Soft Computing (2020) 24:3551–3567
https://doi.org/10.1007/s00500-019-04122-z

METHODOLOGIES AND APPLICATION

A novel parallel local search algorithm for the maximum vertex weight clique problem in large graphs

Ender Sevinc[2] · **Tansel Dokeroglu**[1]

Published online: 11 June 2019

Abstract
This study proposes a new parallel local search algorithm (Par-LS) for solving the maximum vertex weight clique problem (MVWCP) in large graphs. Solving the MVWCP in a large graph with millions of edges and vertices is an intractable problem. Parallel local search methods are powerful tools to deal with such problems with their high-performance computation capability. The Par-LS algorithm is developed on a distributed memory environment by using message passing interface libraries and employs a different exploration strategy at each processor. The Par-LS introduces new operators parallel(ω,1)-swap and parallel(1,2)-swap, for searching the neighboring solutions while improving the current solution through iterations. During our experiments, 172 of 173 benchmark problem instances from the DIMACS, BHOSLIB and Network Data Repository graph libraries are solved optimally with respect to the best/optimal reported results. A new best solution for the largest problem instance of the BHOSLIB benchmark (*frb100-40*) is discovered. The Par-LS algorithm is reported as one of the best performing algorithms in the literature for the solution of the MVWCP in large graphs.

Keywords Maximum clique problem · Parallel search · Vertex weight · MPI

1 Introduction

The maximum vertex weight clique problem (MVWCP) is a general form of the maximum clique problem (MCP) (Wu and Hao 2016; Zhou et al. 2017a). The MVWCP decides a clique with the maximum total value of vertices' weight. When each vertex of a graph has weight 1, then MVWCP becomes the classical MCP (Wu and Hao 2015b; Wu et al. 2012). The MCP is NP-complete, and MVWCP is at least as hard as MCP (Alidaee et al. 2007; Dijkhuizen and Faigle 1993). Given an undirected graph $G = (V, E)$ with vertex set of $V = \{1,\ldots, n\}$ and edge set of $E \subseteq V \times V$. Let $w : V \rightarrow Z^+$ be a weighting function that assigns each vertex $v \in V$ a positive value. The MVWCP has many applications in different areas such as computer vision (Ma and Latecki 2012), coding theory (Zhian et al. 2013), bioinformatics (Zheng et al. 2007), protein structure prediction (Mascia et al. 2010), community (cluster) detection (Tepper and Sapiro 2013), combinatorial auctions (Wu and Hao 2015a; Li et al. 2018).

Exact (brute-force) algorithms that are proposed for solving the MVWCP require too much time and computation power due to the intractable nature of this problem. Therefore, local search techniques and heuristics are more feasible approaches for large and dense graph instances, because they can obtain high-quality solutions in practical times. Here, we can list some of the most important algorithms that are proposed for the solution of the MC and MVWCP. Kumlander (2004) proposes a backtrack tree search algorithm that relies on a heuristic coloring-based vertex order. The algorithm is a brute-force algorithm that is based on a fact that vertices from the same independent set could not be included in the same maximum clique. Those sets are obtained from a heuristic vertex coloring. Color classes and a backtrack search are used for pruning branches of the maximum weight clique search tree. Warren and Hicks (2006) propose three B&B algorithms for the maximum weight independent set problem. The algorithms use weighted clique covers to generate upper bounds, and all perform branching and using according to the method of Balas and Yu (1986). Pullan and Hoos (2006) propose a new stochastic local search algorithm (DLS-MC)

Communicated by V. Loia.

✉ Tansel Dokeroglu
tansel.dokeroglu@tedu.edu.tr

[1] Department of Computer Engineering, TED University, Ankara, Turkey

[2] Department of Computer Engineering, THK University, Ankara, Turkey

for the maximum clique problem. The DLS-MC algorithm alternates between phases of iterative improvement, during which suitable vertices are added to the current clique, and plateau search, during which vertices of the current clique are swapped with vertices not contained in the current clique. Wu et al. (2012) introduce a tabu search heuristic whose key features include a combined neighborhood and a dedicated tabu mechanism using a randomized restart strategy for diversification. Benlic and Hao (2013) present breakout local search (BLS) algorithm. The BLS can be applied to both MC and MVWCP problems without any particular adaptation. BLS explores the search space by joint use of local search and adaptive perturbation strategies. Wang et al. (2016a) recast the MVWCP into a model that is solved by a probabilistic tabu search algorithm designed for binary quadratic programming (BQP). El Baz et al. (2016) propose a parallel ant colony optimization-based meta-heuristic (PACOM) for solving MVWCP. Zhou et al. (2017b) introduce a move operator called PUSH that generalizes the conventional add and swap operators commonly used and can be employed in a local search process for the MVWCP. Nogueira et al. (2017) introduce a hybrid iterated local search technique for the maximum weight independent set (MWIS) problem, which is related to MVWCP. Nogueira and Pinheiro (2018) present a CPU–GPU local search heuristic for solving the MWCP in massive graphs. The neighborhoods are explored using an efficient procedure that is suitable to be mapped onto a GPU-based massively parallel architecture. The algorithm outperforms the best-known heuristics for the MWCP with its speed-up of up to 12 times over the CPU-only implementation. Kiziloz and Dokeroglu (2018) propose a robust and cooperative parallel tabu search algorithm (PTC) for the MVWCP.

Cai et al. (2016) propose a new method that interleaves between clique construction and graph reduction. They propose three novel approaches and design FastWClq algorithm. Wang et al. (2016b) introduce two heuristics and propose two local search algorithms for the MWCP, strong configuration checking (SCC), and develop a local search algorithm named LSCC. In order to improve the performance, a low-complexity heuristic best from multiple selection (BMS) is applied to select the swapping vertex pair quickly and effectively (LSCC+BMS algorithm). The proposed algorithms outperform the state-of-the-art local search algorithm MN/TS and its improved version MN/TS+BMS.

An exact algorithm, branch and bound (B&B) for the MVWCP (WLMC), is suited for large vertex-weighted graphs by Jiang et al. (2017). A new B&B algorithm (TSM-MWC) for the MVWCP is proposed by Jiang et al. (2018). The proposed algorithm is an exact algorithm and uses MaxSAT reasoning to reduce the search space. Another B&B MWCP algorithm (WC-MWC) that reduces the number of branches of the search space incrementally is proposed by Li et al. (2018). Experimental results show that the algorithm WC-MWC outperforms some of the best-performing exact and heuristic MWCP algorithms on both small/medium graphs and real-world massive graphs.

Parallel local search algorithms have been reported to be effective tools for the optimization of NP-hard problems for the last few decades (Kiziloz and Dokeroglu 2018; Cantú-Paz 1998; Dokeroglu 2015; Dokeroglu and Mengusoglu 2017; Kucukyilmaz and Kiziloz 2018). In our opinion, a scalable parallel heuristic algorithm with intelligent and cooperative operators can improve the solution quality of an optimization process significantly. Our study proposes a novel parallel local search algorithm (Par-LS) for the solution of the MVWCP in large graphs. The Par-LS algorithm is specially developed for very large in-memory graphs with millions of edges and vertices. Exploring and exploiting the search space of large graphs requires a great deal of computation power. The search space is intractable, and exact (brute-force) algorithms are not efficient enough to solve the MVWCP in feasible execution times. The Par-LS algorithm introduces a new parallel local search technique with new operators parallel(ω,1)-swap and parallel(1,2)-swap for searching neighboring solutions. The operators are adapted to parallel computation to diversify the exploration by selecting different vertices during the addition and deletion of the vertices. These operators are used in parallel for the first time in the literature (Lourenço et al. 2010; Hansen et al. 2010). The local search Par-LS algorithm uses some algorithmic parameters as it is required by most of the meta-heuristic algorithms. As we experience from our previous studies, tuning the parameters of a heuristic algorithm increases the performance of a local search algorithm significantly. Therefore, we develop and employ a simple and efficient parameter-setting technique for the optimization process of the Par-LS. At each processor, we use a different set of parameters for the Par-LS algorithm. The parameters are randomly selected from a range of values.

The Par-LS is an enhanced parallel version of a recent algorithm called a hybrid iterated local search (ILS-VND) that is proposed for the maximum weight independent set problem (Nogueira et al. 2017). We adapt this algorithm to the MVWCP by evolving its operators to the parallel environment, introducing a new parallel parameter tuning technique and a seeding mechanism for each processor that provides a diversified searching capability. 173 problem instances are optimized from the DIMACS, BHOSLIB and Network Data Repository graph libraries. 172 problems are solved optimally with respect to the optimal/best-known solutions of the problem instances. A new best solution for the largest problem instance of the BHOSLIB benchmark (*frb100-40*) is discovered. The evaluation of the experiments shows that the Par-LS algorithm can outperform most of the state-of-

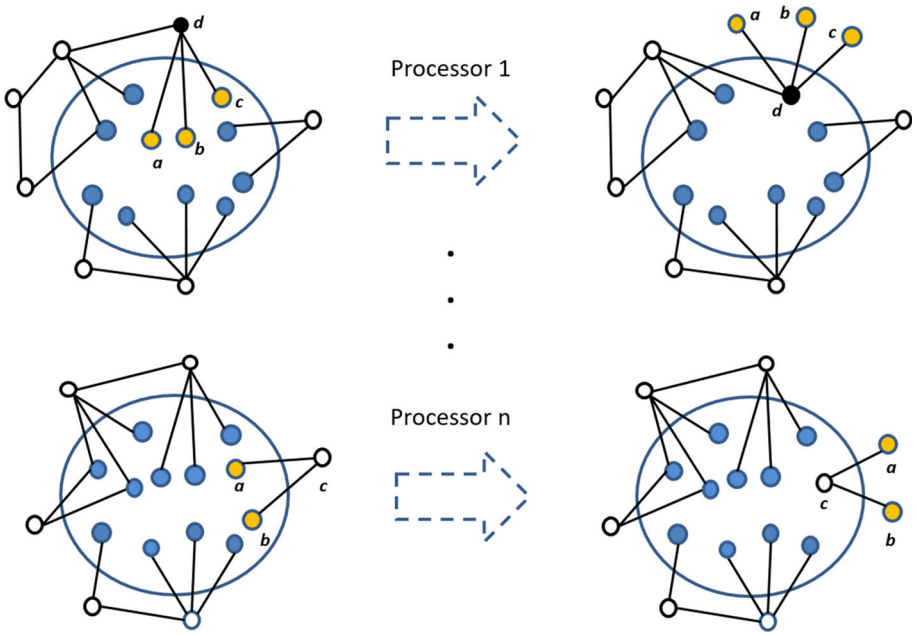

Fig. 1 Parallel (ω,1)-swap operator is concurrently operating with several processors on different solution candidates that are different than one another. Processor 1 inserts vertex d into the current solution and removes vertices a, b, and c. Processor n inserts vertex c into the current solution while vertices a and b are being removed

the-art heuristic algorithms and can be reported as one of the best algorithms.

