Improving Local Search Algorithms via Probabilistic Configuration Checking

Weilin Luo[1], Rongzhen Ye[1], Hai Wan[1,2*], Shaowei Cai[3*], Biqing Fang[1], Delong Zhang[1]

[1] School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China
[2] Key Laboratory of Machine Intelligence and Advanced Computing (Sun Yat-sen University), Ministry of Education, China
[3] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
{luowlin3,yerzh,fangbq3,zhangdlong3}@mail2.sysu.edu.cn, wanhai@mail.sysu.edu.cn, caisw@ios.ac.cn

Abstract

Configuration checking (CC) has been confirmed to alleviate the cycling problem in local search for *combinatorial optimization problems* (COPs). When using CC heuristics in local search for graph problems, a critical concept is the configuration of the vertices. All existing CC variants employ either 1- or 2-level neighborhoods of a vertex as its configuration. Inspired by the idea that neighborhoods with different levels should have different contributions to solving COPs, we propose the *probabilistic configuration* (PC), which introduces probabilities for neighborhoods at different levels to consider the impact of neighborhoods of different levels on the CC strategy. Based on the concept of PC, we first propose *probabilistic configuration checking* (PCC), which can be developed in an automated and lightweight favor. We then apply PCC to two classic COPs which have been shown to achieve good results by using CC, and our preliminary results confirm that PCC improves the existing algorithms because PCC alleviates the cycling problem.

1 Introduction

Local search is a popular method for solving *combinatorial optimization problems* (COPs). One severe issue with local search is the *cycling problem*, *i.e.*, revisiting a candidate solution that has been visited recently (Michiels, Aarts, and Korst 2007). More generally, the cycling problem refers to the phenomenon that a local search algorithm spends too much time in searching a small part of the search space.

Configuration checking (CC) (Cai, Su, and Sattar 2011) is an effective mechanism to alleviate the cycling problem for COPs, particularly those graph theoretic ones. It is an interesting alternative to the standard use of tabu mechanism (Glover 1989, 1990) to alleviate the cycling problem in COPs. The main idea of the CC strategy is that if the configuration of a vertex remains unchanged since its last removal from the candidate solution, then it is forbidden to be added back into the candidate solution. In recent years, there have been variants of CC adapted to different COPs, such as *maximum weight clique problem* (MWCP) (Wang, Cai, and Yin 2016; Wang et al. 2020), *minimum weight dominating set problem* (MWDS) (Wang, Cai, and Yin 2017), and *maximum k-plex problem* (Chen et al. 2020). Although there are numerous variants of CC, the way of defining the configuration relies on manual design. The configuration based on the 1-level neighborhoods of a vertex (Definition 1) is widely used in CC-based local search algorithms. Another one is the 2-level neighborhoods definition introduced for MWDS.

We argue that neighborhoods with more than two levels can have an impact on a vertex and should be considered in the configuration. To confirm that, we conduct a motivation experiment to give empirical evidence (details in Section 4), where we consider the 1st-, 2nd- and 2^+-level (Definition 1) neighborhoods in the configuration for MWDS. We randomly choose half vertices at each level as its new configuration. The results show that the new configuration improves the average performance for 71% instances.

In this paper, we focus on a new variant of CC that can capture the contributions of neighborhoods with different levels to the CC strategy. Our work revolves around the following two questions. Firstly, is there a general way to define the configuration of a vertex? Then, how does the new configuration lead to a further improvement of the CC strategy?

To answer the above questions, we propose a *probabilistic configuration* (PC) which can be grounded into a specific configuration by training for a given COP. The highlights of PC are as follows. Firstly, it allows automatic design of configurations, as its general form provides a design space of every possible definition of configuration based on neighborhoods. Secondly, it can easily lead to *probabilistic configuration checking* (PCC) strategy by following the updating rules of CC strategy. Finally, by introducing probability, it enhances the ability to explore new candidate solutions.

We apply PCC to two CC-based local search algorithms (Wang, Cai, and Yin 2017, 2016) that achieve state-of-the-art performance for MWDS and MWCP respectively. Following the assessment of the methods (Wang, Cai, and Yin 2017, 2016), we conduct experiments to compare the improved algorithms using PCC with the original ones on a wide range of benchmarks, which shows the superiority of PCC over the original CC strategies. Besides, further analyses show that by replacing the original CC strategies with PCC, the number of different candidate solutions visited by the algorithms in fixed steps significantly increases. This indicates that PCC can better alleviate the cycling issue, which

*Corresponding authors.

accounts for the performance improvement.

2 Preliminaries

An undirected graph $G = (V, E)$ consists of a vertex set V and an edge set $E \subseteq V \times V$. For an edge $e = \{u, v\}$, vertices u and v are the endpoints of edge e. Two vertices are *neighboring vertices* if and only if they belong to one edge. For a vertex v, the set of its neighboring vertices is $N(v) = \{u \in V \mid \{u, v\} \in E\}$, and its *degree* is $deg(v) = |N(v)|$. The distance between two vertices u and v, denoted by $dist(u, v)$, is the number of edges in the shortest path between them.

Definition 1. *For a vertex v, we define its i-th-level neighborhoods as $N_i(v) = \{u \in V \mid dist(u, v) = i\}$, i-level neighborhoods as $N_{\leq i}(v) = \bigcup_{j=1}^{i} N_j(v)$, and its i^+-level neighborhoods as $N_{>i}(v) = \bigcup_{j=i+1} N_j(v)$, particularly, $N_1(v) = N_{\leq 1}(v) = N(v)$.*

Configuration Checking. Let $G = (V, E)$ be an undirected graph, $C \subset V$ the current candidate solution, and v a vertex. $s_v \in \{1, 0\}$ is the state of v, where $s_v = 1$ (*resp.* $s_v = 0$) means $v \in C$ (*resp.* $v \notin C$). The CC strategy is based on the concept of configuration. In previous variants of CC for COPs, the configuration of a vertex is a vector which consists of the states of all vertices in its 1-level or 2-level neighborhoods (Wang, Cai, and Yin 2017).

