Improving the Performance of Stochastic Local Search for Maximum Vertex Weight Clique Problem Using Programming by Optimization

Yi Chu\textsuperscript{a,b}, Chuan Luo\textsuperscript{c}, Holger H. Hoos\textsuperscript{d}, Qingwei Lin\textsuperscript{e}, Haihang You\textsuperscript{a,∗}

\textsuperscript{a}State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China  
\textsuperscript{b}University of Chinese Academy of Sciences, Beijing 100049, China  
\textsuperscript{c}Microsoft Research, China  
\textsuperscript{d}Leiden Institute of Advanced Computer Science, Leiden University, Leiden, The Netherlands

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 which 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.

Keywords: Maximum Clique Problem, Stochastic Local Search, Programming by Optimization

1. Introduction

Given an undirected graph $G$, a clique is a subset of vertices of $C \subseteq G$ 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 \cite{1}. 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 maximum weight, i.e., a clique $C$ with maximum total weight over the vertices contained in $C$. The MVWCP has a wide range of practical applications, including computer vision, pattern recognition, robotics \cite{2}, broadband network design \cite{3} and wireless telecommunication \cite{4}.

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., \cite{5, 6, 7, 8, 9}). 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 reasonable time (see, e.g., \cite{10, 11, 12, 13, 14, 15, 16, 17, 18, 19}).

Among these incomplete algorithms, MN/TS \cite{15} shows an advantage on the well-known BHOSLIB benchmark, where instances are generated in the phase-transition area according to Model RB \cite{20} and known to be difficult in theory and practice \cite{21}, while the strong configuration checking (SCC) strategy underlying the LSCC algorithm \cite{16} exhibits stronger performance on the prominent DIMACS benchmark \cite{22}, 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 (PhO) \cite{23} 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 PhO, major improvements have been achieved in the state of the art for solving a broad range of NP-hard problems, including propositional satisfiability \cite{24}, mixed integer programming \cite{25} and minimum vertex cover \cite{26}.

In this work, we present what we believe to be the first application of the PhO 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 PhO-based local search framework for MVWCP dubbed PhO-MWC, which is highly configurable and incorporates a broad range of effective techniques.

- We conduct extensive experiments to compare PhO-MWC against five state-of-the-art solvers on four benchmarks, including two benchmarks that have been widely used in the literature (BHOSLIB and DIMAC), and two benchmarks derived from practical applications (kidney exchange and research excellence assessment). Our empirical results indicate that PhO-MWC outperforms its competitors on all four benchmarks. PhO-MWC significantly improves over the performance of state-of-the-art solvers for solving MVWCP on three benchmarks (BHOSLIB, kidney exchange and research excellence assessment). Notably, using PhO-MWC, we were able to achieve an improvement in the best known solution for a challenging graph known as ‘MANNa81’.

- 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 PhO-MWC solver framework and provide detailed descriptions of all its core components. In Section 5, we conduct extensive experiments to show the effectiveness of PhO-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 $w$ 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 $w(C) := \Sigma_{v \in C} w(v)$. In the maximum vertex weight clique problem (MVWCP), the objective is to find a clique $C \subseteq V$ with maximum weight $w(C)$ in a given vertex-weighted graph $G$.

As usual, edges $e \in E$ in a given undirected graph $G$ are represented as pairs of vertices $(u,v)$, where $u,v \in C$ are called the end points of $e = (u,v)$. Finally, for a vertex $v \in V$, we use $N(v) := \{ u \in V \mid (u,v) \in E \}$ to denote the set of neighbours of $v$, and $d(v) := |N(v)|$ to denote the degree of $v$.

Stochastic local search (SLS) algorithms for the MVWCP usually build and subsequently modify cliques until some termination criterion is met, and then return the clique with the maximum weight encountered during the search process. Three kinds of search steps are performed: add, swap and drop. To formally define these, we introduce three sets of vertices for a given clique $C$:

- $V_{add}(C) = \{ u \not\in C \mid \exists v \in C : u \in N(v) \}$, i.e., the set of vertices that, when added to $C$, result in a larger clique;
- $V_{swap}(C) = \{ u \not\in C \mid \exists v \in C : (u,v) \notin E \land \forall w \in C \setminus \{v\} : (u,w) \in E \}$, i.e., the set of vertices that can replace some 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.

An add step moves from a clique $C$ to a clique $C \cup V_{add}(C)$; a swap step moves from a clique $C$ to a clique $C \cup V_{swap}(C)$; and a drop step moves from a clique $C$ to a clique $C \cup V_{drop}(C)$. Cliques $C$ and $C'$ are called neighbours if $C'$ can be obtained from $C$ by a single add, swap or drop step.