The Par-LS algorithm is introduced in Sect. 2. Section 3 gives the details for the performance evaluation of the experimental results and comparison of the algorithm to the state-of-the-art algorithms on selected set of large graphs. Concluding remarks and future work are provided in the last section.

2 Proposed parallel local search algorithm, Par-LS

In this section, we present our proposed Par-LS algorithm for the MVWCP. Par-LS is an enhanced parallel version of the ILS-VND algorithm introduced by Nogueira et al. (2017). The Par-LS is developed by using C++ and Message Passing Interface (MPI) library. A seeding function is used to diversify random number generation at each node while initializing a starting point. A star communication topology is used between processors. The master node/processor controls the slaves and receives their best solutions at the end of the optimization process.

Parallel (ω,1)*-swap operator* adds a new vertex, v, to the existing solution and deletes vertices that are neighbors of the v. This is the newer parallel version of the operators introduced by Nogueira et al. (2017). The total weight of the new generated clique is calculated, and if its vertex weight is better than the existing one, it replaces the older clique. With its distributed nature and a seeding function that makes use of the ranks of the processors in the environment, parallel (ω,1)-swap operator generates and searches the new cliques efficiently by selecting diversified vertices at each processor. In Fig. 1, we can see how the parallel (ω,1)-swap operator works on the current solutions at different processors concurrently. If there are n number of processors in the computation environment, randomly selected vertices are inserted to the current solution and its neighbors inside the clique are removed. The parallel (ω,1)-swap operator provides a diversified exploration search space and it is a very efficient way of optimizing the maximum vertex weight clique.

Parallel (1,2)-swap operator deletes one vertex and adds two vertices to the current solution. Depending on the rank number of the processors, the selection of the vertices for insertion and deletion is executed from well-diversified locations. Therefore, our new enhanced operators, parallel (ω,1)-swap and parallel (1,2)-swap operators are very efficient while exploring the search space of large graphs. In Fig. 2, we can follow how the parallel(1,2)-swap operator works on different candidate solutions concurrently. The parallel(1,2)-swap selects two vertices from outside of the current solution.

Generate_a_random_Clique function selects a random vertex number and continues adding new vertices to construct an initial maximum vertex weight clique. This procedure is repeated until there is no examined vertex left. The perturbation function changes the current clique by adding new vertices and removing older ones randomly depending on the seeding mechanism of the processor. Local Search function uses our two new distributed operators parallel (ω,1)-swap and parallel (1,2)-swap during the optimization. The selection of the new vertices continues until the new vertex fails to improve the quality of the solution.

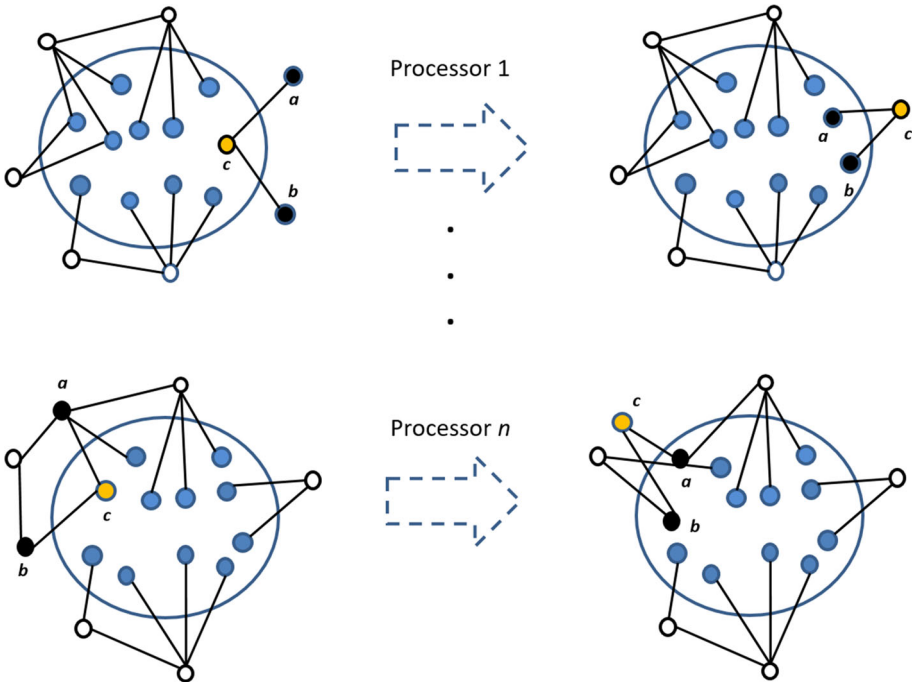

Fig. 2 parallel(1,2)-swap operator is concurrently operating with several processors. Processor 1 inserts the vertices *a* and *b* into the current solution and removes the vertices *c*. Processor *n* inserts vertices *a* and *b* into the current solution while vertex *c* is being removed

Acceptance function monitors the exploration and intensification processes during the optimization. If a new clique is better, it is always accepted. Parameters (p_1, p_2, p_3, p_4) are used as search exploration and intensification parameters in the Par-LS algorithm. p_1 is for shrinking the non-solution vertices set where uniformly chosen non-solution vertices into the current solution. The smaller the size is, the shorter the search time will be. p_2 is for limiting the search space whenever a local maximum is met. This will decrease the search time. p_3 is similar to p_2. When a global maximum is reached, the counter is adjusted for less exploring the search space. p_4 is similar to p_1 which commonly takes the values between 1 and 4. It again forces and squeezes the search space for spending less exploration time. The flowchart of the algorithm is presented in Fig. 3. Algorithms 1, 2, and 3 give the details of the Par-LS Algorithm.

Parameter tuning technique of the Par-LS algorithm The performance of the local search algorithms mostly depends on selecting the best algorithmic specific parameters. Deciding the algorithmic parameters is a crucial process for good performance. In order to provide (near-)optimal settings for the Par-LS algorithm, we apply a simple mechanism that generates different sets of parameters at each processor. The Par-LS algorithm uses these four parameters during the optimization, and each parameter set is selected as a different set at each processor. The parameters are randomly decided within a range of values. This technique provides an efficient way to optimize with different set of values. In addition to this, the structure of the optimization problems is varied. Therefore, an optimized set of parameters may not be a good preference for all the problem instances (Pullan 2008).

Algorithm 1: Proposed parallel local search algorithm, Par-LS

Data: $G = (V, E)$, *termination_limit*
Result: Maximum Vertex Weight Clique C^*

```
1 if (Master node) then
2     receive_results_from_slaves(); find_the_best_result();
      report_the_best_result();
3 else
4     (Slave node) C_0 = Generate_a_random_Clique(G); C=
      Local_Search(C_0, G); // current clique C*= C; // best clique
      solution best_weight = Weight(C); //best local weight counter
      = 0 ; // iterations k = 1; // local iterations
5     while (counter ++ ≤ termination_limit) do
6         C' = Perturb(p_1, C, G); C' = Local_Search(C', G); (C,
          C*, k , best_weight) = Accept(C, C*, C', k, best_weight,
          G)
7     end
8     send C* to the master node;
9 end
```

3 Performance evaluation of the experimental results

We give details of our high-performance computation environment, problem instances, solution quality, execution time,

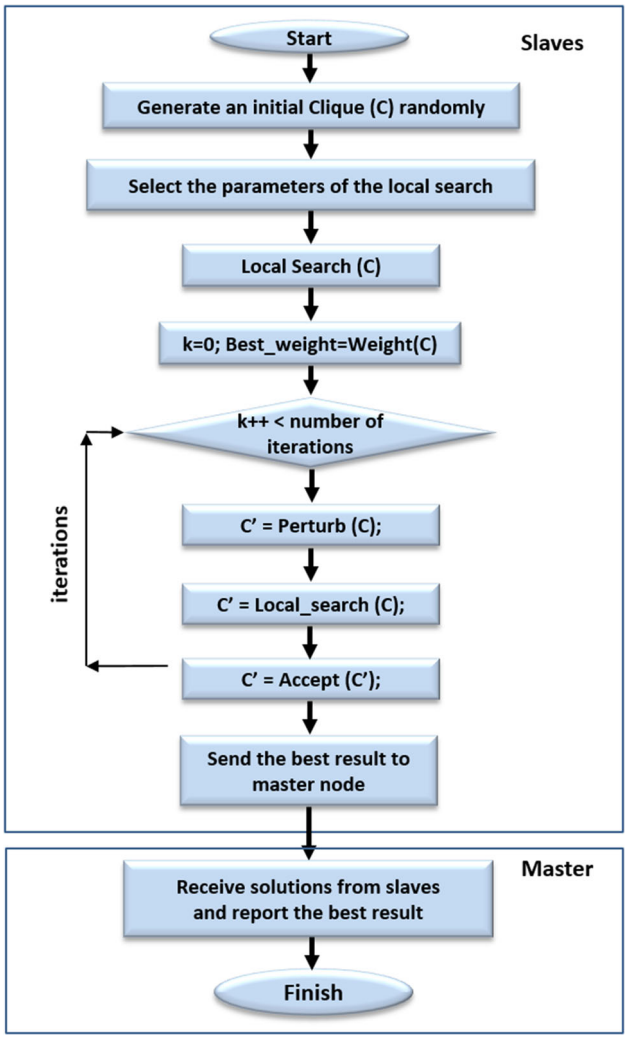

Fig. 3 Flowchart of the Par-LS algorithm

Algorithm 2: Local search procedure

Data: Clique C, Graph $G = (V, E)$
Result: Maximum vertex weight clique C

```
1 k=1; // structure selector while k ≤ 2 do
2   C' = FirstImprovement(k, C, G) if Weight(S') ≤ Weight(C)
    then
3     k ++ ;
4   else
5     k = 1; C = C'; C = AddFreeVertices(C, G); // add free
      vertices randomly
6   end
7 end
8 return C;
```

