.361, while the figure is 782.708, 617.978, 723.330, 774.523 and 710.339 for its competitors respectively.

**Table 5**
Experimental results on DIMACS benchmark. For the *MANN* instance family, each solver was performed with its optimized configuration trained on MANN_a45 instance. For other instances, each solver was performed with its optimized configuration trained on 9 instances of the instances in the upper part that do not contain MANN_a45.

| Graph | PbO-MWC $w_{max}(w_{avg})$ $t_{avg}$ | MN/TS $w_{max}(w_{avg})$ $t_{avg}$ | LSCC $w_{max}(w_{avg})$ $t_{avg}$ | LSCC+BMS $w_{max}(w_{avg})$ $t_{avg}$ | RRWL $w_{max}(w_{avg})$ $t_{avg}$ | TSM-MWC $w_{sol}$ *time* |
|---|---|---|---|---|---|---|
| MANN_a45 | 34263(34262.59) 1490.467 | 34226(34199.31) 1815.412 | 34256(34254.02) 425.650 | 34258(34253.84) 1291.260 | 34263(34254.72) 357.249 | **34265** 404.800 |
| brock800_4 | 2971(**2971.00**) **42.734** | 2971(2970.98) 774.713 | 2971(2970.80) 1176.934 | 2971(2970.78) 1128.025 | 2971(**2971.00**) 126.596 | 2971 2540.720 |
| C2000.9 | **10999(10999.00)** **101.025** | **10999(10999.00)** 191.816 | **10999**(10951.90) 1919.433 | **10999**(10951.25) 1902.930 | **10999**(10951.41) 1437.638 | 8338 2311.820 |
| c-fat500-10 | 11586(11586.00) 0.248 | 11586(11586.00) 0.059 | 11586(11586.00) <**0.001** | 11586(11586.00) <**0.001** | 11586(11586.00) 0.379 | 11586 0.190 |
| DSJC1000.5 | 2186(2186.00) 0.083 | 2186(2186.00) **0.047** | 2186(2186.00) 5.955 | 2186(2186.00) 5.989 | 2186(2186.00) 1.158 | 2186 54.910 |
| gen400_p0.9_75 | 8006(8006.00) **0.001** | 8006(8006.00) 0.007 | 8006(8006.00) 0.638 | 8006(8006.00) 0.693 | 8006(8006.00) 0.538 | 8006 77.200 |
| hamming10-2 | 50512(50512.00) **0.145** | 50512(50512.00) 0.652 | 50512(50512.00) 0.588 | 50512(50512.00) 0.516 | 50512(50512.00) 0.966 | 50512 43.290 |
| johnson32-2-4 | **2033(2033.00)** **0.003** | **2033(2033.00)** 0.811 | **2033(2033.00)** 0.151 | **2033(2033.00)** 0.154 | **2033(2033.00)** 0.410 | 1891 10.590 |
| p_hat1500-3 | 10321(10321.00) **2.855** | 10321(10321.00) 29.639 | 10321(10321.00) 113.621 | 10321(10321.00) 117.621 | 10321(10321.00) 30.924 | 10321 3336.320 |
| san400_0.9_1 | 9776(9776.00) 3.402 | 9776(9776.00) **1.646** | 9776(9776.00) 2.848 | 9776(9776.00) 3.218 | 9776(9776.00) 3.511 | 9776 75.290 |
| MANN_a27 | **12283(12283.00)** **3.974** | 12282(12276.98) 1377.077 | **12283(12283.00)** 129.249 | **12283(12283.00)** 251.788 | **12283(12283.00)** 270.134 | **12283** 4.400 |
| MANN_a81 | **111355(111342.37)** 1639.896 | 110171(110090.74) 1818.422 | 111302(111250.54) 1639.686 | 111269(111207.88) 1861.101 | 111324(111303.34) 1784.362 | 109970 3202.890 |
| brock400_4 | 3626(3626.00) **0.632** | 3626(3626.00) 0.988 | 3626(3626.00) 12.447 | 3626(3626.00) 13.083 | 3626(3626.00) 1.634 | 3626 136.900 |
| C1000.9 | **9254(9254.00)** 1.214 | **9254(9254.00)** **1.201** | **9254(9254.00)** 177.922 | **9254(9254.00)** 186.512 | **9254(9254.00)** 63.886 | 7477 2806.730 |
| C4000.5 | **2792(2792.00)** **13.808** | **2792(2792.00)** 14.050 | **2792(2792.00)** 77.724 | **2792(2792.00)** 79.793 | **2792(2792.00)** 129.973 | 2502 3497.290 |
| hamming10-4 | **5129(5129.00)** **1.148** | **5129(5129.00)** 2.846 | **5129(5129.00)** 19.140 | **5129(5129.00)** 21.761 | **5129(5129.00)** 23.739 | 4828 1244.040 |
| keller5 | **3317(3317.00)** 0.332 | **3317(3317.00)** **0.245** | **3317(3317.00)** 15.977 | **3317(3317.00)** 18.548 | **3317(3317.00)** 6.088 | 3097 3472.040 |
| keller6 | **8062(8062.00)** **83.535** | **8062(8062.00)** 509.848 | **8062**(7858.60) 1729.344 | **8062**(7862.85) 1895.647 | **8062**(7892.65) 1633.382 | 4793 3564.280 |
| san1000 | 1716(1716.00) 11.055 | 1716(1716.00) 7.871 | 1716(1716.00) 656.871 | 1716(1716.00) 90.903 | 1716(1716.00) 14.471 | 1716 **5.580** |
| san400_0.7_1 | 3941(3941.00) 158.120 | 3941(3941.00) 42.184 | 3941(3941.00) 62.246 | 3941(3941.00) 88.384 | 3941(3941.00) 7.087 | 3941 **1.490** |
| san400_0.7_2 | 3110(3110.00) 245.999 | 3110(3110.00) 75.019 | 3110(3110.00) 197.050 | 3110(3110.00) 223.260 | 3110(3110.00) 15.661 | 3110 **3.890** |