### Algorithm 1: SLS for MVWCP

**Input:** graph $G=(V,E,W)$, the cutoff time;

**Output:** $C^*$;

1. initialize $C^* := \emptyset$, $C := \emptyset$;
2. while no termination criteria are met do
   3. while $V_{add}(C) \neq \emptyset$ do
      4. $v := \text{select a vertex from } V_{add}(C)$;
      5. $C := C \cup \{v\}$;
      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. $C := \text{a neighboring clique of } C$;
         10. if $W(C) > W(C')$ then
             11. $C^* := C$;
   12. return $C^*$;


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

High-performance SLS algorithms usually balance between two types of strategies: intensification 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 the neighbouring clique 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 (here: weight of the current clique).

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

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 \( \text{age}(v) \).

### 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 PbO-MWC.

#### 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) [10], dynamic local search (DLS-MC) [12] and cooperating local search (CLS) [14] 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 moves: \( \text{Add} \) and \( \text{Drop} \). As soon as a vertex is added or dropped, it is put into the tabu list and remains prohibited for the next \( T \) iterations. The prohibition \( T \) is adjusted through feedback from the previous history of the search. RLS performs better than its predecessors on DIMACS benchmark. DLS-MC alternates between clique expansion phase and plateau search phase. The expansion phase selects a vertex from \( V_{\text{add}}(C) \) to add it to the current clique. The plateau search phase selects a vertex pair from \( V_{\text{swap}}(C) \) to perform the \( \text{Swap} \) move.

The selection of vertices is based on vertex penalties that are dynamically adjusted during the search process. DLS-MC shows excellent performance on DIMACS benchmark. CLS also alternates between clique expansion phase and plateau search phase. CLS integrates four low level heuristics which 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 are comparable to state-of-the-art algorithms on DIMACS.

Multi-neighborhood tabu search (MN/TS) [15] is designed for MCP and MVWCP. Local search with SCC (LSCC) [16] and Restart and random walk in local search (RRWL) [19] 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 \( \text{Add} \), \( \text{Drop} \) and \( \text{Swap} \) moves. The general prohibition rule in these algorithms is only prohibit removed vertices to move back to the clique \( C \) during the prohibition period, vertices that 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 are 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 \( \text{Add} \) move. LSCC shows obvious advantage on DIMACS benchmark when comparing with predecessors. RRWL [19] proposes a revisiting based restart strategy and adopts random walk strategy. In the local search phase, RRWL uses the SCC strategy. RRWL utilizes a hash table to record cliques that have been visited and detects revisiting after the first step which increases the clique weight. In order to increase diversity, RRWL adopts random walk strategy when choosing a vertex to perform \( \text{Drop} \) move. RRWL finds a new best-known solution (111,341) on ‘MANN_a81\’ instance from DIMACS benchmark.

These algorithms do not contain various techniques and no single algorithm can perform well on all types of benchmarks. In addition, the performance of the above algorithms for solving MVWCP on many hard instances from BHOSLIB benchmark and DIMACS benchmark still exists the enhanced space. 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 [23]. Algorithm development following 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 [27] [28] [29]. Up to now, PbO-based approaches have shown effectiveness in...
many problems, including Boolean satisfiability \cite{24,30}, mixed integer programming \cite{25,31}, AI planning \cite{32} and Minimum Vertex Cover \cite{26}.

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 \cite{28}. Since it may be difficult to seek for 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 \cite{28}, GGA \cite{27}, F-RACE \cite{33}, SMAC \cite{29} and irace \cite{34}. 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 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 components until the termination criteria are met: initially, PbO-MWC generates an initialized clique; then PbO-MWC iteratively moves the current clique by local search step.

PbO-MWC consists of two components: construction and search. In the construction component, the initial clique is generated and regarded as the starting point of the search component. In the search component, PbO-MWC iteratively performs local search to move the clique to another. The pseudo-code of PbO-MWC is outlined in Algorithm 2. There is an outer loop (lines 2-18 in algorithm 2) and an inner loop (lines 6-18 in algorithm 2). This framework determines whether the algorithm jumps from the inner loop to the outer loop through a parameter called perform\_restart, thereby restarting the local search process by reconstructing the initial clique. If the weight of current clique do not increase after performing intensification\_process() and perform\_restart= True, the algorithm restarts the search process with a probability of restart\_prob (lines 13-16 in algorithm 2).