Algorithm 3: Algorithm of the acceptance function

Data: Current solution C, best solution C^*, candidate solution C'
Result: New current solution C, new best solution C^*

```
1 Acceptance(C, C*, C', i, local_best_w, G) if Weight(C)<
  Weight(C') then
2   C = C' // accept a solution that improves the current one i = 1;
    if local_best_w < Weight(C) then
3     local_best_w = Weight(C) i = i - (| C | / c2);
4   end
5   if Weight(C*) < Weight(C) then
6     C* = C; i = i - (| C | * c3);
7   end
8 else
9   if i ≤ | C |/c2 then
10    i++;
11  else
12    // if the current solution is not improved and i >| C |/c2
      local_best_w = Weight(C); C = Perturb(c4, C, G); i = 1;
13  end
14 end
15 return C, C*, i, local_best_w
```

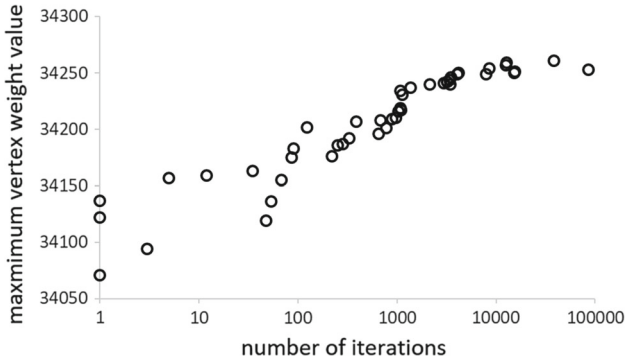

Fig. 4 The effect of increasing the number of iterations on *MANN-a45* problem instance

Table 1 The effect of increasing the number of processors for the *MANN-a45* problem instance. The number of processors is increased from 1 to 128 for the instance, and the performance is observed. 0.4% improvement is obtained

#processors	Result
1	34252.1
2	34253.3
4	34256.2
8	34258.4
16	34260.4
32	34263.7
64	34264.6
128	34265.0

speed-up, and scalability evaluations of the Par-LS algorithm. We discuss the Par-LS algorithm and compare its performance with state-of-the-art MVWCP algorithms for DIMACS,[1] BHOSLIB,[2] and selected large graphs from Network Data Repository.[3]

[1] http://www.cs.hbg.psu.edu/txn131/clique.html.
[2] http://iridia.ulb.ac.be/.
[3] http://networkrepository.com/.

Table 2 Detailed results of the Par-LS algorithm on 80 DIMACS-W benchmark instances **node** is the number of vertices ω is the maximum clique size, **W_best** is the best value that is found until now, **best** is the best value found by the algorithm, $|C|$ is the cardinality of the obtained maximum weighted clique, **avg-sum** is the average of the results, **#proc** is the number of processors used during the optimization process

Instance	node	ω	W_best)	best)	$\|C\|$	avg-sum	Time (s)	#proc.
brock200_1	200	21	2821	2821	19	2821	0.1	2
brock200_2	200	12	1428	1428	9	1428	0.1	2
brock200_3	200	15	2062	2062	13	2062	0.4	2
brock200_4	200	17	2107	2107	13	2107	36.2	2
brock400_1	400	27	3422	3422	21	3422	0.1	2
brock400_2	400	29	3350	3350	21	3350	0.3	2
brock400_3	400	31	3471	3471	23	3471	0.1	2
brock400_4	400	33	3626	3626	22	3626	0.8	2
brock800_1	800	23	3121	3121	20	3121	0.1	2
brock800_2	800	34	3043	3043	18	3043	0.1	2
brock800_3	800	25	3076	3076	20	3076	0.1	2
brock800_4	800	26	2971	2971	26	2971	0.1	2
C125.9	125	34	2529	2529	30	2529	0.7	2
C250.9	250	44	5092	5092	40	5092	0.1	2
C500.9	500	57	6955	6955	48	6955	0.1	2
C1000.9	1000	68	9254	9254	61	9254	176.3	2
C2000.5	2000	16	2466	2466	14	2466	1.7	2
C2000.9	2000	80	10,999	10,999	72	10,999	1600.2	64
C4000.5	4000	18	2792	2792	16	2792	30.4	64
Dsjc500.5	500	13	1725	1725	12	1725	4.2	64
Dsjc1000.5	1000	15	2186	2186	13	2186	1.4	64
keller4	171	11	1153	1153	11	1153	0.1	10
keller5	776	27	3317	3317	27	3317	0.1	10
keller6	3361	59	8062	8062	56	8062	58.4	64
MANN_a9	45	16	372	372	16	372	0.2	10
MANN_a27	378	126	12283	12283	126	12283	0.8	10
MANN_a45	1035	345	34254	34,254	342	34,254	2445.2	10
MANN_a81	3321	1100	111,400	111,400	1100	111,395.4	3401.8	64
hamming6-2	64	32	1072	1072	32	1072	0.1	2
hamming6-4	64	4	134	134	4	134	0.1	2
hamming8-2	256	128	10,976	10,976	128	10,976	0.1	10
hamming8-4	256	16	1472	1472	16	1472	0.1	10
hamming10-2	1024	512	50,512	50,512	512	50,512	0.1	10
hamming10-4	1024	40	5129	5129	35	5129	8.5	64
gen200_p0.9_44	200	44	5043	5043	37	5043	0.1	10
gen200_p0.9_55	200	55	5416	5416	52	5416	0.1	10
gen400_p0.9_55	400	55	6718	6718	47	6718	0.1	10
gen400_p0.9_65	400	65	6940	6940	48	6940	0.1	10
gen400_p0.9_75	400	75	8006	8006	75	8006	0.1	10
c-fat200-1	200	12	1284	1284	12	1284	0.1	10
c-fat200-2	200	24	2411	2411	23	2411	0.1	10
c-fat200-5	200	58	5887	5887	58	5887	0.1	10
c-fat500-1	500	14	1354	1354	12	1354	0.1	10
c-fat500-2	500	26	2628	2628	24	2628	0.1	10
c-fat500-5	500	64	5841	5841	62	5841	0.1	10
c-fat500-10	500	126	11,586	11,586	124	11,586	0.1	10
johnson8-2-4	28	4	66	66	4	66	0.1	10
johnson8-4-4	70	14	511	511	14	511	0.1	10

Table 2 continued

Instance	node	ω	W_best)	best)	$\|C\|$	avg-sum	Time (s)	#proc.
johnson16-2-4	120	8	548	548	8	548	0.1	10
johnson32-2-4	496	16	2033	2033	16	2033	0.1	10
p_hat300-1	300	8	1057	1057	7	1057	0.1	10
p_hat300-2	300	25	2487	2487	20	2487	0.1	10
p_hat300-3	300	36	3774	3774	29	3774	0.1	10
p_hat500-1	500	9	1231	1231	8	1231	0.1	10
p_hat500-2	500	36	3920	3920	31	3892	0.1	10
p_hat500-3	500	50	5375	5375	42	5375	0.1	10
p_hat700-1	700	11	1441	1441	9	1441	0.1	10
p_hat700-2	700	44	5290	5290	40	5290	0.1	10
p_hat700-3	700	62	7565	7565	58	7565	0.1	10
p_hat1000-1	1000	10	1514	1514	9	1514	0.1	10
p_hat1000-2	1000	46	5777	5777	40	5777	0.1	10
p_hat1000-3	1000	68	8111	8111	58	8111	0.1	10
p_hat1500-1	1500	12	1619	1619	10	1619	0.1	10
p_hat1500-2	1500	65	7360	7360	58	7360	0.1	10
p_hat1500-3	1500	94	10321	10,321	84	10,321	0.1	10
san200_0.7_1	200	30	3370	3370	30	3370	0.1	10
san200_0.7_2	200	18	2422	2422	14	2422	0.1	10
san200_0.9_1	200	70	6825	6825	70	6825	0.1	10
san200_0.9_2	200	60	6082	6082	60	6082	0.1	10
san200_0.9_3	200	44	4748	4748	34	4748	0.1	10
san400_0.5_1	400	13	1455	1455	8	1455	0.1	10
san400_0.7_1	400	40	3941	3941	40	3941	3.4	10
san400_0.7_2	400	30	3110	3110	30	3110	1.4	10
san400_0.7_3	400	22	2771	2771	18	2771	0.2	10
san400_0.9_1	400	100	9776	9776	100	9776	14.3	64
san1000	1000	15	1716	1716	9	1716	0.1	10
sanr200-0.7	200	18	2325	2325	15	2325	0.1	10
sanr200-0.9	400	42	5126	5126	36	5126	0.1	10
sanr400-0.5	400	13	1835	1835	11	1835	0.1	10
sanr400-0.7	400	21	2992	2992	18	2992	0.1	10
Average							97.4	

3.1 Experimental setup and problem instances

Our experiments are performed on a high-performance cluster (HPC) computer, HP ProLiant DL585 G7, that has AMD Opteron 6212 CPU running at 2.6 GHz and having 8 cores. CPU has 64-bit computing capacity and AMD SR5690 chipset. The server uses 128 GB PC3-10600 RAM and 1.5 TB hard disk. The software comprises: a Scientific Linux v4.5 64-bit operating system, Open MPI v1.2.4, and C++. We have performed our experiments with 64 processors. We carry out our experiments with large graphs that fit in our main memory and do not use virtual memory. Because using virtual memory has a dramatic negative effect on the performance of the optimization process, we observe that it may cause thousands of times longer execution time due to the paging process of virtual memory. The selected large graph instances are run 10 times, and their best/average results and execution times are reported. The time to read the problem instance from the disk is not included in the execution time of the Par-LS algorithm.