The CC strategy is usually implemented with a Boolean array *confCh* for vertices. During the search procedure, those vertices whose value in *confCh* is 0 are forbidden to add into C. A typical CC strategy is as follows. The *confCh* array is initialized as an all-1 array. After that, when a vertex v is removed from C, $confCh[v]$ is reset to 0, and when a vertex v changes its state, for each vertex u in v's configuration, $confCh[u]$ is set to 1.

3 Related Work

The variants of CC can be classified into three main categories: redefinition of configuration, multi-value CC, and changes of CC rules.

Redefinition of configuration. Many variants consider the 1-level neighborhoods as configuration, such as (Cai et al. 2013; Cai and Su 2013; Wang, Cai, and Yin 2016). Wang, Cai, and Yin (2017) proposed CC^2 for MWDS, which takes the 2-level neighborhoods into account, exploring more search space. The same configuration design appeared in CC^{2}V3 (Wang et al. 2018).

Multi-value CC. Luo et al. (2015) proposed CSCC for *Boolean satisfiability problem* (SAT), where the criterion of adding a vertex is an integer rather than a Boolean. Wang et al. (2018) proposed CC^{2}V3 for MWDS problem in massive graphs, which allows configuration to have three values. For maximum k-plex problem, Chen et al. (2020) proposed DCC combined with incremental neighborhoods updating to reduce configuration changing of the high-degree vertices.

Changes of CC rules. The third category of variants of CC is the largest one. Many CC rules are added according to analyzing different problems to obtain a better performance, such as NCCA+ (Abramé, Habet, and Toumi 2017) for SAT, SCC (Wang, Cai, and Yin 2016) for MWCP, HC-SCC (Luo et al. 2017) for weighted partial maximum satisfiability problem, and CCA (Cai and Su 2012; Li et al. 2018) for minimum weighted vertex cover problem, *etc*.

In this paper, we focus on the line of redefining the configuration. We propose the PC that can represent all existing configuration definitions of various CC heuristics based on neighborhoods. We further propose the PCC heuristic.

To evaluate the effectiveness of our method, we apply PCC to two CC-based algorithms for two important COPs: CC^{2}FS (Wang, Cai, and Yin 2017) for MWDS and LSCC (Wang, Cai, and Yin 2016) for MWCP. The outlines of these two algorithms are shown in Algorithm 1 and Algorithm 2. The reasons why we choose these algorithms will be discussed in Section 6. The brief descriptions of MWDS and MWCP are as follows.

MWDS. Given a vertex weighted graph, the minimum weight dominating set problem (MWDS) consists in identifying a subset of vertices $D \subseteq V$ with the smallest total weight such that all vertices are either in D or adjacent to at

Algorithm 1: CC^{2}FS(G, $cutoff$)

Input: a weighted graph $G = (V, E, W)$, the $cutoff$ time.
Output: a minimum dominating set of G.

```
S := InitGreedyConstruction() and S* := S;
while elapsed time < cutoff do
    if there are no uncovered vertices then
        if w(S) < w(S*) then S* := S;
        remove a vertex from S based on search strategy and update confCh according to CC^2-Rules;
        continue;
    end if
    remove a vertex from S based on search strategy and update confCh according to CC^2-Rules;
    while there are uncovered vertices do
        add a vertex with confCh == 1 to S based on search strategy and update confCh according to CC^2-Rules;
    end while
end while
return S*;
```

Algorithm 2: LSCC(G, $cutoff$)

Input: a weighted graph $G = (V, E, W)$, the $cutoff$ time, the search depth L.
Output: a maximum weight clique of G.

```
C* := ∅;
while elapsed time < cutoff do
    C := GreedyConstruction() and update C* with C
    for step = 0; step < L; step++ do
        pick a vertex v ∉ C with confCh[v] == 1;
        pick vertex pair (u, u') with confCh[u'] == 1;
        pick a vertex x ∈ C;
        add v to C or remove x from C or swap u from C with u' according to search strategy;
        update confCh according to SCC-Rules;
        if w(C) > w(C*) then C* := C;
    end for
end while
return C*;
```

least one vertex in D.

MWCP. The maximum clique problem is to find a clique with the most vertices. When each vertex is associated with a positive integer weight, MCP is extended to MWCP which asks for a clique with the maximum total weight.

4 A Motivating Example

A main motivation of this work is the observation that CC equipped with configurations including more levels (including 1st-, 2nd- and 2^+-levels) leads to improvements to the algorithms based on only 1st- and 2nd-level neighborhoods.