4.2. The Construction Component

To make 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):

Algorithm 2: The PbO-MWC Framework

| Input: | graph G=(V, E, W), the cutoff time; |
| Output: | C'; |
| 1 | initialize C’ := \emptyset; |
| 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’; |

i) Randomized approach: starting with an empty vertex set C, repeats adding a vertex \( v \in V_{add}(C) \) into C until \( V_{add}(C) = \emptyset \).

ii) Weight-based approach: randomly selecting a vertex \( v \) as the initialized clique \( C \), repeats adding a vertex \( v \in V_{add}(C) \) with the largest weight into C until \( V_{add}(C) = \emptyset \).

iii) Degree-based approach: randomly selecting a vertex \( v \) as the initialized clique \( C \), repeats adding a vertex \( v \in V_{add}(C) \) with the largest degree into C until \( V_{add}(C) = \emptyset \).

4.3. The Random Walk Component

PbO-MWC utilizes a parameter called perform\_randomwalk to determine whether to attempt 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 parameter. This procedure is outlined in Algorithm 2.

4.4. The Intensification Component

The Intensification procedure is outlined in algorithm 2. The intensification\_process moves the current clique to a neighboring clique with the maximum weight with prohibition mechanism. It first selects a vertex \( v \in V_{add}(C) \) with the largest Ascore and \( v \) is not forbidden (Line 1), and selects a vertex pair \( ⟨u, u'⟩ \in V_{swap}(C) \) with the largest Sscore and \( u \) is not forbidden (Line 2-5). If an Add move is possible, it selects the move 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 move if there is no Swap move or Dscore>Sscore (Line 13-15), otherwise it picks the Swap move (Line 16-17).

4
In intensification search, local search algorithms correspond to efforts of revisiting promising regions of the search space [35]. 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 [36] and configuration checking (CC) [37]. Tabu mechanism was proposed by Glover [36], and the tabu mechanism forbids reversing the recent changes, where the “strength” of prohibition is controlled by a parameter called tabu tenure(t). Configuration Checking (CC) was proposed by Cai [37]. 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. A brief description is as follows:

The MN/TS algorithm proposes a prohibition rule: a vertex that leaves the current clique C (by a Swap or Drop move) is forbidden to move back to C for the next \( n \) iterations. A vertex in the clique C is free to be removed from C without restriction [15].

For the Swap move, a vertex pair \( \langle u, v \rangle \in V_{swap}(C) \), where v is removed from C, v is prohibited to be moved back to C for the next \( n_{swap} \) iterations.

\[ nt_{swap} = \text{random}(V_{swap}(C)) + T_1 \]

For the Drop move, a vertex \( u \in V_{drop}(C) \), where u is removed from C, u is prohibited to be moved back to C for the next \( n_{drop} \) iterations.

\[ nt_{drop} = T_1 \]

\( T_1 \) is set to 7 as in [15].

The LSCC algorithm proposes a prohibition rule called strong configuration checking (SCC) for MVWCP [16]. 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 \( \langle u, v \rangle \) is swapped, where v is removed from C, \( confChange[v] = 0 \).

In this intensification component, we propose a new prohibition mechanism named TabuCC, which is inspired by tabu strategy and SCC strategy. The aim of MVWCP is to find a clique with the maximum weight, therefore, intuitively, if a vertex v is added to the current clique, then its neighbors should also be encouraged to add to the clique [16]. Based on this idea, we propose a tabu-based prohibition mechanism. The TabuCC mechanism is worked as follows:

i) For the Swap move, a vertex pair \( \langle u, v \rangle \in V_{swap}(C) \), where v is removed from C, v is prohibited to be moved back to C for the next \( nt_{swap} \) iterations.

\[ nt_{swap} = \text{random}(V_{swap}(C)) + T_1 \]

ii) For the Drop move, a vertex \( u \in V_{drop}(C) \), where u is removed from C, u is prohibited to be moved back to C for the next \( nt_{drop} \) iterations.

\[ nt_{drop} = T_1 \]

iii) For the Add move, a vertex v \( \in V_{add}(C) \), where v is added into C, for each vertex u \( \in N(v) \), lift the prohibition of u.

The intensification component provides three prohibition mechanisms for selection, including the tabu mechanism pro-
posed in MN/TS, SCC proposed in LSCC and TabuCC mentioned above. The tabu mechanism and TabuCC involve a parameter called 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 [38]. The BMS strategy randomly selects $bms\_num$ elements ($bms\_num$ is an integer parameter) from source set $S$, and then returns the best element. The activation of BMS strategy is depended on a Boolean-valued parameter perform_BMS. If 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 breakingTies-Rules: i) Breaking ties randomly; ii) Breaking ties in favor of the largest age.

4.5. Configuration space and default configuration

PhO-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 Subsections 4.1–4.4. In Table 1, we give an overview of the full configuration space of PhO-MWC, including all strategies and parameters, as well as the conditions under which strategies and parameters are activated. The default configuration settings of PhO-MWC are shown in Table 2.