**Table 6**
Experimental results on KES benchmark. For all instances, each solver was performed with its optimized configuration trained on 10 instances in the upper part. For the deterministic algorithm TSM-MWC, #*Suc* = 100 indicates that TSM-MWC can obtain the best solution within the cutoff time, and #*Suc* = 0 indicates that TSM-MWC cannot obtain the best solution within the cutoff time. The *time* denotes the time of the final solution found by TSM-MWC.

| Graph | *solBest* | PbO-MWC #*Suc*($t_{avg}$) | MN/TS #*Suc*($t_{avg}$) | LSCC #*Suc*($t_{avg}$) | LSCC+BMS #*Suc*($t_{avg}$) | RRWL #*Suc*($t_{avg}$) | TSM-MWC #*Suc*(*time*) |
|---|---|---|---|---|---|---|---|
| 83 | 1237688860685 | 100 (**0.895**) | 100 (76.022) | 100 (73.769) | 100 (93.026) | 100 (142.617) | 100 (162.390) |
| 84 | 1100166012937 | 100 (**0.593**) | 100 (170.961) | 100 (5.882) | 100 (12.575) | 100 (10.478) | 100 (189.200) |
| 91 | 1306441900046 | **100** (**4.447**) | 90 (1274.412) | **100** (267.249) | 86 (1148.002) | **100** (348.165) | 0 (112.590) |
| 96 | 1375094325251 | **100** (**1.572**) | 13 (271.969) | **100** (272.134) | 85 (959.731) | **100** (440.349) | 0 (6.290) |
| 97 | 1375144632330 | **100** (**2.045**) | 3 (507.525) | **100** (414.291) | 82 (1196.935) | 73 (1052.616) | 0 (744.600) |
| 105 | 1787797020693 | **100** (**7.879**) | 92 (1073.198) | 74 (1363.477) | 70 (1293.008) | 43 (1177.404) | 0 (119.590) |
| 107 | 1650240659468 | **100** (**4.792**) | 95 (1011.776) | 78 (1196.789) | 52 (1548.089) | 41 (1312.153) | 0 (148.390) |
| 114 | 2406557630478 | **100** (**17.364**) | 0 (1535.306) | 2 (1590.658) | 2 (1693.137) | 2 (1776.499) | 0 (1671.880) |
| 116 | 2131511910416 | **100** (**5.393**) | 0 (1383.749) | 82 (1110.178) | 68 (1349.423) | 38 (1491.430) | 0 (414.780) |
| 119 | 2269068296209 | **100** (**4.812**) | 3 (1220.001) | **100** (679.383) | 99 (743.593) | 75 (1281.739) | 0 (488.960) |

(*continued on next page*)

#### 5.5.4. Results on Research Excellence Framework (REF) benchmark

The results shown in Table 7 indicate that PbO-MWC outperforms its competitors on REF benchmark. **On training set**, PbO-MWC is the only solver that achieves a 100% *successRate* on all 3 instances. On 'ref-60-23-0' instance, PbO-MWC reaches #*Suc* = 100 with $t_{avg}$ = 37.242, MN/TS, the second best solver on this instance, reaches #*Suc* = 87 with $t_{avg}$ = 1045.383. **On all 19 test instances**, PbO-MWC reaches a 100% *successRate* for all of them, while the figure is 18, 12, 14, 7 and 1 for MN/TS, LSCC, LSCC+BMS, RRWL and TSM-MWC, respectively. On 'ref-60-500-0' instance, PbO-MWC is the only solver that can achieve a 100% *successRate*, and the $t_{avg}$ of PbO-MWC is much shorter than that of its competitors. In terms of running time, PbO-MWC achieves *solBest* with the shortest $t_{avg}$ on 18 of 19 instances. The average time of PbO-MWC on 19 test instances is 2.272, while the figure is 43.114, 305.898, 245.693, 595.366 and 1584.664 for its competitors respectively.