Instances	CC^{2}FS			CC^{2}FS-H		
	min	avg	time	min	avg	time (s)
frb30-15-1	212	214.2	2.59	212	**212.0**	**0.11**
frb30-15-2	242	242.0	8.81	242	242.0	**0.18**
frb30-15-3	175	175.0	**0.01**	175	175.0	0.03
frb30-15-4	166	168.6	0.19	166	**166.0**	**0.08**
frb30-15-5	160	160.0	**0.01**	160	160.0	0.02
frb35-17-1	274	274.9	0.57	274	**274.0**	**0.11**
frb35-17-2	208	208.3	67.44	208	**208.0**	**0.29**
frb35-17-3	201	201.0	0.38	201	201.0	**0.10**
frb35-17-4	286	286.8	0.73	286	**286.0**	**0.45**
frb35-17-5	295	295.7	5.60	295	**295.0**	**0.11**
frb40-19-1	262	262.0	1.99	262	262.0	**0.14**
frb40-19-2	243	243.9	0.66	243	**243.0**	**0.18**
frb40-19-3	250	251.8	102.43	250	**250.0**	**3.88**
frb40-19-4	250	250.0	0.07	**249**	**249.0**	95.82
frb40-19-5	272	282.3	0.31	272	**277.6**	82.50
frb45-21-1	328	328.3	7.82	328	**328.0**	**0.64**
frb45-21-2	259	261.7	46.82	259	**259.1**	**1.39**
frb45-21-3	233	233.9	9.19	233	**233.0**	**0.89**
frb45-21-4	399	399.2	22.67	399	**399.0**	**0.95**
frb45-21-5	312	318.8	128.19	312	**312.0**	**1.82**
frb50-23-1	261	264.7	90.71	261	**261.0**	**8.97**
frb50-23-2	277	277.0	**0.01**	277	277.0	0.17
frb50-23-3	299	301.9	1.21	**281**	**292.2**	31.13
frb50-23-4	265	265.0	3.04	265	265.0	**0.47**
frb50-23-5	410	418.8	128.09	410	**411.1**	158.96
frb53-24-1	229	230.0	95.41	229	**229.0**	**2.35**
frb53-24-2	298	298.0	1.04	298	298.0	**0.38**
frb53-24-3	182	182.1	0.01	182	**182.0**	0.26
frb53-24-4	189	189.0	**0.02**	189	189.0	0.27
frb53-24-5	204	204.0	**0.01**	204	204.0	0.29
frb56-25-1	229	229.2	0.32	229	**229.0**	0.41
frb56-25-2	319	319.5	25.27	319	**319.0**	**3.46**
frb56-25-3	336	338.7	30.21	336	**336.0**	38.80
frb56-25-4	268	268.0	**0.21**	268	268.0	0.41
frb56-25-5	425	427.6	103.13	425	**425.0**	**1.83**
frb59-26-1	262	264.0	1.62	262	**262.0**	25.31
frb59-26-2	383	391.1	97.04	383	**383.0**	**5.64**
frb59-26-3	248	248.0	1.13	**246**	**246.0**	14.24
frb59-26-4	248	248.9	47.47	248	**248.0**	**0.62**
frb59-26-5	288	288.8	195.62	288	**288.0**	**3.26**
frb100-40	350	350.0	**2.04**	350	350.0	9.70
#win	0	0	7	**3**	**29**	**25**

Table 1: Comparison results of CC^{2}FS and CC^{2}FS-H on BHOSLIB benchmark, where 'min' means the best solution found in all runs, 'avg' means the average value of the solutions found, 'time' means the average time to find the optimal solution, and '#win' means the number of instances where the solver is better. All algorithms are executed 10 times on each instance independently with a cutoff time of 1000 seconds. We use boldface to indicate the better results.

In the experiment, we consider neighborhoods of all levels. We run the MWDS algorithm CC^{2}FS, the first algorithm with 2-level (*i.e.*, $N_{\leq 2}(v)$) CC strategy. As in the paper of CC^{2}FS, the experiment is conducted with the BHOSLIB benchmark. Interestingly, a simple variant of the CC strategy leads to obvious improvements. In this variant, we randomly choose half neighborhoods of each level as the configuration of a vertex. The obtained algorithm by replacing the CC^2 with this half-neighborhood CC is denoted as CC^{2}FS-H.

The comparison results of CC^{2}FS and CC^{2}FS-H are shown in Table 1. The statistics show that CC^{2}FS-H finds better solutions than CC^{2}FS on 3 instances, and on the other instances they have the same solutions. CC^{2}FS-H also dominates CC^{2}FS on the average solution quality, where CC^{2}FS-H outperforms CC^{2}FS on 29 out of 41 instances. These results motivate us to reconsider the definition of configuration. Traditional definition based on only 1st- and 2nd-level neighborhoods is far from an optimal configuration design. This paper proposes a general form of configuration, which can represent all existing configurations based on neighborhoods and also can be designed in an automated way via training the algorithm on the benchmarks. More importantly, this leads to significant improvements for two state-of-the-art CC-based algorithms.

5 Probabilistic Configuration Checking

In this section, we first give the definition of PC. Then we show the PCC strategies for MWDS and MWCP. After that, we analyze the complexity. Finally, we show how to obtain the optimized PC for a COP.

5.1 Probabilistic Configuration

We first introduce *probabilistic configuration* (PC).

Definition 2. *Given an undirected graph $G = (V, E)$ and a candidate solution S, the probabilistic configuration (PC) of a vertex $v \in V$ is a vector consisting of the states of all vertices in $(\bigcup_{i=1}^{2}(N_i(v))^{p_i}) \cup (N_{>2}(v))^{p_3}$, where $p_i \in [0, 1]$, $i = 1, 2, 3$, and $\{*\}^{p_i}$ is randomly chosen vertices from the set $\{*\}$ with the size of $\lceil |\{*\}| \cdot p_i \rceil$.*

Intuitively, the PC generically captures the vertices including the remote and near ones that are supposed to be the configuration of a vertex. The PC contains a set of probabilities ($p_1, p_2, p_3 \in [0, 1]$) characterizing the contributions of vertices at different distances to improve results. Resort to an automatic configuration process (training), we will train the optimized configuration, *i.e.*, the set of probabilities, on the standard benchmarks for a specific COP. The details of this training process will be explained in Section 5.4.

Note that the existing configurations are special cases of the PC as shown in Table 2.