5. Experimental Evaluations

To evaluate the efficiency of our proposed PhO-MWC framework and explore the potential of the configuration space, we conduct extensive experiments to compare PhO-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 describe the configuration protocols used to automatically configure PhO-MWC and its competitors. Then we describe the experimental setup. Finally, we present the experimental results and give some speculations about which strategies work well on which 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 [39]. The BHOSLIB and DIMACS benchmarks have been widely used in the recent literature to test new MCP and MVWCP solvers [40, 13, 14, 15, 16, 7, 41, 19]. The original graphs are unweighted, we adopt the method described in [13]; for each vertex $i$, $W_i$ is set to $(i \mod 200) + 1$.

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 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

---

**Table 1:** The configuration space of PhO-MWC.

| Parameters             | Depended Conditions | Parameter Type | Value Domain | Default Value |
|------------------------|---------------------|----------------|--------------|---------------|
| perform_BMS            | -                   | Categorical    | [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, 0.001] | 0.000001    |
| perform_randomwalk     | -                   | Boolean-valued | [True, False]| True          |
| randomwalk_prob        | perform_randomwalk = 1 | Real           | [0.0001, 0.1] | 0.0001       |
| tabu_type              | -                   | Categorical    | [0, 1, 2]    | 1             |
| tabu_tenure            | tabu_tenure = 1, 2  | Integer        | [1, 100]     | 7             |

**Table 2:** The default configuration of PhO-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 |
of exchange scores. McCreesh et al. [42] proposed that this optimization problem may be solved by reduction to MVWCP, where each vertex is an exchange, 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 the system for assessing the quality of research in higher education institution. 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 an exchange, whose weight is its score. The clique stands for the set of publications that an 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. These two benchmarks are generated by McCreesh et al. [42]. In this paper, we selected 42 instances the KES benchmark and 29 instances from the REF benchmark that are difficult to solve.

5.2. Competitors

In this paper, we compare PbO-MWC (the implementation is available online[1]) against five state-of-the-art solvers, including four SLS solvers and one complete solver:

**SLS solvers:**
- MN/TS [15] 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[2].
- LSCC [16] is an efficient SLS solver which perform well on BHOSLIB and DIMACS. LSCC+BMS [16] is suitable for massive graph instances. We used the version of these two solvers that are available online[3].
- RRWL [19] is an efficient SLS solver without parameters. We used the version of RRWL made available by its authors[3].

**Complete solver:**
- TSM-MWC [9] is a state-of-the-art complete solver for MWC in both small/medium and massive real-world graphs. The source code is available online[4].

### 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=0, 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=True, randomdrop\_prob=0.4, restart\_prob=9.44211698679448E-6, tabu\_type=2, tabu\_tenure=8 |

5.3. Configuration Protocol

In this work, we made use of SMAC (version:2.10.03) to automatically configure our PbO-MWC framework. In this subsection, we describe the protocol used for PbO-MWC and its competitors. We extracted a training set for each benchmark. For BHOSLIB, frb45 family is chosen to be the training set. For DIMACS and REF, we randomly chosen an instance from each family. For KES, we randomly chosen 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 New\_SQ: New\_SQ=0-(solution quality) + (solution time)/1000. We used SMAC to minimize New\_SQ. Throughout the configuration process, we allowed a 2-day time budget and a cutoff time

---

1. [https://github.com/PbO-MWC/PbO-MWC](https://github.com/PbO-MWC/PbO-MWC)
2. [http://www.info.univ-angers.fr/%7ehtao/clique.html](http://www.info.univ-angers.fr/%7ehtao/clique.html)
3. [http://ai.nenu.edu.cn/wangyy/Yiyuanda/Download/mwcp.tar.gz](http://ai.nenu.edu.cn/wangyy/Yiyuanda/Download/mwcp.tar.gz)
4. [https://github.com/Fan-Yi/Restart-and-Random-Walk-in-Local-Search-for-MVWC-in-Large-Sparse-Graphs](https://github.com/Fan-Yi/Restart-and-Random-Walk-in-Local-Search-for-MVWC-in-Large-Sparse-Graphs)
5. [https://home.mis.u-picardie.fr/%7ecli/EnglishPage.html](https://home.mis.u-picardie.fr/%7ecli/EnglishPage.html)
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: 7.6.18110), with Intel(R) Xeon(R) CPU E5-2620 2.10GHz CPU, 20MB L3 cache and 128GB RAM. Except RRWL and TSM-MWC without parameters, other solvers were configured using SMAC with the same configuration protocol. Each local search solver was executed 100 runs on each instance with seeds from 1 to 100. TSM-MWC was performed 1 run on each instance. The cutoff time for each solver run was set to 3600 seconds. 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 solSuc, and the averaged running time of finding the final solution on each instance, denoted by time. We denote the number of successful runs divided by the number of total runs as success rate. Since the main difference between solvers on DIMACS is the solution quality, for each solver on each instance of DIMACS, on the 100 runs, we report the maximum weight \(w_{\text{max}}\) and averaged weight \(w_{\text{avg}}\) of the cliques found by each solver. For TSM-MWC, we report the weight of final clique found, denoted by \(w_{\text{sol}}\), and the time of the final clique found, denoted by time. We do not report the instances that all the local search solvers reach 100% success rate with \(t_{\text{avg}}<10\) seconds. In this experiments, unspecified time units are CPU seconds.