**Table 6** (*continued*).

| Graph | *solBest* | PbO-MWC #*Suc*($t_{avg}$) | MN/TS #*Suc*($t_{avg}$) | LSCC #*Suc*($t_{avg}$) | LSCC+BMS #*Suc*($t_{avg}$) | RRWL #*Suc*($t_{avg}$) | TSM-MWC #*Suc*(*time*) |
|---|---|---|---|---|---|---|---|
| 71 | 1306458693642 | 100 (**0.307**) | 100 (0.506) | 100 (3.692) | 100 (2.532) | 100 (10.101) | 100 (194.990) |
| 75 | 893890125832 | 100 (**0.234**) | 100 (25.565) | 100 (78.181) | 100 (51.920) | 100 (162.163) | 100 (0.600) |
| 81 | 1650240634895 | **100** (**10.112**) | 42 (952.869) | 92 (928.713) | 98 (1004.306) | 51 (1255.399) | 0 (994.070) |
| 82 | 1443914440714 | **100** (**2.676**) | 0 (1454.161) | **100** (52.113) | **100** (52.100) | **100** (96.197) | **100** (601.480) |
| 85 | 1100216320013 | 100 (**1.168**) | 100 (3.232) | 100 (33.098) | 100 (63.250) | 100 (71.106) | 100 (14.990) |
| 87 | 1375194939405 | 100 (**1.127**) | 100 (3.583) | 100 (50.560) | 100 (34.520) | 100 (171.926) | 100 (8.060) |
| 89 | 893940457482 | 100 (**0.329**) | 100 (7.763) | 100 (59.544) | 100 (52.238) | 100 (75.254) | 100 (0.790) |
| 92 | 1581403750408 | **100** (**1.738**) | 0 (1159.796) | **100** (544.064) | 93 (1182.126) | 85 (1183.945) | **100** (5.690) |
| 94 | 1031547150353 | 100 (**1.118**) | 100 (2.025) | 100 (106.281) | 100 (49.779) | 100 (137.152) | 100 (77.780) |
| 95 | 1375245246480 | **100** (**2.556**) | **100** (48.995) | **100** (192.375) | **100** (432.595) | **100** (418.615) | 0 (14.880) |
| 98 | 1443947978765 | 100 (**1.156**) | 100 (8.000) | 100 (13.263) | 100 (11.132) | 100 (21.734) | 100 (501.780) |
| 99 | 1237722398735 | **100** (**1.985**) | **100** (11.295) | **100** (18.543) | **100** (19.415) | **100** (38.765) | 0 (974.380) |
| 100 | 1512701018124 | **100** (**1.586**) | 21 (524.223) | **100** (185.291) | **100** (230.710) | **100** (515.888) | 0 (13.090) |
| 101 | 1650207096842 | **100** (**5.288**) | 0 (1246.877) | 83 (1174.490) | 23 (1066.035) | 68 (1473.039) | **100** (20.390) |
| 102 | 1718960136202 | **100** (**2.812**) | 30 (623.679) | **100** (92.241) | 95 (824.607) | **100** (236.558) | 0 (132.650) |
| 103 | 1512701018125 | **100** (**3.859**) | 38 (668.578) | **100** (62.515) | **100** (109.763) | **100** (82.190) | 0 (34.780) |
| 104 | 1512818425875 | **100** (**4.727**) | 40 (1647.883) | 78 (1141.392) | 74 (1244.486) | 18 (1163.635) | 0 (1162.670) |
| 106 | 1375228477454 | **100** (**2.745**) | **100** (140.282) | **100** (821.318) | 97 (1034.791) | 76 (1485.375) | 0 (10.090) |
| 108 | 1581487595537 | **100** (**5.361**) | 99 (450.271) | 99 (657.236) | 85 (1129.557) | 95 (1050.945) | 0 (32.390) |
| 109 | 1718976905230 | **100** (**3.911**) | 1 (1306.312) | **100** (551.843) | 59 (1248.325) | 88 (1310.915) | 0 (3352.550) |
| 110 | 1512768094224 | **100** (**3.801**) | **100** (391.202) | **100** (15.364) | **100** (13.357) | **100** (17.000) | 0 (2269.480) |
| 111 | 2544013377548 | **100** (**57.323**) | 0 (1567.153) | 1 (1567.423) | 0 (1808.095) | 0 (1519.593) | 0 (98.890) |
| 112 | 2475277107217 | **100** (**30.221**) | 0 (1749.538) | 5 (1692.161) | 0 (1550.411) | 0 (1652.289) | 0 (3593.620) |
| 113 | 2200298487823 | **100** (**8.621**) | 0 (1556.683) | 13 (1357.006) | 16 (1540.398) | 3 (1510.060) | **100** (48.390) |
| 115 | 1581588209685 | **100** (**15.338**) | 2 (1787.766) | 30 (1404.084) | 87 (1038.877) | 33 (1581.393) | 0 (3150.560) |
| 117 | 1925303123981 | **100** (**7.908**) | 0 (1474.054) | 97 (1214.428) | 79 (1456.617) | 64 (1177.945) | 0 (974.580) |
| 118 | 2406641475604 | **100** (**15.660**) | 0 (1670.186) | 0 (1715.431) | 0 (1591.963) | 0 (1529.837) | **100** (1443.190) |
| 120 | 2337821335572 | **100** (**12.455**) | 0 (1433.344) | 19 (1570.731) | 4 (1409.346) | 3 (1737.625) | 0 (162.680) |

**Table 7**
Experimental results on REF benchmark. For all instances, each solver was performed with its optimized configuration trained on 3 instances in the upper part. For the deterministic algorithm TSM-MWC, #*Suc* = 100 indicates that TSM-MWC can obtain the best solution within the cutoff time, and #*Suc* = 0 indicates that TSM-MWC cannot obtain the best solution within the cutoff time. The *time* denotes the time of the final solution found by TSM-MWC.