Configuration	Probabilistic configuration form
$N_1(v)$	$(N_1(v))^{1.0} \cup (N_2(v))^{0.0} \cup (N_{>2}(v))^{0.0}$
$N_1(v) \cup N_2(v)$	$\bigcup_{i=1}^{2}(N_i(v))^{1.0} \cup (N_{>2}(v))^{0.0}$

Table 2: Configurations in the form of PC.

5.2 Probabilistic Configuration Checking Strategy

Now, we introduce how to obtain a PC with a $SelectConf$ procedure, and how to obtain PCC strategy based on the PC.

Algorithm 3: SelectConf(v, $N_1(v)$, $N_2(v)$, $N_{>2}(v)$, P)

Input: a vertex v and $N_1(v)$, $N_2(v)$, $N_{>2}(v)$, $P = \{p_1, p_2, p_3\}$ a set of probabilities.
Output: the configuration $Conf$.

1: $Q_1 \leftarrow$ randomly choose $\lceil |N_1(v)| \cdot p_1 \rceil$ vertices from $N_1(v)$;
2: $Q_2 \leftarrow$ randomly choose $\lceil |N_2(v)| \cdot p_2 \rceil$ vertices from $N_2(v)$;
3: $Q_3 \leftarrow$ randomly choose $\lceil |N_{>2}(v)| \cdot p_3 \rceil$ vertices from $N_{>2}(v)$;
4: $Conf(v) \leftarrow Q_1 \cup Q_2 \cup Q_3$;
5: **return** $Conf(v)$;

Firstly, we carry out a preprocessing to compute the 1st-, 2nd-, and 2^+- level neighborhoods for each vertex, which can be quickly implemented with a breadth-first search algorithm with the time complexity of $O(|V| \cdot |E|)$, which only needs to be run once.

In each search step, for the operated vertex v, we select vertices from $N_1(v)$, $N_2(v)$ and $N_{>2}(v)$ to form its configuration (Algorithm 3). Then, by defining the updating rules for configuration checking, we obtain a concrete PCC strategy. In this work, we apply PCC to two important problems namely MWDS and MWCP. As for the updating rules, we follow those in CC^{2}FS and LSCC algorithms.

A PCC Strategy for MWDS

When removing a vertex u from the candidate solution S

- $confCh[u] := 0$;
- calculate $Conf[u]$ according to Algorithm 3;
- for each vertex $w \in Conf[u]$, $confCh[w]$ is set to 1.

When adding a vertex v to the candidate solution S

- calculate $Conf[v]$ according to Algorithm 3;
- for each vertex $w \in Conf[v]$, $confCh[w]$ is set to 1.

A PCC Strategy for MWCP

When removing a vertex u from the candidate solution S

- $confCh[u] := 0$;

When adding a vertex v to the candidate solution S

- calculate $Conf[v]$ according to Algorithm 3;
- for each vertex $w \in Conf[v]$, $confCh[w]$ is set to 1.

When swapping a vertex u from the candidate solution S with a vertex v

- $confCh[u] := 0$;
- calculate $Conf[v]$ according to Algorithm 3;
- for each vertex $w \in Conf[v]$, $confCh[w]$ is set to 1.

A notable thing for the PC is that the configuration $Conf$ is determined until applying CC rules to $confCh$ array. Therefore, each time we update the states of the configuration, the configuration $Conf$ for the same vertex could be totally different, which favors exploration.

5.3 Complexity Analysis

We compare the complexity of two existing configurations and PC. Given an undirected graph $G = (V, E)$, we use $\Delta(G)$ to denote $max\{|N(v)| \mid v \in V\}$. For the updating rules of the 1-level neighborhoods configuration, the worst time complexity is $O(\Delta(G))$. As for the rules of the 2-level neighborhoods configuration, the worst time complexity is $O(\Delta(G)^2)$. For PC, the worst time complexity is $O(|V|)$ ($p_i = 1.0$ for all $1 \leq i \leq 3$), which is the amount of the vertices in the graph. This could result in a rather time-consuming CC strategy. Fortunately, p_i is not 1.0 after a training process in most cases. As a result, the complexity of PCC is roughly comparable with CC.

5.4 Automatic Configuration Process

There are three parameters, p_1, p_2, and p_3, in the definition of PC. In order to obtain a good configuration for a given COP, we use a configurator similar to the state-of-the-art automatic algorithm configurator – SMAC (Hutter, Hoos, and Leyton-Brown 2011) to configure the probabilistic configuration. We will illustrate the training process for different training sets in Section 6 as follows.

6 Preliminary Results

We conduct extensive experiments to evaluate the viability of the PCC. We first apply it to two different COPs, *i.e.*, MWDS and MWCP. Secondly, we study the performance of special PCC trained for different problems and different benchmarks. Thirdly, we verify the effectiveness of PCC in alleviating the cycling problem. Finally, we analyze parameter sensitivity for p_1, p_2, and p_3.

Benchmarks. We carry out experiments to evaluate the performance of the algorithms using PC for MWDS and MWCP on two standard benchmarks, *i.e.*, DIMACS (37 instances) and BHOSLIB (41 instances). DIMACS benchmark is from the Second DIMACS Implementation Challenge (Johnson and Trick 1996) including problems from real applications and randomly generated graphs. BHOSLIB instances are generated randomly based on the model RB at the phase transition area (Xu et al. 2005). These instances are originally unweighted, and to obtain the corresponding weighted instances, we use the same method as in (Pullan 2008; Wu, Hao, and Glover 2012). For the i-th vertex v_i, we set $w(v_i) = (i \bmod 200) + 1$ to obtain the weighted benchmarks DIMACS and BHOSLIB for MWCP and MWDS. And we also adopt T1 (530 instances), T2 (530 instances), and UDG (120 instances) from (Jovanovic, Tuba, and Simian 2010) for MWDS problem.