We also report the averaged PAR10 (penalized running time, if the solver can not get the solBest in a given cutoff time, it counts the running time as 10 times the given cutoff time.) of each solver on each benchmark, denoted by avgPAR10.

### 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 outperforms the best solver on the training set and test set. On training set, PbO-MWC achieves 100% success rate on all 5 instances. MN/TS, the second best solver also achieves a 100% success rate on all training instances, but in terms of running time, the averaged time of PbO-MWC is 4.464 and the averaged time of MN/TS is 32.054. The success rate of LSCC, LSCC+BMS and RRWL are 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% success rate for 27 of them, while the figure is 15, 8, 8, 7 and 1 for MN/TS, LSCC,
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 | MN/1S | LSCC | LSCC+BMS | RRWL | TSM-MWC |
|----------------|---------|-------|------|----------|------|---------|
|                | $w_{max}(w_{avg})$ | $t_{avg}$ | $w_{max}(w_{avg})$ | $t_{avg}$ | $w_{max}(w_{avg})$ | $t_{avg}$ | $w_{max}(w_{avg})$ | $t_{avg}$ | $w_{sol}$ | $time$ |
| MANN_a45       | 34203 | 34220 | 34220 | 34258 | 34258 | 34263 | 34263 |
| brock800_4     | 2971 | 2971 | 2971 | 2971 | 2971 | 2971 | 2971 | 2971 | 34263 |
| C2000_9        | 10999 | 10999 | 10999 | 10999 | 10999 | 10999 | 10999 | 10999 | 34263 |
| c-fat500-10    | 0.248 | 0.059 | <0.001 | 0.379 | 0.190 | 0.190 | 0.190 | 0.190 | 34263 |
| DSJC1000.5     | 2186 | 2186 | 2186 | 2186 | 2186 | 2186 | 2186 | 2186 | 34263 |
| gen400_p0.9_75 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 34263 |
| hamming10-2    | 0.145 | 0.652 | 0.516 | 0.466 | 0.230 | 0.466 | 0.466 | 0.466 | 34263 |
| johnson32-2-4  | 0.003 | 0.003 | 0.003 | 0.003 | 0.003 | 0.003 | 0.003 | 0.003 | 34263 |
| p_hat1500-3    | 2.855 | 29.639 | 113.621 | 117.621 | 117.621 | 117.621 | 117.621 | 117.621 | 34263 |
| san400_0.9_1   | 1.402 | 1.402 | 1.402 | 1.402 | 1.402 | 1.402 | 1.402 | 1.402 | 34263 |
| MANN_a27       | 12283 | 12283 | 12283 | 12283 | 12283 | 12283 | 12283 | 12283 | 34263 |
| MANN_a81       | 3.974 | 137.077 | 129.249 | 251.788 | 270.134 | 4.400 |
| brock400_4     | 3626 | 3626 | 3626 | 3626 | 3626 | 3626 | 3626 | 3626 | 34263 |
| C1000.9        | 9254 | 9254 | 9254 | 9254 | 9254 | 9254 | 9254 | 9254 | 34263 |
| C4000.5        | 12792 | 12792 | 12792 | 12792 | 12792 | 12792 | 12792 | 12792 | 34263 |
| hamming10-4    | 5129 | 5129 | 5129 | 5129 | 5129 | 5129 | 5129 | 5129 | 34263 |
| keller5        | 3317 | 3317 | 3317 | 3317 | 3317 | 3317 | 3317 | 3317 | 34263 |
| keller6        | 8062 | 8062 | 8062 | 8062 | 8062 | 8062 | 8062 | 8062 | 34263 |
| san1000        | 1716 | 1716 | 1716 | 1716 | 1716 | 1716 | 1716 | 1716 | 34263 |
| san400_0.7_1   | 3941 | 3941 | 3941 | 3941 | 3941 | 3941 | 3941 | 3941 | 34263 |
| san400_0.7_2   | 3110 | 3110 | 3110 | 3110 | 3110 | 3110 | 3110 | 3110 | 34263 |