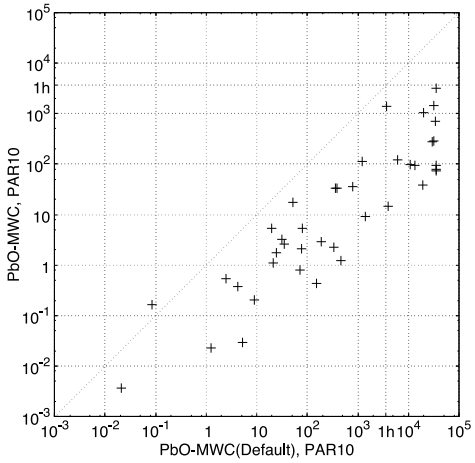
| Graph | *solBest* | PbO-MWC #*Suc*($t_{avg}$) | MN/TS #*Suc*($t_{avg}$) | LSCC #*Suc*($t_{avg}$) | LSCC+BMS #*Suc*($t_{avg}$) | RRWL #*Suc*($t_{avg}$) | TSM-MWC #*Suc*(*time*) |
|---|---|---|---|---|---|---|---|
| ref-60-1000 | 743 | 100 (5.978) | 100 (**0.096**) | 100 (2.604) | 100 (2.815) | 100 (3.587) | 100 (277.880) |
| ref-60-230-0 | 506 | **100** (**37.242**) | 87 (1045.383) | 0 (770.114) | 0 (758.782) | 0 (1240.335) | 0 (2477.980) |
| ref-60-500-7 | 700 | **100** (**5.511**) | **100** (58.170) | 2 (57.438) | 2 (76.478) | 0 (39.422) | 0 (399.550) |
| ref-60-10000 | 768 | 100 (14.181) | 100 (**0.032**) | 100 (0.097) | 100 (0.118) | 100 (0.527) | 100 (12.890) |
| ref-60-230-1 | 506 | **100** (**14.083**) | **100** (52.042) | 2 (259.120) | 0 (201.255) | 0 (992.196) | 0 (830.790) |
| ref-60-230-2 | 524 | **100** (**0.030**) | **100** (0.249) | **100** (337.901) | **100** (260.969) | 98 (629.556) | 0 (3054.720) |
| ref-60-230-3 | 502 | **100** (**0.072**) | **100** (0.244) | **100** (173.394) | **100** (194.756) | 97 (800.821) | 0 (1781.070) |
| ref-60-230-4 | 504 | **100** (**0.098**) | **100** (0.712) | 98 (855.686) | **100** (499.303) | 61 (1091.397) | 0 (1511.560) |
| ref-60-230-5 | 503 | **100** (**0.280**) | **100** (0.339) | **100** (362.522) | **100** (429.003) | 82 (1048.387) | 0 (128.590) |
| ref-60-230-6 | 505 | **100** (**0.027**) | **100** (0.091) | **100** (135.267) | **100** (79.723) | **100** (548.991) | 0 (3351.150) |
| ref-60-230-7 | 506 | **100** (**2.353**) | **100** (6.780) | 14 (432.156) | 15 (429.282) | 3 (936.927) | 0 (2569.800) |
| ref-60-230-8 | 494 | **100** (**0.149**) | **100** (0.310) | **100** (355.894) | **100** (364.748) | 93 (1022.198) | 0 (1219.250) |
| ref-60-230-9 | 526 | **100** (**0.352**) | **100** (2.677) | 98 (1065.225) | **100** (599.256) | 61 (1003.228) | 0 (44.490) |
| ref-60-300 | 599 | **100** (**0.035**) | **100** (0.232) | **100** (183.132) | **100** (103.767) | 99 (708.523) | 0 (3567.480) |
| ref-60-500-0 | 704 | **100** (**9.023**) | 48 (735.908) | 0 (353.410) | 0 (121.051) | 0 (325.049) | 0 (3552.830) |
| ref-60-500-1 | 709 | **100** (**0.020**) | **100** (0.090) | **100** (9.942) | **100** (3.448) | **100** (11.789) | 0 (908.640) |
| ref-60-500-2 | 702 | **100** (**0.834**) | **100** (12.230) | 4 (145.902) | 3 (155.079) | 1 (254.645) | 0 (68.080) |
| ref-60-500-4 | 690 | **100** (**0.168**) | **100** (0.727) | **100** (339.175) | **100** (109.800) | **100** (804.839) | 0 (500.710) |
| ref-60-500-6 | 715 | **100** (**0.035**) | **100** (0.089) | **100** (8.521) | **100** (6.583) | **100** (18.712) | 0 (3161.160) |
| ref-60-500-8 | 714 | **100** (**0.021**) | **100** (0.100) | **100** (286.912) | **100** (123.516) | **100** (850.038) | 0 (411.690) |
| ref-60-500-9 | 704 | **100** (**0.923**) | **100** (5.372) | 17 (366.494) | 47 (905.199) | 9 (204.615) | 0 (642.830) |
| ref-60-500 | 704 | **100** (**0.476**) | **100** (0.940) | **100** (141.320) | **100** (81.310) | **100** (59.510) | 0 (2790.890) |