The Par-LS algorithm is tested on 173 graphs from DIMACS benchmark (80 problem instances), the BHOSLIB benchmark (40 instances) and the Network Data Repository (53 instances). All the benchmark instances are originally unweighted. In order to provide the standard weighted instances, we use the literature and give each vertex i a weight given by (i *mod* 200) + 1 (Pullan 2008). In Table 4, we give the details of the large graphs that are used in our experi-

Table 3 Detailed results of the Par-LS algorithm on BHLOBS-W benchmark instances

Instance	Node	ω	W_best	Best	$\|C\|$	Avg-sum	Time (s)	#proc.
frb30-15-1	450	30	2990	2990	27	2990	2.1	64
frb30-15-2	450	30	3006	3006	28	3006	1.4	64
frb30-15-3	450	30	2995	2995	27	2995	1.6	64
frb30-15-4	450	30	3032	3032	28	3032	1.3	64
frb30-15-5	450	30	3011	3011	27	3011	0.1	64
frb35-17-1	595	35	3650	3650	33	3650	8.9	64
frb35-17-2	595	35	3738	3738	33	3738	11.5	64
frb35-17-3	595	35	3716	3716	33	3716	2.6	64
frb35-17-4	595	35	3683	3683	35	3683	5.7	64
frb35-17-5	595	35	3686	3686	33	3686	1.8	64
frb40-19-1	760	40	4063	4063	37	4063	17.8	64
frb40-19-2	760	40	4112	4112	36	4112	4.7	64
frb40-19-3	760	40	4115	4115	36	4115	16.8	64
frb40-19-4	760	40	4136	4136	37	4136	7.5	64
frb40-19-5	760	40	4118	4118	36	4118	3.6	64
frb45-21-1	945	45	4760	4760	41	4760	24.2	64
frb45-21-2	945	45	4784	4784	41	4784	15.7	64
frb45-21-3	945	45	4765	4765	43	4765	7.7	64
frb45-21-4	945	45	4799	4799	42	4799	2.9	64
frb45-21-5	945	45	4779	4779	43	4779	15.4	64
frb50-23-1	1150	50	5494	5494	47	5494	64.8	64
frb50-23-2	1150	50	5462	5462	47	5462	34.2	64
frb50-23-3	1150	50	5486	5486	47	5486	34.2	64
frb50-23-4	1150	50	5454	5453	46	5453	28.1	64
frb50-23-5	1150	50	5498	5498	47	5498	21.8	64
frb53-24-1	1272	53	5670	5670	50	5670	20.8	64
frb53-24-2	1272	53	5707	5707	48	5707	33.6	64
frb53-24-3	1272	53	5640	5655	49	5655	38.4	64
frb53-24-4	1272	53	5714	5714	48	5714	28.4	64
frb53-24-5	1272	53	5659	5659	49	5657.6	9.6	64
frb56-25-1	1400	56	5916	5916	53	5912.4	37.2	64
frb56-25-2	1400	56	5872	5872	52	5868.1	32.6	64
frb56-25-3	1400	56	5859	5859	51	5847.2	41.5	64
frb56-25-4	1400	56	5892	5892	51	5888.1	47.1	64
frb56-25-5	1400	56	5839	5839	51	5834.3	3.4	64
frb59-26-1	1534	59	6591	6591	55	6591	29.7	64
frb59-26-2	1534	59	6645	6645	56	6645	26.4	64
frb59-26-3	1534	59	6608	6608	56	6608	19.2	64
frb59-26-4	1534	59	6592	6592	54	6592	29.3	64
frb59-26-5	1534	59	6584	6584	53	6584	84.2	64
Average							20.4	

ments. The number of nodes, number of vertices, the density of the graph, the best and average results discovered by the Par-LS algorithm, execution time, and the number of processors used during the optimization are presented in the Table. At first, we try to solve the problem instances with five processors. These are the easier problem instances like *bio-dmela*, *bio-yeast*, and *ca-AstroPh*. At the next step, we apply 64 processors for harder problem instances that need more computation power. *hamming10-4*, *p-hat1000-2*, and *san1000* are the examples of harder large graphs in the problem set.

Table 4 The properties of the large graph instances that are used during our experiments and the obtained solutions by the Par-LS algorithm

Instance name	$\|V\|$	$\|E\|$	edge density	best	average	time (s)	#processors
bio-dmela	7393	25,569	0.000936	805	805	199.1	5
bio-yeast	1458	1948	0.001834	629	629	0.4	5
C.1000.9	1000	450,079	0.901059	954	954	176.3	64
C.2000.5	2000	999,836	0.500168	2466	2466	1.7	64
C.2000.9	2000	1,799,532	0.900216	10,999	109,999	1600.2	64
C4000.5	4000	4000,268	0.500,159	2792	2792	30.4	64
ca-AstroPh	17,903	196,972	0.001229	5338	5338	95.0	5
ca-CondMat	21,363	91,286	0.000400	2887	2887	14.2	5
ca-CSphd	1882	1740	0.000983	489	489	4.32	5
ca-Erdos992	6100	7515	0.000404	958	958	4.6	5
ca-GrQc	4158	13,422	0.001553	4279	4279	4.3	5
ca-HepPh	11,204	117,619	0.001874	24,533	24,533	18.9	5
DSJC1000-5	1000	249,826	0.500152	2186	2186	1.4	64
frb100-40	4000	7,425,226	0.928385	**10,709**	10,681.4	2645.7	64
hamming10-2	1024	518,656	0.990225	50,512	50,512	5.2	64
hamming10-4	1024	518,656	0.990225	5129	5129	6.4	64
ia-email-EU	32,430	54,397	0.000103	1350	1350	153.4	5
ia-email-univ	1133	5451	0.008500	1473	1473	1.8	5
ia-enron-large	33,696	180,811	0.000319	2490	2490	152.3	5
ia-fb-messages	1266	6451	0.008056	791	791	1.8	5
ia-reality	6809	7680	0.000331	374	374	7.4	5
inf-power	4941	6594	0.000540	888	888	8.0	5
keller6	3361	4,619,898	0.818191	8062	8062	58.4	64
MANN-a45	1035	533,115	0.996300	34,265	34,265	2445.2	64
MANN-a81	3321	5,506,380	0.998825	111,400	111,400	3401.8	64
p-hat1000-1	1000	122,253	0.244751	1514	1514	2.3	64
p-hat1000-2	1000	244,799	0.490088	5777	5.777	2.4	64
p-hat1000-3	1000	371,746	0.744236	8111	8111	1.9	64
p-hat1500-1	1500	284,923	0.253434	1619	1619	2.3	64
p-hat1500-2	1500	568,960	0.506080	7360	7360	2.1	64
p-hat1500-3	1500	847,244	0.753608	10,321	10,321	1.9	64
san1000	1000	250,500	0.501502	1716	1716	1.8	64
sc-nasasrb	54,870	1,311,227	0.000871	4548	4548	620.2	5
soc-brightkite	56,739	212,945	0.000132	3672	3672	728.1	5
soc-buzznet	101,163	2,763,066	0.000540	2981	2981	6.4	5
soc-epinions	26,588	100,120	0.000283	1657	1657	111.7	5
socfb-Berkeley13	22,900	852,419	0.003251	4906	4906	86.8	5
socfb-CMU	6621	249,959	0.011406	4141	4141	8.7	5
socfb-Duke14	9885	506,437	0.010367	3694	3694	64.6	5
socfb-Indiana	29,732	1,305,757	0.002954	5412	5412	242.9	5
socfb-MIT	6402	251,230	0.012261	3658	3658	10.9	5
socfb-OR	63,392	816,886	0.000407	3523	3523	88.7	5
socfb-Penn94	41,536	1,362,220	0.001579	4738	4738	485.9	5
socfb-Stanford3	11,586	568,309	0.008468	5769	5769	20.4	5
socfb-Texas84	36,364	1,590,651	0.002406	5546	5546	695.7	5
socfb-UCLA	20,453	747,604	0.003574	5595	5595	98.8	5

Table 4 continued

Instance name	$\|V\|$	$\|E\|$	edge density	best	average	time (s)	#processors
socfb-Uconn	17,206	604,867	0.004087	5733	5733	41.4	5
socfb-UCSB37	14,917	482,215	0.004334	5669	5669	29.6	5
socfb-UF	35,111	1,465,654	0.002378	6043	6043	666.0	5
socfb-Uillinois	30,795	1,264,421	0.002667	5730	5730	215.7	5
socfb-Wisconsin87	23,831	835,946	0.002944	4239	4239	117.9	5
tech-as-caida2007	26,475	53,381	0.000152	1869	1869	87.1	5
tech-internet-as	40,164	85,123	0.000106	1692	1692	219.8	5
tech-p2p-gnutella	62,561	147,878	0.000076	703	703	609.6	5
tech-routers-rf	2113	6632	0.002972	1460	1460	1.2	5
tech-WHOIS	7476	56,943	0.002038	6154	6154	35.3	5
web-edu	3031	6474	0.001410	2077	2077	1.0	5
web-google	1299	2773	0.003289	1749	1749	0.5	5
web-indochina-2004	11,358	47,606	0.000738	6997	6997	33.0	5
web-spam	4767	37,375	0.003290	2503	2503	3.4	5
web-webbase-2001	16, 062	25,593	0.000198	3574	3574	11.8	5

A new best solution is discovered for the largest problem instance of the BHOSLIB benchmark, *frb100-40*

3.2 The effect of iterations and increasing the number of processors

In Fig. 4, we give the effect of the iterations for the Par-LS algorithm. The experiments are run with the *MANN-a45* problem instance from BHOSLIB benchmark library. The figure presents the obtained results with increasing number of iterations. Increasing the number of iterations has a positive effect on the optimization process of the Par-LS algorithm. 1,800,000 iterations are used during the optimization process of all problem instances. Most of the time, the Par-LS algorithm was able to obtain the optimal/best results at earlier iterations. The reported execution time of the Par-LS algorithm is the average execution time of the ten runs that the optimal/best results are discovered. The improvement of the optimization is observed to be 0.6% in the average when the number of iterations is increased from 1 to 100,000. Although there is an improvement in the quality of the solutions, the robustness of the Par- LS algorithm is also affected significantly, which means that larger number of iterations can guarantee the same results with less deviation than lower number of iterations.