Competitors. For each problem, we select the algorithm that achieves state of the art due to the variants of CC.

For MWDS, we select CC^{2}FS (Wang, Cai, and Yin 2017), the algorithm with the first 2-level neighborhoods CC strategy, as the competitor. We implemented the version of CC^{2}FS with PC, namely CC^{2}FS-P. We do not select the algorithm (Wang et al. 2018) because it applies the same configuration as CC^{2}FS, and it is proposed for massive graphs.

For MWCP, we compare with LSCC, the algorithm with SCC strategy (Wang, Cai, and Yin 2016), which is the first

variant that introduces a strict strategy to reduce more unnecessary search spaces. We also implemented the version of LSCC with PC, namely LSCC-P. Note that we do not use the optimized version of LSCC, *i.e.*, LSCC+BMS, because it is optimized for massive graphs. And we do not select the algorithm (Wang et al. 2020) because this algorithm applies the same configuration.

Settings. All competitors are implemented by their authors. All algorithms are implemented in C++ and compiled by g++ with '-O3' option. All experiments are run on a Linux computer with an Intel i7-9800x processor with 3.6 GHz and 64 GB RAM. The cutoff time of all instances is 1000s except that the cutoff time of instances with the number of vertices less than 500 is 50s in T1, T2, and UDG benchmark.

6.1 Comparison PCC with CC

We first separately train PCs for MWDS and MWCP, then compare the performance of CC and PCC in each problem.

	Benchmarks	#inst.	CC^{2}FS		CC^{2}FS-P		sum of diff	
			#w.m.	#w.a.	#w.m.	#w.a.	min	avg
train	T1	5	0	0	**5**	**5**	**-167**	**-167.0**
	T2	5	0	0	**3**	**3**	**-49**	**-49.0**
	UDG	4	**4**	**4**	0	0	30	30.0
	BHOSLIB	4	0	0	**1**	**4**	**-1**	**-8.4**
	DIMACS	8	0	0	0	0	0	0.0
test	T1	525	10	10	**152**	**152**	**-2129**	**-2129.0**
	T2	525	0	0	**57**	**57**	**-697**	**-697.0**
	UDG	116	**81**	**81**	16	16	541	541.0
	BHOSLIB	37	0	0	**6**	**26**	**-52**	**-105.1**
	DIMACS	29	0	0	**2**	**9**	**-4**	**-27.2**
Total		1258	95	95	**242**	**272**	**-2528**	**-2611.7**

Table 3: Comparison results of CC^{2}FS and CC^{2}FS-P on 7 benchmarks, where '#inst.' means the number of instances for training or testing, '#w.m.' means the number of instances where the solver is better in 'min', 'w.a.' means the number of instances where the solver is better in 'avg'. We also report the sum of difference ('sum of diff') between CC^{2}FS-P and CC^{2}FS in 'min' and 'avg'. The smaller the difference, the better for CC^{2}FS-P.

Training process. To obtain the best PC of the algorithm for a COP, we randomly selected a graph from each type of graphs with the largest number of vertices in each benchmark as the training set. Note that for DIMACS, where the graphs are generated for different applications, we chose the graphs with the largest number of vertices in each application. It is reasonable to use the graphs with the largest number of vertices because these graphs potentially contain a number of patterns that show the impact of remote and near vertices on the PCC strategy. Following the recommended protocol (Hutter et al. 2017), we utilize the configurator to optimize solution quality. We allow a time budget of 24 hours for the entire configuration process.

Results for MWDS. Table 3 shows that CC^{2}FS-P is the better solver on these benchmarks except the UDG benchmark. Particularly, CC^{2}FS-P achieves much better minimum and average weights than those of CC^{2}FS on instances. The performance of CC^{2}FS-P for UDG benchmark is not good because it performs worse even in the training set with respect to the UDG benchmark. Therefore, we argue that it results from the adverse bias generated from training data from other benchmarks. We will discuss the bias of training in this Section 6.2 as follows.

Results for MWCP. Table 4 illustrates that LSCC-P achieves better performance than LSCC in BHOSLIB benchmark, while it shows comparable performance with LSCC in DIMACS (the sum of difference is not significant).

	Benchmarks	#inst.	LSCC		LSCC-P		sum of diff	
			#w.m.	#w.a.	#w.m.	#w.a.	min	avg
train	BHOSLIB	4	1	0	**2**	**4**	**13**	**67**
	DIMACS	8	0	1	**3**	**3**	**124**	**166.8**
test	BHOSLIB	37	2	3	**14**	**26**	**315**	**462.7**
	DIMACS	29	**2**	2	0	2	-19	**52.9**
Total		78	5	6	**19**	**35**	**433**	**749.4**

Table 4: Experimental results of LSCC and LSCC-P on DIMACS and BHOSLIB benchmarks. We report the sum of difference between LSCC-P and LSCC in 'min' and 'avg'. The larger the difference, the better for LSCC-P.

COP	Configuration	p_1	p_2	p_3
MWDS	CC^2	1.00	1.00	0.00
	PC	0.24	0.13	0.06
MWCP	SC	1.00	0.00	0.00
	PC	0.49	0.81	0.29

Table 5: The PCs for MWDS and MWCP. CC and CC^2 are original configurations, and PC the train configuration.

Discussion. We report the comparison results between the original configuration designed manually and optimized PCs for each problem after the automatic training process in Table 5. The results show that the optimized PCs are quite different from the configurations designed by human. For example, $p_3 = 0.29$ for MWCP means that there are 29% vertices in the 2^+-level neighborhoods as the configuration.

6.2 Discussion About the Bias of Training

From the results in this Section 6.1, we notice that training a PC based on all benchmarks leads to a decrease in the performance for some specific benchmarks, *e.g.*, UDG benchmark. A natural question raised is whether different benchmarks have different biases for PC. We argue that the bias leads to the PC that is not good for some specific benchmarks.