**Table 8**
The *avgPAR10* on each benchmark.

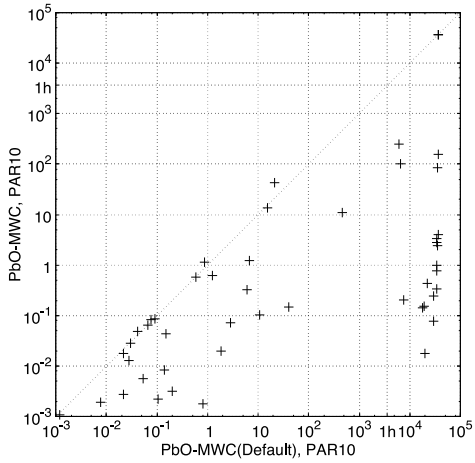
| Benchmark | *Num* | PbO-MWC *avgPAR10* | MN/TS *avgPAR10* | LSCC *avgPAR10* | LSCC+BMS *avgPAR10* | RRWL *avgPAR10* | TSM-MWC *avgPAR10* |
|---|---|---|---|---|---|---|---|
| BHOSLIB | 40 | **228.682** | 7234.972 | 18707.556 | 19151.201 | 18743.424 | 32519.003 |
| DIMACS | 80 | **903.982** | 1379.709 | 1885.051 | 1897.256 | 1770.183 | 5113.066 |
| KES | 43 | **6.016** | 16340.106 | 6573.983 | 8329.434 | 9931.283 | 20242.079 |
| REF | 24 | **3.830** | 1054.483 | 10178.544 | 9666.741 | 12286.660 | 33012.115 |

### 5.5.5. An overview of results on all benchmarks

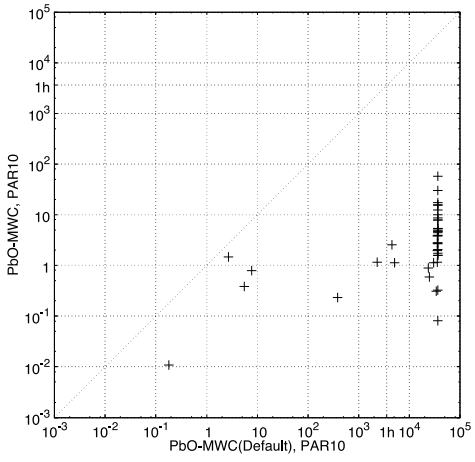
We summarize all the results in Table 8. Table 8 shows that PbO-MWC outperforms all its competitors in terms of *avgPAR10* on all four benchmarks. On BHOSLIB, DIMACS, KES and REF, the ratios of *avgPAR10* of the best performing competitor to *avgPAR10* of PbO-MWC are 31.64, 1.53, 1092.75 and 275.32, respectively. The state-of-the-art



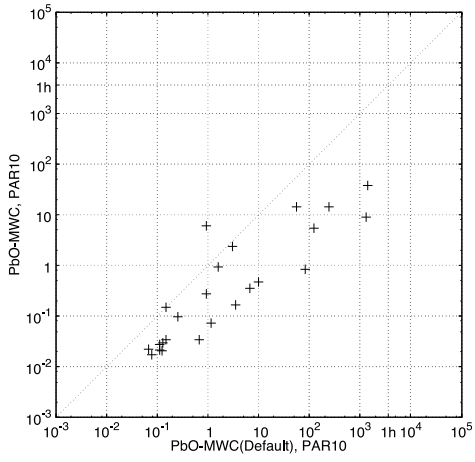

(a) PAR10 between PbO-MWC and PbO-MWC (Default) on BHOSLIB.




(b) PAR10 between PbO-MWC and PbO-MWC (Default) on DIMACS.




(c) PAR10 between PbO-MWC and PbO-MWC (Default) on KES.




(d) PAR10 between PbO-MWC and PbO-MWC (Default) on REF.

**Fig. 1.** Scatter plots of PAR10 between PbO-MWC and PbO-MWC(Default) on four benchmarks. (1 h [3600 seconds] is the cutoff time of each run.).

solving for MVWCP on the BHOSLIB benchmark and two benchmarks that are transformed from real-world problems have been improved remarkably.

#### 5.5.6. The effect of automatically configuring PbO-MWC

To illustrate the effect of automatic configuration of PbO-MWC, we report the performance comparison between PbO-MWC with optimized configuration (PbO-MWC) and PbO-MWC with default configuration (PbO-MWC (Default)) on the four benchmarks, as shown in Fig. 1. Fig. 1 clearly illustrates that optimized configuration leads to performance improvements on a large majority of instances on all four benchmarks.

#### 5.5.7. The suggestions on the effectiveness of different strategies

Based on the extensive experiments, we made some suggestions. We presume that random walk strategy plays an important role on BHOSLIB. Among the three prohibition mechanisms utilized in PbO-MWC, SCC strategy performs best on DIMACS (especially on MANN family instances). In our experiments, all efficient configurations for KES contain BMS strategy with a parameter *bms_num* that is less than 10. We consider an effective framework on REF benchmark has the following properties: restart local search with a lower probability and adopt BMS strategy with a lower *bms_num*.

### 5.6. A discussion of the effectiveness of different strategies

In this section, we empirically evaluate the effect of different strategy options on the performance of the PbO-MWC framework on each benchmark.