We analyze the effect of increasing the number of processors for the Par-LS algorithm. Table 1 gives the effect of increasing the number of processors for the *MANN-a45* problem instance. The number of processors is 1 with the initial tests, and we double this value by two with every new experiment up to 128 processors. The positive effect of the increasing number of processors is observed during the experiments. 0.4% improvement is obtained. It can be seen that optimizations performed with larger number of processors are less prone to stagnation. As the number of processors is increased, better results are observed at earlier phases of the optimization.

3.3 Experiments with *DIMACS-W* and *BHLOBS-W* benchmark instances

In this part of our experiments, we carry out some experiments with the well-known problem instances from the graph libraries DIMACS-W and BHLOBS-W. Although most of the problem instances of these benchmarks are not large graphs, obtaining the performance results of the PAR-LS algorithm will give valuable evaluations about the robustness, scalability, and speed-up performance of the algorithm. The size of the maximum vertex weight clique size, execution time of the algorithm, and obtained best results are reported for the problem instances. At the last column of Tables 2 and 3, we report the number of the processors that we use during the optimization process. For easier problem instances, we try to minimize the number of processors while we are using 64 processors for harder problems.

During the experiments performed with DIMACS-W problem instances, all the problem instances are observed to be solved optimally with respect to the reported optimal/best results. The average execution time of the algorithm is 79.4 s. These results outperform those of state-of-the-art algorithms in the literature. Except the *frb50-23-4* problem instance (the optimal result is reported to be 5454, we discover 5453) from BHOSLIB library, all the problems are solved optimally by finding the best/optimal results. For the problem instance *frb53-24-3*, the BHLOBS-W library reports 5640

Table 5 Selected **#vertices** for the reported best resulting maximum vertex weight cliques. **weight** is the total sum of the weights of the nodes. A new best solution is reported for the instance *frb100-40* with weight 10,709

Instance name	Weight	#vertices	Selected vertices
C2000-9	10,999	72	138, 141, 155, 177, 178, 244, 287, 323, 328, 385, 387, 390, 394, 396, 439, 511, 526, 561, 566, 578, 625, 669, 749, 757, 765, 768, 770, 780, 791, 796, 938, 942, 970, 976, 987, 997, 1120, 1133, 1152, 1155, 1168, 1181, 1184, 1186, 1196, 1353, 1363, 1385, 1429, 1457, 1527, 1533, 1548, 1572, 1586, 1591, 1595, 1599, 1695, 1727, 1732, 1733, 1764, 1784, 1794, 1797, 1917, 1975, 1977, 1980, 1986, 1997
ca-CondMat	2,887	26	846, 1345, 2331, 2561, 2861, 3187, 3511, 4789, 6763, 6826, 7176, 7185, 7278, 8045, 8380, 10312, 10455, 13145, 14213, 14841, 15464, 15699, 15850, 17707, 17873, 17992
ca-GrQc	4279	44	5, 97, 117, 250, 350, 436, 470, 529, 673, 739, 1002, 1064, 1103, 1266, 1419, 1553, 1759, 1783, 1923, 1942, 1994, 2004, 2211, 2250, 2276, 2386, 2753, 2759, 2984, 3074, 3174, 3206, 3283, 3297, 3347, 3387, 3418, 3487, 3613, 3653, 3714, 3951, 4011, 4079
frb100-40	**10,709**	**89**	26, 69, 108, 155, 192, 277, 286, 354, 362, 442, 519, 531, 595, 634, 643, 719, 740, 797, 837, 867, 906, 958, 996, 1063, 1118, 1156, 1174, 1277, 1319, 1355, 1397, 1428, 1468, 1515, 1553, 1594, 1672, 1717, 1757, 1792, 1838, 1875, 1918, 1948, 1976, 2069, 2114, 2157, 2189, 2233, 2264, 2313, 2339, 2396, 2438, 2474, 2503, 2557, 2587, 2621, 2657, 2715, 2752, 2793, 2838, 2913, 2950, 2987, 3076, 3113, 3127, 3195, 3277, 3319, 3356, 3388, 3504, 3558, 3590, 3631, 3660, 3713, 3751, 3796, 3809, 3857, 3911, 3951, 3967
keller6	8062	56	93, 157, 250, 257, 278, 292, 300, 366, 373, 393, 432, 514, 684, 692, 706, 723, 777, 778, 791, 797, 994, 1172, 1180, 1182, 1388, 1506, 1515, 1597, 1706, 1708, 1716, 1748, 1986, 1994, 1996, 2154, 2156, 2164, 2281, 2288, 2308, 2388, 2482, 2509, 2583, 2587, 2592, 2594, 2785, 2790, 2843, 2960, 2987, 3145, 3354, 3359
sc-nasasrb	4,548	24	48186, 48187, 48188, 48189, 48190, 48191, 48192, 48193, 48194, 48195, 48196, 48197, 48978, 48979, 48980, 48981, 48982, 48983, 48984, 48985, 48986, 48987, 48988, 48989
soc-brightkite	3,672	37	250, 2809, 2810, 2813, 2817, 2828, 2838, 2856, 2898, 2913, 6885, 6901, 6903, 6906, 10314, 10320, 10329, 10331, 10343, 10363, 10571, 10574, 10576, 10580, 10584, 10586, 10587, 10593, 10598, 10601, 10603, 10608, 10627, 10631, 10674, 10683, 10695
soc-buzznet	2,981	21	504, 731, 791, 1997, 2577, 2585, 2586, 2592, 2594, 2603, 2714, 2719, 2896, 28143, 30979, 31699, 82695, 82890, 83197, 99278, 99369
socfb-Berkeley13	4,906	41	381, 548, 594, 597, 797, 1060, 1125, 1322, 1491, 1499, 1580, 1749, 2139, 2685, 2714, 3148, 3259, 3570, 4686, 4894, 5009, 5781, 8133, 8255, 8562, 9699, 10757, 10796, 11169, 11245, 14303, 14682, 14993, 17043, 17544, 18271, 20408, 21731, 22073, 22332, 22600
socfb-Texas84	5,546	54	1861, 1999, 3097, 3210, 3256, 3787, 3790, 4572, 4680, 4982, 5124, 5330, 5738, 5751, 6476, 7772, 7811, 7864, 8456, 9327, 10715, 10750, 15251, 15724, 15988, 16229, 16398, 16513, 16830, 18155, 22854, 23370, 23435, 23471, 25951, 26381, 26936, 27463, 27561, 27613, 28055, 28166, 28689, 28971, 29121, 29328, 29714, 31240, 32576, 34315
socfb-UIllinois	5,730	50	797, 1169, 1960, 3290, 3929, 4656, 4722, 6495, 7141, 7991, 8049, 8182, 10240, 10403, 10587, 10650, 11190, 12159, 12385, 12667, 12801, 12888, 13895, 15789, 17270, 17393, 18287, 18670, 18904, 19546, 19778, 20499, 20522, 20534, 20726, 20985, 21729, 21977, 22443, 22446, 23324, 23623, 23638, 24460, 24786, 25032, 25531, 29112, 29537, 30643
tech-WHOIS	6,154	56	271, 364, 385, 454, 455, 472, 498, 543, 547, 702, 707, 741, 771, 773, 814, 844, 963, 1086, 1319, 1341, 1438, 1467, 1744, 1843, 1854, 2054, 2131, 2262, 2453, 2479, 2522, 2644, 2739, 2996, 3127, 3157, 3180, 3787, 3819, 3929, 3941, 3966, 3995, 4072, 4176, 4271, 4570, 4636, 4801, 4928, 4985, 5209, 5511, 5939, 6040, 6127
web-indochina-2004	6,997	40	7358, 7359, 7360, 7361, 7362, 7363, 7364, 7365, 7366, 7367, 7368, 7369, 7370, 7371, 7372, 7373, 7374, 7375, 7376, 7377, 7378, 7379, 7380, 7381, 7382, 7383, 7384, 7385, 7386, 7387, 7388, 7389, 7390, 7391, 7392, 7393, 7394, 7395, 7396, 7414

as the optimal solution. We obtain 5665 value for the same problem instance during our experiments, which has been only reported by ILS-VND algorithm so far. The average execution time of the algorithm is reported to be 20.4 s for the problems in BHLOBS-W library. It is interesting that we discover some maximum vertex size clique sizes that are smaller than the maximum clique of the optimized graph. It shows that the size of the maximum vertex weight clique can be smaller than maximum clique in the graph but the weight of the vertices can be larger. Although ILS-VND algorithm obtains most of the optimal/best results reported in the benchmark libraries, Par-LS is better in the average results and also there are new best results that have been reported by the Par-LS algorithm.