COP	Benchmarks	Configuration		
		p_1	p_2	p_3
MWDS	T1	0.05	0.06	0.02
	T2	0.60	0.02	0.05
	UDG	0.94	0.02	0.95
	BHOSLIB	0.27	0.02	0.99
	DIMACS	0.00	0.45	0.51
MWCP	BHOSLIB	0.58	0.15	0.98
	DIMACS	0.49	0.81	0.29

Table 6: The PCs for each benchmark.

Training process. To investigate this problem, we retrain the special PCs for each benchmark for each problem. We follow the same training set in Section 6.1. The only difference is that we train PCs for each benchmark using their own training instances. The consistent training setting allows us to compare the newly obtained algorithms (CC^{2}FS-SP for MWDS, LSCC-SP for MWCP) with CC^{2}FS and LSCC-P more clearly. The trained PCs are shown in Table 6.

	Benchmarks	#inst.	CC^{2}FS		CC^{2}FS-SP		sum of diff	
			#w.m.	#w.a.	#w.m.	#w.a.	min	avg
train	T1	5	0	0	**5**	**5**	**-184**	**-184.0**
	T2	5	0	0	**3**	**3**	**-49**	**-49.0**
	UDG	4	1	1	**2**	**2**	**-8**	**-8.0**
	BHOSLIB	4	0	0	**1**	**4**	**-1**	**-8.4**
	DIMACS	8	0	0	0	0	0	0.0
test	T1	525	10	10	**154**	**154**	**-2205**	**-2205.0**
	T2	525	0	0	**58**	**58**	**-697**	**-697.0**
	UDG	116	**47**	**47**	41	41	**-14**	**-14.0**
	BHOSLIB	37	0	0	**6**	**26**	**-52**	**-109.1**
	DIMACS	29	0	0	**2**	**9**	**-4**	**-27.1**
Total		1258	58	58	**272**	**302**	**-3214**	**-3301.6**

Table 7: Comparison results of CC^{2}FS and CC^{2}FS-SP.

	Benchmarks	#inst.	CC^{2}FS-P		CC^{2}FS-SP		sum of diff	
			#w.m.	#w.a.	#w.m.	#w.a.	min	avg
train	T1	5	0	0	**2**	**2**	**-17**	**-17.0**
	T2	5	0	0	0	0	0	0.0
	UDG	4	0	0	**4**	**4**	**-38**	**-38.0**
	BHOSLIB	4	0	0	0	0	0	0.0
	DIMACS	8	0	0	0	0	0	0.0
test	T1	525	17	17	**36**	**36**	**-76**	**-76.0**
	T2	525	1	1	1	1	0	0.0
	UDG	116	22	22	**82**	**82**	**-555**	**-555.0**
	BHOSLIB	37	0	0	0	**1**	0	**-4.0**
	DIMACS	29	0	**1**	0	0	0	0.1
Total		1258	40	41	**125**	**126**	**-686**	**-689.9**

Table 8: Comparison results of CC^{2}FS-P and CC^{2}FS-SP.

Results for MWDS and MWCP. We report the results of MWDS in Table 7 and Table 8. CC^{2}FS-SP achieves a better performance than CC^{2}FS on three benchmarks. In UDG benchmark, CC^{2}FS-SP and CC^{2}FS are comparable. Compared with CC^{2}FS-P, CC^{2}FS-SP shows obvious advantages on T1 and UDG benchmarks, particularly UDG benchmark. Table 9 shows the runtime comparison among three algorithms on MWDS benchmark. For most cases, compared with CC^{2}FS, CC^{2}FS-P and CC^{2}FS-SP faster find better results and are more robust ('#best' is higher). Combined with the differences between the PCs for UDG benchmark in Table 6 and that in Table 5, this improvement suggests that different benchmarks have their own best PCs.

The experiment for MWCP illustrates the similar results (Table 10 and Table 11): the PC specially trained for each benchmark can improve the performance. Surprisingly, in DIMACS benchmark, the performance of LSCC-SP and LSCC-P are the same because they share the same PC. Table 12 illustrates the runtime comparison among three algorithms on MWCP benchmark. It shows that LSCC-P and

	Benchmarks	CC^{2}FS		CC^{2}FS-P		CC^{2}FS-SP	
		time	#best	time	#best	time	#best
train	T1	-	0	467.81	3	**174.28**	**5**
	T2	**0.23**	2	156.64	**5**	29.98	**5**
	UDG	44.69	2	-	0	**4.45**	**3**
	BHOSLIB	**5**	12	18.21	**40**	6.47	**40**
	DIMACS	7.56	**80**	6.02	**80**	**5.39**	**80**
test	T1	**5.93**	370	20.36	485	21.48	**503**
	T2	5.32	467	7.8	**524**	**1.97**	**524**
	UDG	**14.7**	**72**	50.99	25	25.11	61
	BHOSLIB	14.39	219	30.24	361	**10.31**	**366**
	DIMACS	**1.53**	240	11.4	**288**	6.35	287

Table 9: Runtime and solution comparison, where '#best' means the number of runs where the solver obtains the best result of the instances among these three algorithms and 'time' means the average runtime for the best result.

LSCC-SP are better solvers than LSCC in BHOSLIB benchmark, while they are comparable in DIMACS benchmark. Their comparable results show that the manual designed configuration and the training configuration are both suitable for MWCP on the DIMACS benchmark.