*Experimental methodology and setup.* In the experiment, PbO-MWC with optimized configuration ( Table 3, determined by SMAC) is used as the baseline algorithm for each benchmark. Then, we obtain an alternative configuration by replacing one strategy in the optimized configuration, resulting in a new algorithm corresponding to the alternative configuration (If the parameters in the alternative strategy do not involve the optimized configuration, we use the default parameter settings; otherwise, we adopt the parameter settings in the optimized configuration). The PbO-MWC framework has seven strategy options ( Table 1), corresponding to ten strategies different from those in the optimized configuration. In our experiments, we compare the baseline algorithm with ten alternative algorithms. The greater the performance difference between the alternative algorithm and the baseline algorithm, the greater the impact of the strategy option corresponding to the alternative strategy on the performance of the algorithm.

Each alternative algorithm was executed 100 runs on each instance with seeds from 1 to 100. The cutoff time for each run was set to 3600 s. We use *config* to denote the configuration, and use *optimized* to denote the optimized configuration. For BHSOLIB, DIMACS (except

**Table 9**
Comparison between PbO-MWC (with the optimized configuration) and the alternative algorithms on BHOSLIB, DIMACS (except MANN family), KES, and REF benchmarks.

| Strategy options | BHOSLIB | | DIMACS (except MANN) | | KES | | REF | |
|---|---|---|---|---|---|---|---|---|
| | *config* | *avgPAR10* | *config* | *avgPAR10* | *config* | *avgPAR10* | *config* | *avgPAR10* |
| | *optimized* | 228.682 | *optimized* | 8.855 | *optimized* | 6.016 | *optimized* | 3.830 |
| *init_construction* | 0 | 177.675 | 1 | 527.373 | 1 | 6.351 | 1 | 8.180 |
| | 2 | 201.156 | 2 | 988.412 | 2 | 3.624 | 2 | 4.032 |
| *perform_randomwalk* | *False* | 16164.274 | *False* | 51.907 | *True* | 5.715 | *True* | 3.036 |
| *perform_BMS* | *True* | 284.497 | *True* | 12.074 | *False* | 21030.383 | *False* | 1350.912 |
| *breaking_ties* | 0 | 201.227 | 0 | 8.949 | 0 | 8.448 | 0 | 5.843 |
| *tabu_type* | 0 | 1363.928 | 0 | 12.409 | 0 | 15.752 | 0 | 3583.880 |
| | 2 | 264.836 | 2 | 183.678 | 2 | 4.013 | 1 | 4.154 |
| *drop_vertex* | 1 | 202.801 | 1 | 8.915 | 0 | 8.258 | 0 | 3.775 |
| | 2 | 190.379 | 2 | 8.416 | 1 | 7.733 | 2 | 5.397 |
| *perform_restart* | *False* | 236.803 | *False* | 3324.071 | *False* | 5.020 | *False* | 546.336 |

**Table 10**
Comparison between PbO-MWC (with the optimized configuration) and the alternative algorithms on DIMACS (MANN family) instances.

| Strategy options | *config* | MANN_a9 | MANN_a27 | MANN_a45 | MANN_a81 |
|---|---|---|---|---|---|
| | | $w_{avg}$ ($t_{avg}$) | $w_{avg}$ ($t_{avg}$) | $w_{avg}$ ($t_{avg}$) | $w_{avg}$ ($t_{avg}$) |
| | *optimized* | 372(<0.001) | 12283 (3.974) | 34262.59 (1490.467) | 111342.37 (1639.896) |
| *init_construction* | 0 | 372(<0.001) | 12283 (3.826) | 34262.54 (1398.266) | 111340.81 (1718.638) |
| | 2 | 372(<0.001) | 12283 (4.496) | 34262.49 (1397.768) | 111341.27 (1913.193) |
| *perform_randomwalk* | *False* | 372(<0.001) | 12280.39 (0.131) | 34238.86 (0.273) | 111271.43 (2.704) |
| *perform_BMS* | *True* | 372(<0.001) | 12283 (67.657) | 34255.23 (1007.651) | 111213.88 (1784.614) |
| *breaking_ties* | 0 | 372(<0.001) | 12283 (4.495) | 34256.33 (545.964) | 111331.98 (1815.077) |
| *tabu_type* | 1 | 372(<0.001) | 12249.29 (180.579) | 34172.07 (32.233) | 111068.53 (2.103) |
| | 2 | 372(<0.001) | 12253.34 (325.536) | 34183 (1440.989) | 111115.07 (217.869) |
| *drop_vertex* | 0 | 372(<0.001) | 12283 (3.546) | 34262.6 (1475.438) | 111341.72 (1735.512) |
| | 2 | 372(<0.001) | 12283 (5.009) | 34262.63 (1507.567) | 111341.92 (1740.722) |
| *perform_restart* | *True* | 372(<0.001) | 12283 (4.167) | 34262.55 (1648.249) | 111340.37 (1870.848) |

MANN family), KES, and REF benchmarks, we adopt *avgPAR10* (smaller is better) as the metric of performance. For DIMACS (MANN family) instances, since the main difference between the results of all algorithms is the solution-quality, we use $w_{avg}$ (larger is better) and $t_{avg}$ as the metrics.