We compare our results with state-of-the-art heuristic algorithms in the literature for the DIMACS-W and

Table 6 Selected vertices for the maximum vertex weight of the MANN-a45 problem instance with weight 34,265

14, 16, 19, 21, 22, 23, 24, 27, 28, 30, 32, 38, 40, 41, 42, 47, 50, 53, 56, 59, 62, 65, 68, 71, 74, 77, 80, 83, 86,
88, 92, 95, 98, 101, 103, 107, 109, 112, 116, 119, 121, 125, 128, 130, 134, 137, 140, 143, 146, 148, 152, 153, 156,
159, 163, 167, 169, 172, 174, 178, 182, 185, 186, 189, 194, 197, 198, 209, 210, 215, 218, 221, 223, 227, 229, 232,
236, 238, 242, 245, 246, 249, 254, 257, 260, 263, 265, 269, 272, 275, 278, 280, 284, 285, 289, 292, 296, 299, 301,
304, 306, 311, 314, 317, 320, 323, 324, 327, 332, 333, 338, 341, 344, 346, 349, 352, 356, 359, 362, 365, 368, 369,
374, 377, 380, 383, 384, 389, 390, 393, 396, 401, 404, 405, 408, 411, 416, 419, 422, 425, 428, 430, 433, 437, 439,
443, 446, 449, 451, 453, 456, 461, 464, 467, 470, 473, 476, 477, 482, 483, 488, 491, 494, 496, 500, 503, 506, 509,
512, 515, 516, 521, 523, 525, 530, 532, 535, 537, 541, 543, 546, 550, 552, 557, 560, 563, 564, 569, 572, 575, 578,
580, 584, 587, 590, 593, 595, 598, 600, 603, 607, 611, 614, 615, 619, 621, 626, 629, 632, 635, 638, 640, 642, 647,
648, 653, 656, 659, 661, 663, 667, 671, 674, 676, 680, 681, 684, 689, 692, 693, 698, 699, 702, 707, 709, 713, 716,
719, 722, 725, 728, 731, 734, 735, 740, 742, 746, 749, 751, 755, 756, 761, 764, 766, 770, 771, 776, 779, 782, 785,
788, 791, 792, 797, 798, 803, 806, 809, 810, 815, 818, 821, 824, 827, 830, 831, 836, 838, 841, 845, 847, 850, 852,
856, 858, 862, 865, 867, 872, 875, 878, 880, 884, 886, 890, 891, 894, 899, 902, 904, 908, 909, 913, 917, 918, 923,
926, 929, 932, 935, 938, 941, 944, 945, 950, 952, 956, 959, 961, 965, 967, 971, 974, 976, 980, 981, 986, 989, 991,
995, 997, 1001, 1004, 1007, 1010, 1013, 1014, 1019, 1021, 1024, 1026, 1031, 1034

Table 7 Selected vertices for the maximum vertex weight of the *MANN-a81* problem instance with weight 111,400

13, 26, 32, 37, 41, 43, 44, 48, 49, 52, 53, 61, 67, 68, 69, 71, 74, 76, 77, 79, 83, 86, 89, 92, 95, 98, 101, 104,
107, 110, 113, 116, 119, 122, 125, 128, 131, 134, 137, 140, 143, 146, 149, 152, 155, 158, 160, 163, 167, 169, 173,
176, 179, 182, 185, 187, 191, 194, 196, 198, 203, 205, 209, 212, 215, 218, 221, 223, 227, 230, 233, 236, 239, 242,
245, 248, 251, 254, 257, 260, 263, 266, 268, 271, 275, 277, 281, 284, 287, 290, 293, 295, 299, 302, 305, 308, 311,
314, 317, 320, 323, 326, 329, 332, 335, 338, 341, 344, 347, 350, 353, 356, 359, 362, 365, 368, 371, 374, 377, 380,
383, 386, 389, 392, 395, 398, 401, 404, 407, 410, 413, 416, 419, 422, 425, 428, 430, 434, 436, 440, 443, 446, 449,
452, 455, 458, 461, 464, 467, 470, 472, 474, 478, 480, 484, 488, 490, 494, 497, 499, 501, 506, 509, 510, 513, 517,
519, 524, 525, 530, 533, 534, 538, 542, 545, 546, 550, 552, 556, 560, 562, 566, 569, 571, 573, 578, 579, 582, 585,
590, 591, 596, 597, 601, 604, 606, 611, 614, 617, 619, 622, 625, 627, 632, 634, 638, 641, 642, 646, 650, 651, 656,
659, 662, 665, 668, 671, 674, 677, 680, 683, 686, 688, 691, 694, 698, 699, 704, 706, 709, 712, 714, 719, 722, 724,
728, 731, 734, 737, 740, 743, 746, 749, 752, 755, 758, 761, 764, 766, 769, 773, 776, 777, 780, 785, 788, 791, 794,
796, 798, 803, 805, 809, 812, 814, 818, 821, 822, 826, 828, 833, 835, 837, 842, 843, 847, 851, 852, 856, 858, 863,
866, 867, 872, 875, 877, 881, 884, 885, 890, 892, 894, 899, 902, 903, 908, 911, 913, 917, 920, 921, 926, 929, 930,
933, 936, 940, 944, 945, 950, 951, 956, 958, 960, 964, 967, 971, 974, 976, 980, 983, 985, 989, 992, 994, 998, 1001,
1004, 1007, 1010, 1013, 1016, 1019, 1021, 1023, 1028, 1031, 1032, 1037, 1039, 1041, 1045, 1048, 1052, 1054, 1058,
1060, 1064, 1066, 1069, 1073, 1076, 1079, 1082, 1085, 1088, 1091, 1093, 1095, 1100, 1103, 1104, 1109, 1112, 1114,
1118, 1119, 1122, 1127, 1129, 1133, 1136, 1139, 1142, 1145, 1146, 1151, 1154, 1156, 1160, 1163, 1165, 1169, 1172,
1174, 1176, 1180, 1184, 1186, 1188, 1193, 1195, 1198, 1202, 1203, 1207, 1209, 1214, 1217, 1219, 1223, 1226, 1227,
1232, 1235, 1236, 1241, 1242, 1246, 1250, 1253, 1254, 1259, 1262, 1264, 1268, 1271, 1273, 1277, 1280, 1281, 1284,
1287, 1290, 1295, 1297, 1301, 1303, 1307, 1308, 1311, 1315, 1318, 1322, 1325, 1327, 1331, 1334, 1336, 1340, 1343,
1344, 1349, 1352, 1355, 1358, 1361, 1364, 1367, 1370, 1371, 1375, 1379, 1382, 1383, 1388, 1390, 1393, 1396, 1398,
1403, 1404, 1409, 1411, 1415, 1416, 1420, 1424, 1427, 1430, 1433, 1436, 1439, 1442, 1444, 1447, 1451, 1454, 1455,
1460, 1463, 1465, 1469, 1471, 1473, 1478, 1479, 1484, 1487, 1490, 1492, 1496, 1499, 1502, 1505, 1508, 1511, 1514,
1517, 1520, 1523, 1524, 1529, 1530, 1533, 1538, 1540, 1544, 1546, 1550, 1551, 1554, 1559, 1561, 1563, 1566, 1569,
1574, 1576, 1580, 1582, 1586, 1587, 1590, 1595, 1597, 1601, 1602, 1605, 1610, 1612, 1616, 1618, 1622, 1623, 1626,
1631, 1632, 1635, 1638, 1643, 1644, 1648, 1652, 1653, 1656, 1661, 1663, 1667, 1670, 1671, 1676, 1679, 1682, 1685,
1688, 1691, 1694, 1697, 1700, 1703, 1704, 1708, 1711, 1715, 1717, 1720, 1724, 1726, 1729, 1733, 1735, 1739, 1742,
1743, 1748, 1751, 1754, 1757, 1760, 1763, 1766, 1769, 1772, 1775, 1776, 1781, 1783, 1786, 1790, 1792, 1796, 1798,
1802, 1804, 1807, 1811, 1814, 1817, 1820, 1821, 1825, 1828, 1832, 1833, 1838, 1841, 1844, 1847, 1849, 1853, 1856,
1858, 1862, 1865, 1867, 1871, 1874, 1876, 1879, 1882, 1886, 1887, 1890, 1895, 1897, 1900, 1904, 1906, 1909, 1911,

Table 7 continued

1916, 1919, 1921, 1925, 1928, 1930, 1934, 1937, 1939, 1943, 1945, 1948, 1952, 1955, 1957, 1961, 1964, 1966, 1970, 1973, 1975, 1979, 1982, 1984, 1987, 1990, 1993, 1997, 1998, 2003, 2005, 2009, 2011, 2014, 2017, 2020, 2024, 2027, 2028, 2033, 2036, 2038, 2042, 2045, 2047, 2051, 2054, 2057, 2060, 2063, 2066, 2069, 2072, 2074, 2077, 2081, 2084, 2085, 2090, 2091, 2095, 2098, 2101, 2105, 2107, 2111, 2113, 2117, 2119, 2122, 2126, 2129, 2132, 2135, 2138, 2141, 2144, 2146, 2149, 2153, 2156, 2158, 2162, 2165, 2167, 2171, 2173, 2175, 2180, 2182, 2186, 2189, 2192, 2195, 2198, 2201, 2204, 2207, 2210, 2213, 2216, 2219, 2222, 2225, 2228, 2229, 2234, 2237, 2240, 2243, 2246, 2249, 2252, 2255, 2258, 2261, 2264, 2267, 2270, 2273, 2276, 2279, 2280, 2284, 2288, 2291, 2292, 2297, 2300, 2303, 2306, 2309, 2312, 2315, 2318, 2321, 2324, 2327, 2330, 2333, 2336, 2339, 2342, 2345, 2348, 2351, 2354, 2357, 2360, 2363, 2366, 2369, 2372, 2373, 2378, 2381, 2384, 2387, 2390, 2393, 2396, 2397, 2402, 2405, 2408, 2410, 2414, 2417, 2420, 2423, 2426, 2429, 2432, 2435, 2438, 2441, 2444, 2447, 2450, 2452, 2456, 2459, 2460, 2465, 2468, 2469, 2474, 2477, 2480, 2483, 2486, 2488, 2492, 2495, 2497, 2501, 2504, 2506, 2510, 2513, 2516, 2519, 2522, 2525, 2528, 2531, 2532, 2537, 2540, 2543, 2545, 2549, 2552, 2555, 2558, 2561, 2564, 2567, 2570, 2573, 2576, 2578, 2582, 2584, 2587, 2591, 2592, 2597, 2598, 2603, 2604, 2608, 2612, 2613, 2617, 2620, 2623, 2627, 2629, 2633, 2635, 2639, 2641, 2644, 2648, 2650, 2654, 2656, 2659, 2663, 2665, 2669, 2671, 2675, 2677, 2680, 2684, 2686, 2689, 2692, 2696, 2698, 2701, 2705, 2707, 2710, 2714, 2716, 2720, 2723, 2725, 2729, 2732, 2735, 2738, 2741, 2744, 2747, 2750, 2753, 2756, 2758, 2761, 2764, 2768, 2770, 2772, 2777, 2779, 2782, 2786, 2787, 2792, 2795, 2797, 2801, 2804, 2807, 2810, 2813, 2816, 2819, 2822, 2825, 2828, 2829, 2834, 2836, 2839, 2843, 2845, 2849, 2851, 2855, 2857, 2860, 2864, 2867, 2870, 2873, 2875, 2877, 2881, 2885, 2887, 2891, 2894, 2897, 2900, 2903, 2906, 2909, 2912, 2915, 2918, 2921, 2924, 2927, 2930, 2931, 2936, 2939, 2942, 2945, 2948, 2951, 2954, 2957, 2960, 2963, 2966, 2969, 2972, 2975, 2978, 2981, 2982, 2985, 2990, 2993, 2995, 2997, 3002, 3005, 3008, 3011, 3014, 3017, 3020, 3023, 3026, 3029, 3032, 3035, 3038, 3041, 3044, 3047, 3050, 3053, 3056, 3059, 3062, 3065, 3068, 3071, 3074, 3076, 3080, 3083, 3086, 3089, 3092, 3095, 3098, 3101, 3104, 3107, 3110, 3111, 3116, 3119, 3122, 3125, 3128, 3131, 3134, 3137, 3140, 3143, 3146, 3149, 3152, 3154, 3158, 3161, 3163, 3167, 3170, 3171, 3176, 3179, 3182, 3185, 3188, 3189, 3194, 3197, 3198, 3203, 3206, 3208, 3212, 3215, 3218, 3221, 3224, 3227, 3230, 3233, 3234, 3239, 3242, 3245, 3248, 3251, 3254, 3257, 3260, 3262, 3266, 3269, 3272, 3275, 3278, 3280, 3282, 3286, 3290, 3291, 3296, 3299, 3301, 3305, 3306, 3310, 3314, 3315, 3320