LSCC+BMS, RRWL and TSM-MWC, respectively. On the four instances where PBO-MWC can’t reach the 100% success rate, the success rate 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 shortest averaged time on 28 of 31 instances. The averaged time of PBO-MWC 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 competitive results of PBO-MWC and its competitors on DIMACS benchmark. From Table 5 our PBO-MWC algorithm performs much better than its competitors. On training set, PBO-MWC is the only solver achieves a 100% success rate with the shortest $t_{avg}$ on all 10 instances. On all 27 test instances, PBO-MWC reaches a 100% success rate for all of them, while the figure is 10, 16, 12, 13 and 11 for MN/TS, LSCC, LSCC+BMS, RRWL and TSM-MWC, respectively.
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.

| Graph                  | solBest                      | PbO-MWC       | MNT/TS      | LSCC       | LSCC+BMS    | RRWL       | TSM-MWC     |
|------------------------|------------------------------|---------------|-------------|------------|-------------|------------|-------------|
|                        |                              | #Suc(\text{avg}) | #Suc(\text{avg}) | #Suc(\text{avg}) | #Suc(\text{avg}) | #Suc(\text{avg}) | #Suc(\text{avg}) |
| 83                     | 123768868686868                 | 100 (0.895)   | 100 (76.022) | 100 (75.769) | 100 (32.062) | 100 (142.617) | 100 (162.990) |
| 84                     | 11001661019237                | 100 (0.593)   | 100 (58.822) | 100 (52.100) | 100 (56.980) | 100 (176.084) | 100 (173.049) |
| 91                     | 13064410904076                | 100 (2.147)   | 100 (272.249) | 74 (136.477) | 72 (103.199) | 4 (177.404)  | 1 (119.590)  |
| 96                     | 1375094525251                 | 100 (1.572)   | 100 (271.969) | 85 (959.731) | 100 (440.349) | 0 (6.290)   |
| 97                     | 1375144632330                | 100 (2.045)   | 100 (241.925) | 82 (119.963) | 73 (1026.612) | 0 (744.600) |             |
| 105                    | 1787972026069                | 100 (7.879)   | 92 (1073.198) | 74 (136.477) | 72 (103.199) | 4 (177.404)  | 1 (119.590)  |
| 107                    | 1650240569546                | 100 (4.792)   | 95 (1011.776) | 78 (196.789) | 52 (1548.089) | 41 (1321.153) | 0 (148.390)  |
| 114                    | 2406557630478                | 100 (17.364)  | 153 (350.685) | 2 (169.137) | 2 (1776.499) | 0 (1671.880) |
| 116                    | 2131511910416                | 100 (5.393)   | 138 (374.749) | 82 (1110.178) | 68 (1349.423) | 38 (1491.430) | 0 (414.780)  |
| 119                    | 2269082892629                | 100 (4.812)   | 3 (1220.001)  | 100 (679.383) | 99 (743.993)  | 75 (2817.739) | 0 (488.960)  |

- Relatively. The total #Suc of PbO-MWC is 2700, while the figure is 1273, 2117, 2010, 1884 and 1100 for its competitors respectively. In terms of running time, on 21 of the 27 instances, PbO-MWC can achieve solBest with \( \text{avg}\_t <= 10 \) seconds. The averaged time of PbO-MWC is 7.625, while the figure is 810.750, 637.970, 748.197, 797.203 and 736.626 for its competitors respectively.

5.5.4. Results on Research Excellence Framework (REF) Benchmark

The results shown in Table 2 indicate that PbO-MWC outperforms its competitors on REF benchmark. On training set, PbO-MWC is the only solver achieves a 100% success rate on all 3 instances. On ‘reft-60-23-0’ instance, PbO-MWC reaches #Suc=100 with \( \text{avg}\_t = 37.242 \), MN/TS, the second best solver on this instance, reaches #Suc=87 with \( \text{avg}\_t = 1045.383 \). On all 19 test instances, PbO-MWC reaches a 100% success rate 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 ‘reft-60-50-0’ instance, PbO-MWC is the only solver that can achieve a 100% success rate, and the \( \text{avg}\_t \) of PbO-MWC is much shorter than that of its competitors. In terms of running time, PbO-MWC achieves solBest with shortest \( \text{avg}\_t \) on 18 of 19 instances. The averaged time of PbO-MWC is 2.272, while the figure is 43.114, 305.898, 245.693, 595.366 and 1584.664 for its competitors respectively.

5.5.5. An Overview of Results on All Benchmarks

We summarize all the results in Table 8. Table 9 shows that PbO-MWC outperforms all its competitors in terms of avgPAR10 on all four benchmarks. On BHOSLIB, DIMACS, KES and REF, the ratio of avgPAR10 of the best performing competitor to avgPAR10 of PbO-MWC is 31.64, 1.53, 1093.38 and 275.29, respectively. The performance of state-of-the-art solver for MMVCP on BHOSLIB benchmark and two benchmarks that are transformed from real-world problems has 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 and PbO-MWC (Default) on the four benchmarks, as shown in Figure 1. Figure 1 clearly illustrates that configuration lead to performance improvements on a large majority of instances on all the four benchmarks.