*Experimental results and discussion.* The comparative results of PbO-MWC (with the optimized configuration) and the alternative algorithms on BHOSLIB, DIMACS (except MANN family), KES, and REF benchmarks are shown in Table 9. It is not surprising that some alternative algorithms perform slightly better than PbO-MWC (with the optimized configuration), because the configuration with the best performance on the training set with a cutoff time of 300 s is not necessarily the same as the configuration with the best performance on the test set with a cutoff time of 3600 s. Table 9 clearly shows that:

- On the BHOSLIB benchmark: the random walk strategy and the tabu mechanism greatly impact the performance of the algorithm. The *avgPAR10* of the alternative algorithm (*perform_randomwalk* = *False*) is around 70 times larger than that of the baseline algorithm. The tabu-based prohibition strategy (*tabu_type* = 1) is the most suitable prohibition mechanism for this benchmark.
- On the DIMACS (except MANN family) benchmark: the restart strategy, the initial construction strategy, the tabu mechanism, and the random walk strategy play an important role. The *avgPAR10* of the alternative algorithm (*perform_restart* = *False*) is around 375 times larger than that of the baseline algorithm. The randomized initial construction approach (*init_construction* = 0) is suitable for this benchmark.
- On the KES benchmark: the BMS strategy greatly impacts the performance of the algorithm. The *avgPAR10* of the alternative algorithm (*perform_BMS* = *False*) is around 3495 times larger than that of the baseline algorithm.
- On the REF benchmark: the tabu mechanism, the BMS strategy, and the restart strategy play a crucial role. The *avgPAR10* of the alternative algorithm (SCC strategy, *i.e., tabu_type* = 0) is around 935 times larger than that of the baseline algorithm.

The comparative results of PbO-MWC (with the optimized configuration) and the alternative algorithms on DIMACS (MANN family) instances are shown in Table 10. Table 10 demonstrates that, on the MANN family instances of the DIMACS benchmark, the SCC strategy (*tabu_type* = 0) is the most suitable prohibition mechanism. The random walk strategy improves the performance of the algorithm, and the BMS strategy is ineffective in solving this group of instances.

## 6. Conclusions and future work

In this work, we proposed a parametric SLS framework for MVWCP, named PbO-MWC, which contains many effective techniques and restarts the local search process with a certain probability when it getting stuck local optima. We used the automated algorithm configuration procedure SMAC to configure PbO-MWC and its competitors. We conducted experiments to compare PbO-MWC with four state-of-the-art solvers on four benchmarks. The experimental results show that PbO-MWC achieves significant performance improvement on most MVWCP instances, especially on BHOSLIB and two real-world application benchmarks.

In the future, we plan to design novel strategies to integrate them into the PbO-MWC framework to further improve the performance. In addition, we would like to conduct experiments on more benchmarks that are transformed from real-world problems.

## CRediT authorship contribution statement

**Yi Chu:** Investigation, Methodology, Software, Formal analysis, Writing – original draft. **Chuan Luo:** Methodology, Validation, Writing – original draft. **Holger H. Hoos:** Conceptualization, Project administration, Resources, Supervision, Writing – review & editing. **Haihang You:** Funding acquisition, Writing – editing.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

The implementation of PbO-MWC, all the benchmarking instances adopted in this work and the complete comparative results are publicly available at https://github.com/filyouzicha/PbO-MWC.

## Acknowledgment

This work was partially supported by the National Key Research and Development Program of China under Grant 2017YFB0202502, and the National Natural Science Foundation of China under Grant 62202025 and Grant 41930110.

## References

Ansótegui, C., Sellmann, M., & Tierney, K. (2009). A gender-based genetic algorithm for the automatic configuration of algorithms. In *International conference on principles and practice of constraint programming* (pp. 142–157). Springer.

Balasundaram, B., & Butenko, S. (2006). *Graph domination, coloring and cliques in telecommunications* (pp. 865–890).

Ballard, D. H., & Brown, C. M. (1982). *Computer vision, 1982*. Englewood Cliffs, NJ: Prenice-Hall.

Battiti, R., & Protasi, M. (2001). Reactive local search for the maximum clique problem 1. *Algorithmica*, *29*(4), 610–637.

Birattari, M., Yuan, Z., Balaprakash, P., & Stützle, T. (2010). F-race and iterated F-race: An overview. In *Experimental methods for the analysis of optimization algorithms* (pp. 311–336). Springer.

Cai, S. (2015). Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In *Proceedings of IJCAI 2015* (pp. 747–753).

Cai, S., & Lin, J. (2016). Fast solving maximum weight clique problem in massive graphs. In *Proceedings of IJCAI 2016* (pp. 568–574).

Cai, S., Su, K., & Sattar, A. (2011). Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence*, *175*(9–10), 1672–1696.

Fan, Y., Li, N., Li, C., Ma, Z., Latecki, L. J., & Su, K. (2017). Restart and random walk in local search for maximum vertex weight cliques with evaluations in clustering aggregation. In *Proc. of international joint conference on artificial intelligence* (pp. 622–630).

Fang, Z., Li, C.-M., & Xu, K. (2016). An exact algorithm based on maxsat reasoning for the maximum weight clique problem. *Journal of Artificial Intelligence Research*, *55*, 799–833.

Glover, F. W. (1989). Tabu search - Part I. *INFORMS Journal on Computing*, *1*(3), 190–206.

Grosso, A., Locatelli, M., & Della Croce, F. (2004). Combining swaps and node weights in an adaptive greedy approach for the maximum clique problem. *Journal of Heuristics*, *10*(2), 135–152.