BHLOBS-W problem instances: phased local search (PLS) (Pullan 2008), multi-neighborhood tabu search (MN/TS) (Wu et al. 2012), breakout local search (BLS) (Benlic and Hao 2013), ReTS-I (Zhou et al. 2017b), iterated local search variable neighborhood descent (ILS-VND) (Nogueira et al. 2017), and BQP problem with the probabilistic tabu search algorithm (BQP-PTS) (Alidaee et al. 2007). For the other 119 instances, the Par-LS algorithm provides better or the same results that have been found by the other algorithms. Even for the hard problem instances, *MANN-a45* and *MANN-a81*, the Par-LS performs better than the other algorithms. Detailed performance comparison of the Par-LS algorithm is presented in Table 8 for large graph problem instances.

3.4 Performance analysis of the Par-LS algorithm on large graphs

Experiments are carried out with 61 different large graph instances from DIMACS, BHOSLIB, and Network Data Repository. The number of vertices, edges and the density of the edges of the graphs is presented in Table 4. We report two weight results (the best and the average). The best results are the maximum values obtained during experiments. Average results are the mean of ten different executions. First, we try to solve the problem instances with five processors. If we cannot get good performance, then we increase our number of processors up to 64. Tables 5, 6, and 7 report the selected vertices of the maximum vertex weight cliques for some of the large graphs. A new best solution is reported for the largest graph instance of BHOSLIB benchmark (*frb100-40*). All of the other results are the best/optimal results that have been reported by researchers up to now. The average of the execution time is observed as 268.9 s. The simple parameter setting mechanism of the Par-LS algorithm is observed to provide better results. We carried out a set of experiments on the *MANN-a81* problem instance. It was possible to obtain the maximum vertex weight value as 111,400 when we use different parameters for each processor. With 10 different runs, it was not possible to get the value, 111,400, when the simple parameter setting mechanism is not used.

3.5 Comparison with state-of-the-art algorithms

The Par-LS algorithm is compared with state-of-the-art large MVWCP algorithms for large graphs. LSCC (Wang et al. 2016b), MN/TS (Wang et al. 2016b), ReTS-I (Zhou et al.

Table 8 Comparison with state-of-the-art algorithms

Instance	LSCC+BMS	FastWClq	LSCC		MN/TS		ReTS-I		GPULS(CPU)-R		Par-LS	
	Best (Avg.)	Best (Avg.[a])	Best (Avg.)	*t* (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)
bio-dmela	805	805	805 (805)	0.00	805 (805)	0.00	805 (805)	0.00	805 (805)	0.00	805 (805)	199.1
bio-yeast	629	629[a]	629 (629)	0.00	629 (629)	2.25	629 (629)	0.06	629 (629)	0.13	629 (629)	0.4
C.1000.9	–	–	9254 (9214.8)	78.41	9254 (9254)	32.80	9254 (9254)	2.28	9254 (9254)	4.28	9254 (9254)	176.3
C2000.5	–	–	2466 (2466)	1.49	2466 (2466)	0.96	2466 (2466)	0.40	2466 (2466)	0.66	2466 (2466)	1.7
C2000.9	–	–	10,999 (10,839.5)	99.33	10,964 (10,910.5)	100.00	10,999 (10,981.5)	74.91	10,999 (10,915.3)	99.39	**10,999 (10,999)**	1600.2
C4000.5	–	–	2792 (2790.4)	44.95	2792 (2788.4)	53.50	2792 (2792)	25.12	2792 (2791.6)	40.70	2792 (2792)	30.4
ca-AstroPh	5338	5338[a]	5338 (5338)	11.01	5338 (5336.6)	25.70	5338 (5338)	18.49	5338 (5338)	5.53	5338 (5338)	95.0
ca-CondMat	2887	2887[a]	2887 (2887)	1.14	2887 (2831.9)	37.46	2887 (2887)	1.49	2887 (2887)	6.63	2887 (2887)	14.2
ca-CSphd	–	489[a]	489 (489)	0.00	489 (489)	9.02	489 (489)	0.01	489 (489)	0.53	489 (489)	4.32
ca-Erdos992	958	958[a]	958 (958)	0.00	958 (958)	0.25	958 (958)	0.01	958 (958)	0.03	958 (958)	4.6
ca-GrQc	4279	4279[a]	4279 (4279)	0.04	4279 (4279)	0.25	4279 (4279)	0.01	4279 (4279)	0.01	4279 (4279)	4.3
ca-HepPh	24,533	24,533[a]	24,533 (24,533)	0.03	24,533 (24,533)	0.23	24,533 (24,533)	0.02	24,533 (24,533)	0.03	24,533 (24,533)	18.9
DSJC1000-5	–	–	2186 (2186)	6.72	2186 (2186)	0.13	2186 (2186)	0.05	2186 (2186)	0.03	2186 (2186)	1.4
frb100-40	–	–	10,424 (10,321)	100.00	10,427 (10,253)	100.00	10,398 (10,255)	100.00	10,443 (10,339.3)	98.97	**10,709 (10,681.4)**	2645.7
hamming10-2	–	–	50,512 (50,512)	0.27	50,512 (50,512)	3.45	50,512 (50,512)	0.03	50,512 (50,512)	0.08	50,512 (50,512)	5.2
hamming10-4	–	–	5129 (5129)	12.72	5129 (5129)	9.10	5129 (5129)	3.17	5129 (5129)	2.63	5129 (5129)	6.4
ia-email-EU	1350	1350	1350 (1350)	0.12	1350 (1350)	0.15	1350 (1350)	0.02	1350 (1350)	0.03	1350 (1350)	153.4
ia-email-univ	1473	1473[a]	1473 (1473)	0.00	1473 (1473)	0.06	1473 (1473)	0.00	1473 (1473)	0.00	1473 (1473)	1.8
ia-enron-large	2490	2490	2490 (2490)	3.48	2490 (2490)	0.86	2490 (2490)	0.08	2490 (2490)	0.06	2490 (2490)	152.3
ia-fb-messages	791	791	791 (791)	0.00	791 (791)	0.00	791 (791)	0.00	791 (791)	0.00	791 (791)	1.8
ia-reality	374	374[a]	374 (374)	0.04	374 (368.875)	44.95	374 (374)	0.02	374 (374)	2.17	374 (374)	7.4
inf-power	888	888[a]	888 (888)	0.02	888 (888)	13.16	888 (888)	0.06	888 (888)	0.45	888 (888)	8.0
keller6	–	–	7861 (7629.88)	100.00	7861 (7628.18)	100.00	7895 (7625.45)	100.00	7968 (7797.77)	98.17	**8062 (8062)**	58.4
MANN-a45	–	–	34,249 (34,219)	99.39	34,171 (34,154)	100.00	34,184 (34,177)	100.00	34,197 (34,188)	100.00	**34,265 (34,265)**	2445.2
MANN-a81	–	–	111,077 (111,030)	100.00	111,069 (111,011)	100.00	111,115 (111,096)	100.00	111,150 (111,127)	97.95	**111,400 (111,400)**	3401.8
p-hat1000-1	–	–	1514 (1514)	0.91	1514 (1514)	0.02	1514 (1514)	0.02	1514 (1514)	0.08	1514 (1514)	2.3
p-hat1000-2	–	–	5777 (5777)	0.15	5777 (5777)	0.08	5777 (5777)	0.00	5777 (5777)	0.00	5777 (5777)	2.4
p-hat1000-3	–	–	8111 (8111)	3.25	8111 (8111)	2.43	8111 (8111)	0.01	8111 (8111)	0.08	8111 (8111)	1.9
p-hat1500-1	–	–	1619 (1619)	0.01	1619 (1619)	0.05	1619 (1619)	0.03	1619 (1619)	0.08	1619 (1619)	2.3
p-hat1500-2	–	–	7360 (7360)	1.33	7360 (7360)	0.86	7360 (7360)	0.14	7360 (7360)	0.04	7360 (7360)	2.1
p-hat1500-3	–	–	10,321 (10,303)	74.76	10,321 (10,290)	90.21	10,321 (10,321)	0.33	10,321 (10,321)	6.78	10,321 (10,321)	1.9