	Benchmarks	#inst.	LSCC		LSCC-SP		sum of diff	
			#w.m.	#w.a.	#w.m.	#w.a.	max	avg
train	BHOSLIB	4	1	0	**2**	**4**	**8**	**69.4**
	DIMACS	8	0	1	**3**	**3**	**124**	**166.8**
test	BHOSLIB	37	2	2	**14**	**27**	**371**	**502.3**
	DIMACS	29	**2**	2	0	2	-19	**52.9**
Total		78	5	5	**19**	**36**	**484**	**791.4**

Table 10: Comparison results of LSCC and LSCC-SP.

	Benchmarks	#inst.	LSCC-P		LSCC-SP		sum of diff	
			#w.m.	#w.a.	#w.m.	#w.a.	max	avg
train	BHOSLIB	4	1	2	**2**	2	-5	**2.4**
	DIMACS	8	0	0	0	0	0	0
test	BHOSLIB	37	**9**	8	6	**15**	**56**	**39.6**
	DIMACS	29	0	0	0	0	0	0
Total		78	**10**	10	8	**17**	**51**	**42**

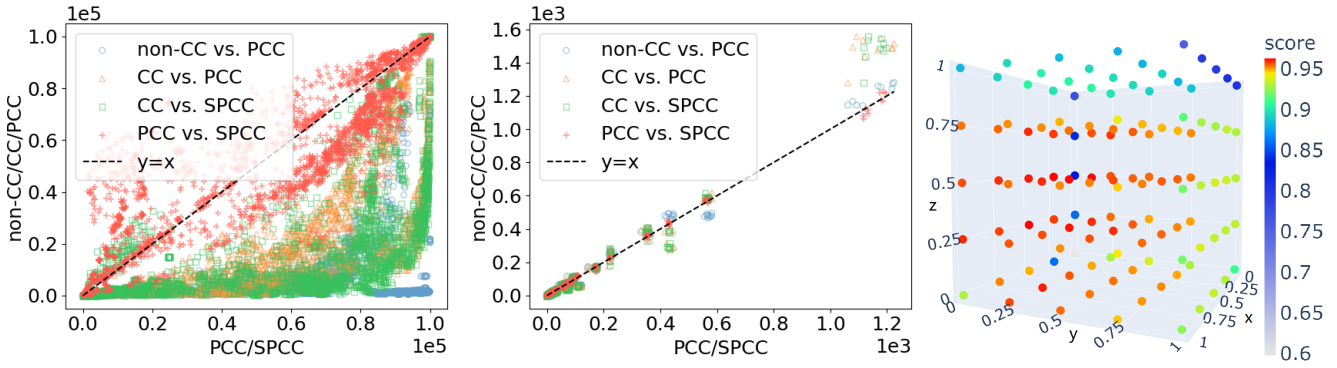
Table 11: Comparison results of LSCC-P and LSCC-SP.

Summary. Overall, our method can automatically provide an excellent configuration for a CC strategy by training without the need for careful design. Besides, our experiment confirms that the bias of training does affect the performance. We can improve the performance by training special PCs for different benchmarks for different COPs.

We also compare the performance among PCC, SPCC and HCC which we introduce in Section 4 to confirm our motivation. The results show that it is necessary to train for the best PC rather than just setting all probabilities to 0.5.

6.3 Performance on Reducing the Cycling Problem

To verify that PCC can bring more potential explorations, we calculate the number of different candidate solutions each



(a) (b) (c)

Figure 1: (a) Comparison of the number of different candidate solutions of non-CC, CC, PCC, SPCC. The X-axis and Y-axis represent the number of different candidate solutions in 10^5 candidate solutions. Non-CC includes CC^{2}FS and LSCC without CC strategy. CC includes CC^{2}FS and LSCC. PCC includes CC^{2}FS-P and LSCC-P. SPCC includes CC^{2}FS-SP and LSCC-SP. (b) Comparison of CPU time. The X-axis and Y-axis represent the execution time in seconds to search for 10^5 candidate solutions. (c) The scores of CC^{2}FS-P with different configurations on MWDS benchmark. The X-axis, Y-axis, and Z-axis represent p_1, p_2, and p_3 respectively.

	Benchmarks	LSCC time	LSCC #best	LSCC-P time	LSCC-P #best	LSCC-SP time	LSCC-SP #best
train	BHOSLIB	664.02	5	**439.36**	**8**	590.35	7
	DIMACS	**93.3**	49	116.76	**56**	116.76	**56**
test	BHOSLIB	237.67	147	176.61	**194**	**158.42**	191
	DIMACS	22.44	**269**	**13.14**	265	**13.14**	265

Table 12: Runtime comparison of LSCC, LSCC-P and LSCC-SP on BHOSLIB and DIMACS benchmarks.

algorithm has found in the first 10^5 candidate solutions. In addition, we report the cost of time to search the 10^5 candidate solutions, in order to show that PCC does not cost much time to make the improvement in exploration.

Competitors. We compare among CC^{2}FS, CC^{2}FS-P and CC^{2}FS-SP for MWDS and compare among LSCC, LSCC-P, and LSCC-SP for MWCP. CC^{2}FS-P and LSCC-P use the PCs in Table 5. CC^{2}FS-SP and LSCC-SP use the PCs in Table 6. To show the improvement in the exploration of CC, PCC, and SPCC, we also choose the non-CC versions of CC^{2}FS and LSCC, where the CC strategy is removed.

Settings and Benchmarks. Experimental settings and benchmarks are the same as those described in Section 6.

Results. As Figure 1(a) shows, CC, PCC, and SPCC enhance the ability to explore new candidate solutions, PCC and SPCC improve much more than CC in nearly all instances. Meanwhile, the cost of time (Figure 1(b)) of PCC and SPCC are comparable with that using the other two configurations and even better when the cost of time is large.

Summary. The result shows that PCC does not have much more time consumption to improve the exploration in local search and potentially alleviates the cycling problem.