Hoos, H. H. (2012). Programming by optimization. *Communications of the ACM*, *55*(2), 70–80.

Hoos, H. H., & Stützle, T. (2004). *Stochastic local search: Foundations & applications*. Elsevier / Morgan Kaufmann.

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2010). Automated configuration of mixed integer programming solvers. In *International conference on integration of artificial intelligence (AI) and operations research (OR) techniques in constraint programming* (pp. 186–202). Springer.

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *Proceedings of LION 2011* (pp. 507–523).

Hutter, F., Hoos, H. H., Leyton-Brown, K., & Stützle, T. (2009). ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, *36*(1), 267–306.

Hutter, F., Lindauer, M., Balint, A., Bayless, S., Hoos, H., & Leyton-Brown, K. (2017). The configurable SAT solver challenge (CSSC). *Artificial Intelligence*, *243*, 1–25.

Jiang, H., Li, C.-M., Liu, Y., & Manyà, F. (2018). A two-stage maxsat reasoning approach for the maximum weight clique problem. In *Thirty-second AAAI conference on artificial intelligence*.

Jiang, H., Li, C.-M., & Manyà, F. (2017). An exact algorithm for the maximum weight clique problem in large graphs. In *AAAI* (pp. 830–838).

Johnson, D. S., & Trick, M. A. (Eds.), (1996a). Cliques, coloring, and satisfiability. In *DIMACS series in discrete mathematics and theoretical computer science*: *vol. 26*, DIMACS/AMS.

Johnson, D. S., & Trick, M. A. (1996b). *Cliques, coloring, and satisfiability: Second DIMACS implementation challenge, October 11-13, 1993, vol. 26*. American Mathematical Soc..

Karp, R. M. (1972). Reducibility among combinatorial problems. *Journal of Symbolic Logic*, *40*(4), 618–619.

KhudaBukhsh, A. R., Xu, L., Hoos, H. H., & Leyton-Brown, K. (2016). SATenstein: Automatically building local search SAT solvers from components. *Artificial Intelligence*, *232*, 20–42.

Li, C.-M., Jiang, H., & Manyà, F. (2017). On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. *Computers & Operations Research*, *84*, 1–15.

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

Luo, C., H.Hoos, H., Cai, S., Lin, Q., Zhang, H., & Zhang, D. (2019). Local search with efficient automatic configuration for minimum vertex cover. In *Proceedings of IJCAI 2019*.

Luo, C., Hoos, H. H., & Cai, S. (2020). PbO-CCSAT: Boosting local search for satisfiability using programming by optimisation. In *Proceedings of PPSN 2020* (pp. 373–389).

McCreesh, C., Prosser, P., Simpson, K., & Trimble, J. (2017). On maximum weight clique algorithms, and how they are evaluated. In *International conference on principles and practice of constraint programming* (pp. 206–225). Springer.

Östergård, P. R. (2001). A new algorithm for the maximum-weight clique problem. *Nordic Journal of Computing*, *8*(4), 424–436.

Park, K., Lee, K., & Park, S. (1996). An extended formulation approach to the edge-weighted maximal clique problem. *European Journal of Operational Research*, *95*(3), 671–682.

Pullan, W. (2008). Approximating the maximum vertex/edge weighted clique using local search. *Journal of Heuristics*, *14*(2), 117–134.

Pullan, W., & Hoos, H. H. (2006). Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, *25*, 159–185.

Pullan, W., Mascia, F., & Brunato, M. (2011). Cooperating local search for the maximum clique problem. *Journal of Heuristics*, *17*(2), 181–199.

Richter, S., Helmert, M., & Gretton, C. (2007). A stochastic local search approach to vertex cover. In *Annual conference on artificial intelligence* (pp. 412–426). Springer.

Vallati, M., Fawcett, C., Gerevini, A. E., Hoos, H., & Saetti, A. (2013). Automatic generation of efficient domain-optimized planners from generic parametrized planners. In *Sixth annual symposium on combinatorial search*.

Wang, Y., Cai, S., & Yin, M. (2016). Two efficient local search algorithms for maximum weight clique problem. In *Proceedings of AAAI 2016* (pp. 805–811).

Wu, Q., Hao, J.-K., & Glover, F. (2012). Multi-neighborhood tabu search for the maximum weight clique problem. *Annals of Operations Research*, *196*(1), 611–634.

Xu, K., Boussemart, F., Hemery, F., & Lecoutre, C. (2005). A simple model to generate hard satisfiable instances. In *Proceedings of IJCAI 2005* (pp. 337–342).

Xu, K., Boussemart, F., Hemery, F., & Lecoutre, C. (2007). Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artificial Intelligence*, *171*(8–9), 514–534.

Xu, L., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In *RCRA workshop on experimental evaluation of algorithms for solving problems with combinatorial explosion at the international joint conference on artificial intelligence* (pp. 16–30).

Yamaguchi, K., & Masuda, S. (2008). A new exact algorithm for the maximum weight clique problem. In *ITC-CSCC: international technical conference on circuits systems, computers and communications* (pp. 317–320).

Zhou, Y., Hao, J.-K., & Goëffon, A. (2017). PUSh: A generalized operator for the maximum vertex weight clique problem. *European Journal of Operational Research*, *257*(1), 41–54.