5.5.7. The speculations on the effectiveness of different strategies

Based on the extensive experiments, we made some speculations. We presumed that random walk strategy plays an im-

10
import role on BHOSLIB. Among the three prohibition mechanisms utilized in PbO-MWC, SCC strategy performs best on DIMACS. In our experiments, all efficient configurations for KES contain BMS strategy with a parameter $bms_num$ that are less than 10. On REF, the optimized configuration includes TabuCC strategy, which works well on the REF benchmark. We consider an effective framework on REF benchmark has the following properties: restarting local search with a lower probability, including BMS strategy with a lower $bms_num$ and no random walk strategy is included.

6. Conclusions and Future work

In this work, we proposed a parametric SLS framework for MVWCP, called PbO-MWC, which contains many effective techniques that 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 the performance of all the configured solvers on four benchmarks. On BHOSLIB, KES and REF, the ratio of avg-PAR10 of the second best solver to avg-PAR10 of PbO-MWC is 31.64, 1093.38 and 275.29 respectively. On ‘MANNa81’ from DIMACS, a new optimal solution (111,355) is found by PbO-MWC.

In the future, we plan to do some research to understand which strategies are best suitable for particular problem types, and design novel strategies to integrate them into 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.

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.

| Graph  | solBest | PbO-MWC #Suc(tavg) | MIN/TS #Suc(tavg) | SCC #Suc(tavg) | SCC+BMS #Suc(tavg) | RRWL #Suc(tavg) | TS-MWC #Suc(tavg) | avgPAR10 | avgPAR10 | avgPAR10 |
|--------|---------|---------------------|------------------|--------------|------------------|----------------|------------------|-----------|-----------|-----------|
| ref-60-1000 | 743     | 100 (5.978)         | 100 (0.096)      | 100 (2.304)  | 100 (2.815)      | 100 (5.381)    | 100 (277.880)    |           |           |           |
| ref-60-230-0  | 506     | 100 (37.242)        | 87 (1045.383)    | 0 (778.114)  | 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 (14.161)        | 100 (0.032)      | 100 (0.097)  | 100 (0.118)      | 100 (0.527)    | 100 (112.890)    |           |           |           |

Table 8: The avgPAR10 on each benchmark.

| Benchmark | Num | PbO-MWC avgPAR10 | MIN/TS avgPAR10 | SCC avgPAR10 | SCC+BMS avgPAR10 | RRWL avgPAR10 | TS-MWC avgPAR10 |
|-----------|-----|-----------------|----------------|--------------|-----------------|---------------|-----------------|
| BHOSLIB   | 40  | 228.662         | 724.972        | 18707.356    | 19131.201       | 18743.424     | 32519.003       |
| DIMACS    | 80  | 903.982         | 1379.709       | 1885.051     | 1887.256        | 1770.183      | 5113.066        |
| KES       | 42  | 6.154           | 16728.550      | 6728.632     | 8526.514        | 10163.870     | 20723.301       |
| REF       | 29  | 3.170           | 872.675        | 8423.623     | 8000.062        | 10168.337     | 27520.371       |

References

[1] R. M. Karp, Reducibility among combinatorial problems, Journal of Symbolic Logic 40 (4) (1972) 618–619.
[2] D. H. Ballard, C. M. Brown, Computer vision (1982) 145.
[3] K. Park, K. Lee, S. Park, An extended formulation approach to the edge-weighted maximal clique problem, European Journal of Operational Research 95 (3) (1996) 671–682.
[4] B. Balasundaram, S. Butenko, Graph Domination, Coloring and Cliques in Telecommunications, 2006.
[5] P. R. Östergård, A new algorithm for the maximum-weight clique problem, Nordic Journal of Computing 8 (4) (2001) 424–436.
[6] K. Yamaguchi, S. Masuda, A new exact algorithm for the maximum weight clique problem, in: ITC-CSCC: International Technical Conference on Circuits Systems, Computers and Communications, 2008, pp. 317–320.
[7] Z. Fang, C.-M. Li, K. Xu, An exact algorithm based on maxsat reasoning for the maximum weight clique problem, Journal of Artificial Intelligence Research 55 (2016) 799–833.
[8] H. Jiang, C.-M. Li, F. Manyà, A two-stage maxsat reasoning approach for the maximum weight clique problem, in: Thirty-Second AAAI Conference on Artificial Intelligence, 2018.
[9] R. Battiti, M. Protasi, Reactive local search for the maximum clique problem, Algorithmica 29 (4) (2001) 610–637.
[10] A. Grosso, M. Locatelli, F. Della Croce, Combining swaps and node weights in an adaptive greedy approach for the maximum clique problem, Journal of Heuristics 10 (2) (2004) 135–152.
[12] W. Pullan, H. H. Hoos, Dynamic local search for the maximum clique problem, Journal of Artificial Intelligence Research 25 (2006) 159–185.