To verify that it is necessary to tune for an optimal PC for a given COP rather than using a random PC, we discuss the parameter sensitivity. We calculate the scores of CC^{2}FS-P with different PCs for MWDS, where p_1, p_2, and p_3 vary from 0 to 1 in steps of 0.25. The score of the i-th PC is calculated as Equation 1, where $score_{i,j}$ means the average value of the solutions found in the j-th instance with the i-th PC. Equation 1 shows that the score of the better PC is nearer to 1. In Figure 1(c), the scores of different PCs are quite different, which suggests that it is hard to obtain a best performance with a random PC. The performance under most of the PCs we enumerate is worse than the PC used for MWDS in Table 5. So it is necessary to tune for an optimal PC for a given COP.

$$score_i = \text{mean}_j \frac{\max_k score_{k,j} - score_{i,j}}{\max_k score_{k,j} - \min_k score_{k,j}} \quad (1)$$

7 Conclusions and Future work

The cycling problem is an intractable problem for COPs. We have observed that neighborhoods with different levels should have different contributions to alleviate the cycling problem. Therefore, we have expanded the configuration with probability, *i.e.*, *probabilistic configuration* (PC), to capture the contributions of vertices at different levels, resulting in the *probabilistic configuration checking* (PCC) strategy. Our experimental results have confirmed that the PC can improve existing local search algorithms in two classic COPs, *i.e.*, MWDS and MWCP, due to alleviating the cycling problem.

Future work include making the method suitable for massive graphs and extending our approach to other COPs.

Acknowledgments

We thank anonymous referees for helpful comments. This paper was supported by the National Natural Science Foundation of China (No. 61976232), Guangdong Basic and Applied Basic Research Foundation (No. 2022A1515011355 and 2020A1515010642), Guizhou Science Support Project

(No. 2022-259), Humanities and Social Science Research Project of Ministry of Education (18YJCZH006).

References

Abramé, A.; Habet, D.; and Toumi, D. 2017. Improving configuration checking for satisfiable random k-SAT instances. *Annals of Mathematics and Artificial Intelligence*, 79(1-3): 5–24.

Cai, S.; and Su, K. 2012. Configuration Checking with Aspiration in Local Search for SAT. In *AAAI*.

Cai, S.; and Su, K. 2013. Local search for Boolean Satisfiability with configuration checking and subscore. *Artificial Intelligence*, 204: 75–98.

Cai, S.; Su, K.; Luo, C.; and Sattar, A. 2013. NuMVC: An efficient local search algorithm for minimum vertex cover. *Journal of Artificial Intelligence Research*, 46: 687–716.

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

Chen, P.; Wan, H.; Cai, S.; Li, J.; and Chen, H. 2020. Local Search with Dynamic-Threshold Configuration Checking and Incremental Neighborhood Updating for Maximum k-plex Problem. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence, (AAAI 2020)*, 2343–2350.

Glover, F. 1989. Tabu search—part I. *ORSA Journal on computing*, 1(3): 190–206.

Glover, F. 1990. Tabu search—part II. *ORSA Journal on computing*, 2(1): 4–32.

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2011. Sequential model-based optimization for general algorithm configuration. In *LION*, 507–523.

Hutter, F.; Lindauer, M.; Balint, A.; Bayless, S.; Hoos, H.; and Leyton-Brown, K. 2017. The Configurable SAT Solver Challenge (CSSC). *Artificial Intelligence*, 243: 1–25.

Johnson, D. S.; and Trick, M. A. 1996. *Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993*, volume 26. American Mathematical Soc.

Jovanovic, R.; Tuba, M.; and Simian, D. 2010. Ant colony optimization applied to minimum weight dominating set problem. In *Proceedings of the 12th WSEAS international conference on Automatic control, modelling & simulation, (ACMOS-2010)*, 322–326.

Li, R.; Cai, S.; Hu, S.; Yin, M.; and Gao, J. 2018. NuMWVC: A Novel Local Search for Minimum Weighted Vertex Cover Problem. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-2018)*, 8107–8108.

Luo, C.; Cai, S.; Su, K.; and Huang, W. 2017. CCEHC: An efficient local search algorithm for weighted partial maximum satisfiability. *Artificial Intelligence*, 243: 26–44.

Luo, C.; Cai, S.; Su, K.; and Wu, W. 2015. Clause States Based Configuration Checking in Local Search for Satisfiability. *IEEE Transactions on Cybernetics*, 45(5): 1014–1027.

Michiels, W.; Aarts, E.; and Korst, J. 2007. *Theoretical aspects of local search*. Springer Science & Business Media.

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

Wang, Y.; Cai, S.; Chen, J.; and Yin, M. 2018. A Fast Local Search Algorithm for Minimum Weight Dominating Set Problem on Massive Graphs. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, (IJCAI-2018)*, 1514–1522.

Wang, Y.; Cai, S.; Chen, J.; and Yin, M. 2020. SCCWalk: An efficient local search algorithm and its improvements for maximum weight clique problem. *Journal of Artificial Intelligence*, 280: 103230–103263.

Wang, Y.; Cai, S.; and Yin, M. 2016. Two efficient local search algorithms for maximum weight clique problem. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, (AAAI-2016)*, 805–811.

Wang, Y.; Cai, S.; and Yin, M. 2017. Local Search for Minimum Weight Dominating Set with Two-Level Configuration Checking and Frequency Based Scoring Function. *Journal of Artificial Intelligence Research*, 58: 267–295. ArXiv: 1702.04594.

Wu, Q.; Hao, J.-K.; and 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.; and Lecoutre, C. 2005. A simple model to generate hard satisfiable instances. *arXiv preprint cs/0509032*.