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

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

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

[16] Y. Wang, S. Cai, M. Yin, Two efficient local search algorithms for maximum weight clique problem, in: Proceedings of AAAI 2016, 2016, pp. 805–811.

[17] S. Cai, J. Lin, Fast solving maximum weight clique problem in massive graphs, in: Proceedings of IJCAI 2016, 2016, pp. 568–574.

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

[19] Y. Fan, N. Li, C. Li, Z. Ma, L. J. Latecki, K. Su, 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 (IJCAI), 2017, pp. 622–630.

[20] K. Xu, F. Boussenart, F. Hemery, C. Lecoutre, A simple model to generate hard satisfiable instances, in: Proceedings of IJCAI 2005, 2005, pp. 337–342.

[21] K. Xu, F. Boussenart, F. Hemery, C. Lecoutre, Random constraint satisfaction: Easy generation of hard (satisfiable) instances, Artificial Intelligence 171 (8-9) (2007) 514–534.

[22] D. S. Johnson, M. A. Trick (Eds.), Cliques, Coloring, and Satisfiability, Vol. 26 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, DIMACS/AMS, 1996.

[23] H. H. Hoos, Programming by optimization, Commun. ACM 55 (2) (2012) 70–80.

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

[25] F. Hutter, H. H. Hoos, K. Leyton-Brown, Automated configuration of mixed integer programming solvers, in: International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming, Springer, 2010, pp. 186–202.

[26] C. Luo, H. H.Hoos, S. Cai, Q. Lin, H. Zhang, D. Zheng, Local search with efficient automatic configuration for minimum vertex cover, in: Proceedings of IJCAI 2019, 2019.

[27] C. Ansótegui, M. Sellmann, K. Tierney, A gender-based genetic algorithm for the automatic configuration of algorithms, in: International Conference on Principles and Practice of Constraint Programming, Springer, 2009, pp. 142–157.

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

[29] F. Hutter, H. H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: Proceedings of LION 2011, 2011, pp. 507–523.

[30] F. Hutter, M. Lindauer, A. Balint, S. Bayless, H. Hoos, K. Leyton-Brown, The configurable sat solver challenge (cscs), Artificial Intelligence 243 (2017) 1–25.

[31] L. Xu, F. Hutter, H. H. Hoos, K. Leyton-Brown, 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 (IJCAI), 2011, pp. 16–30.

[32] M. Vallati, C. Fawcett, A. E. Gerevini, H. Hoos, A. Saetti, Automatic generation of efficient domain-optimized planners from generic parametrized planners, in: Sixth Annual Symposium on Combinatorial Search, 2013.

[33] M. Birattari, Z. Yuan, P. Balaprakash, T. Stützle, F-race and iterated f-race: An overview, in: Experimental methods for the analysis of optimization algorithms, Springer, 2010, pp. 311–336.

[34] M. Lopez-Ibanez, J. Dubois-Lacoste, L. P. Ceccé, M. Birattari, T. Stützle, The irace package: Iterated racing for automatic algorithm configuration,
[35] H. H. Hoos, T. Stützle, Stochastic Local Search: Foundations & Applications, Elsevier / Morgan Kaufmann, 2004.
[36] F. Glover, Tabu search?part i, ORSA Journal on computing 1 (3) (1989) 190–206.
[37] S. Cai, K. Su, A. Sattar, Local search with edge weighting and configuration checking heuristics for minimum vertex cover, Artificial Intelligence 175 (9-10) (2011) 1672–1696.
[38] S. Cai, Balance between complexity and quality: Local search for minimum vertex cover in massive graphs, in: Proceedings of IJCAI 2015, 2015, pp. 747–753.
[39] D. S. Johnson, M. A. Trick, Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993, Vol. 26, American Mathematical Soc., 1996.
[40] S. Richter, M. Helmert, C. Gretton, A stochastic local search approach to vertex cover, in: Annual Conference on Artificial Intelligence, Springer, 2007, pp. 412–426.
[41] C.-M. Li, H. Jiang, F. Manyà, On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem, Computers & Operations Research 84 (2017) 1–15.
[42] C. McCreesh, P. Prosser, K. Simpson, J. Trimble, On maximum weight clique algorithms, and how they are evaluated, in: International Conference on Principles and Practice of Constraint Programming, Springer, 2017, pp. 206–225.