Table 8 continued

Instance	LSCC+BMS	FastWClq	LSCC		MN/TS		ReTS-I		GPULS(CPU)-R		Par-LS	
	Best (Avg.)	Best (Avg.[a])	Best (Avg.)	*t* (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)
san1000	–	–	1716 (1715.25)	37.22	1716 (1709.5)	75.55	1716 (1716)	2.58	1716 (1716)	21.02	1716 (1716)	1.8
sc-nasasrb	4548 (4540.8)	4548	4548 (4531.8)	73.10	4548 (4254.5)	98.42	4548 (4545)	41.47	4548 (4469.8)	85.35	**4548 (4548)**	620.2
soc-brightkite	3650 (3643.7)	3672	3672 (3645.15)	99.21	3672 (3598.2)	96.09	3672 (3412.5)	62.35	3672 (3655.53)	76.40	**3672 (3672)**	728.1
soc-buzznet	2981 (2980)	2981	2981 (2979.5)	50.33	2981 (2980.25)	44.63	2981 (2981)	8.07	2981 (2981)	0.74	2981 (2981)	6.4
soc-epinions	1657	1657	1657 (1657)	19.41	1657 (1654.4)	20.88	1657 (1646.6)	45.62	1657 (1657)	1.85	1657 (1657)	111.7
socfb-Berkeley13	4906 (4839.6)	4906	4906 (4906)	24.69	4906 (4855.85)	40.86	4906 (4906)	15.38	4906 (4906)	6.07	4906 (4906)	86.8
socfb-CMU	4141	4141	4141 (4141)	2.17	4141 (4141)	2.25	4141 (4141)	1.36	4141 (4141)	0.15	4141 (4141)	8.7
socfb-Duke14	3694	3694	3694 (3694)	2.99	3694 (3694)	9.61	3694 (3694)	1.51	3694 (3694)	0.31	3694 (3694)	64.6
socfb-Indiana	5412 (5274.8)	5412	5412 (5343)	44.05	5412 (5377.7)	44.97	5412 (5387.8)	34.03	5412 (5412)	8.03	5412 (5412)	242.6
socfb-MIT	3658	3658	3658 (3658)	1.79	3658 (3658)	2.85	3658 (3658)	2.76	3658 (3658)	0.27	3658 (3658)	10.9
socfb-OR	3523 (3459.8)	3523	3523 (3417.07)	74.11	3523 (3467.45)	65.98	3523 (3504.45)	53.92	3523 (3523)	13.94	3523 (3523)	88.7
socfb-Penn94	4738 (4668.6)	4738	4738 (4569)	46.17	4738 (4474.57)	78.12	4738 (4386.65)	74.62	4738 (4738)	10.77	4738 (4738)	485.9
socfb-Stanford3	5769	5769	5769 (5769)	9.08	5769 (5769)	12.46	5769 (5769)	8.98	5769 (5769)	2.59	5769 (5769)	20.4
socfb-Texas84	5546 (5545.2)	5546	5546 (5533.68)	36.62	5546 (5423.23)	93.15	5546 (5546)	19.84	5546 (5543.2)	56.13	5546 (5546)	695.7
socfb-UCLA	5595	5595	5595 (5595)	20.01	5595 (5587.25)	30.67	5595 (5579.5)	33.84	5595 (5595)	4.87	5595 (5595)	98.8
socfb-UConn	5733	5733	5733 (5733)	11.38	5733 (5733)	5.52	5733 (5733)	2.23	5733 (5733)	0.75	5733 (5733)	41.4
socfb-UCSB37	5669	5669	5669 (5669)	11.42	5669 (5669)	16.30	5669 (5669)	7.52	5669 (5669)	2.87	5669 (5669)	29.6
socfb-UF	6043	6043	6043 (6035.1)	34.95	6043 (5968.25)	89.15	6043 (6035.1)	43.17	6043 (6042.4)	28.27	**6043 (6043)**	666.0
socfb-UIllinois	5730 (5685.6)	5730	5730 (5695.88)	54.73	5730 (5662.88)	94.86	5730 (5675.18)	66.90	5730 (5728.6)	63.28	5730 (5730)	215.7
socfb-Wisconsin87	4239	4239	4239 (4239)	27.15	4239 (4239)	19.83	4239 (4239)	10.95	4239 (4239)	2.35	4239 (4239)	117.9
tech-as-caida2007	1869	1869	1869 (1869)	0.06	1869 (1869)	0.04	1869 (1869)	0.00	1869 (1869)	0.00	1869 (1869)	87.1
tech-internet-as	1692	1692	1692 (1692)	0.54	1692 (1692)	0.07	1692 (1692)	0.02	1692 (1692)	0.05	1692 (1692)	219.8
tech-p2p-gnutella	703	703[a]	703 (703)	0.59	703 (619.575)	82.21	703 (590.45)	95.44	703 (686.7)	44.16	**703 (703)**	609.8
tech-routers-rf	1460	1460[a]	1460 (1460)	0.08	1460 (1460)	0.18	1460 (1460)	0.18	1460 (1460)	0.03	1460 (1460)	1.2
tech-WHOIS	6154	6154	6154 (6154)	0.27	6154 (6154)	9.90	6154 (6154)	0.11	6154 (6154)	1.14	6154 (6154)	35.3
web-edu	2077	2077[a]	2077 (2077)	1.81	2077 (2077)	1.94	2077 (1988.8)	45.83	2077 (2077)	0.38	2077 (2077)	1.0
web-google	1749	1749[a]	1749 (1749)	0.07	1749 (1749)	0.06	1749 (1749)	0.01	1749 (1749)	0.10	1749 (1749)	0.5
web-indochina-2004	45,477 (44373.5)	6997[a]	6997 (6997)	0.03	6997 (6997)	7.09	6997 (6997)	4.93	6997 (6997)	0.56	6997 (6997)	33.0
web-spam	2503	2503	2503 (2503)	6.75	2503 (2503)	6.69	2503 (2503)	6.23	2503 (2503)	0.77	2503 (2503)	3.4
web-webbase-2001	3574 (3339.4)	3574[a]	3574 (3251.43)	57.19	3574 (3222.1)	61.82	2401 (2401)	100.00	3574 (3574)	9.14	3574 (3574)	11.8

The best and the average of the results after ten execution of the instances are reported. Bold results (either the best or average values) represent the values that are discovered only by one of the algorithms. Underlined value reports the new best value found by the Par-LS algorithm

[a] means that optimal solutions are not found at every try

2017b), and GPULS (CPU)-R (Nogueira and Pinheiro 2018) are the selected best performing recent algorithms. Table 8 gives the details of comparison with the algorithms. LSCC, MN/TS, ReTS-I and GPULS(CPU)-R obtain 57, 56, 57, and 57 of the optimal results, respectively, whereas the Par-LS algorithm is able to find 61 optimal results. When the average value results are considered, the Par-LS algorithm outperforms the other four algorithms. Although the Par-LS algorithm spends more time for some of the problem instances, its execution time can be considered as reasonable when compared with the others. Parallel ant colony optimization-based meta-heuristic (PACOM) for solving the MVWCP is a high-performance algorithm (El Baz et al. 2016). The Par-LS performs better than the PACOM algorithm with all the instances. WLMC is a recent exact B&B algorithm that is reported to be very efficient for large graphs (Jiang et al. 2017). With its limited (3600 s) optimization process, Par-LS finds the same or better results than this algorithm. WLMC reports 111,139 for the *MANN-a81* instance, whereas the Par-LS reports 111,400.

In order to evaluate the performance of our algorithm, the available results of experiments for FastWClq (Cai and Lin 2016) and LSCC+BMS (Wang et al. 2016b) are added to Table 8. The execution time of the FastWClq is fast. It finds the solutions in a few seconds. Due to space limitation of the page, we are not able to give the details of each algorithm's execution time. The cutoff times for FastWClq and LSCC+BMS are 100 s. Some of the problems cannot be solved optimally by these two algorithms, whereas Par-LS is capable of finding optimal/best results for 172 of 173 problem instances. The execution time of Par-LS is worse than these two algorithms.

TSM-MWC (Jiang et al. 2018) and WC-MWC (Li et al. 2018) are exact algorithms that we can compare Par-LS algorithm to. In a recent study by Li et al. (2018), comprehensive experimental results can be observed for these exact algorithm. WC-MWC algorithm has the highest performance on DIMACS and BHOSLIB problem instances. The Par-LS algorithm again has the longest execution times (but still reasonable when compared with brute-force approaches) when compared with these algorithms. However, the performance of the Par-LS is higher than the other algorithms while finding optimal/best solutions.

3.6 Speed-up and scalability performance

Since the Par-LS algorithm is a kind of island parallel heuristic algorithm, it does not spend much time due to the dependent jobs that will be sent by the other processors. The speed-up of an algorithm is described as the ratio of the sequential execution of the algorithm for solving a problem to the time obtained by the parallel algorithm. The Par-LS algorithm provides nearly a linear speed-up. This is one of the best properties of this algorithm. For each processor, a different local search algorithm with different parameter settings and with a different clique selection is processed. With the increasing number of processors, the delay of the parallel algorithm is observed to be very small. Therefore, we can evaluate the Par-LS algorithm as a scalable parallel algorithm with an almost linear speed-up performance. Stagnation is a critical drawback of local search algorithms. A scalable and diversified parallel algorithm that performs a (near)-linear speed-up can provide good performance while dealing with the stagnation problem.

4 Conclusions and future work

In this study, we introduce a novel parallel local search algorithm for the solution of the MVWCP in large graphs. Single-processor computers have reached to their computation limitations due to the technological restrictions and power wall problem. Therefore, we believe that large NP-Hard problem instances will be solved better with parallel computation tools. Our experiments prove that we have obtained significantly improved results. We report a new best result for the largest problem instance of the BHOSLIB benchmark and better average maximum values for large graph instances. Absolutely, intelligent operators are still important means of optimization algorithms. We introduce new operators parallel(ω,1)-swap and parallel(1,2)-swap by using parallel computation techniques. The Par-LS algorithm is observed to be a scalable algorithm during the experiments. This means that increasing the number of processors will positively influence the optimization quality of the Par-LS algorithm. Stagnation is a common problem of the optimization algorithms. Parallel computation that starts each optimization process from a different starting point (vertex) and works with diversified vertices can be considered as a mechanism to prevent stagnation of local search optimization techniques. As future work, the performance of the Par-LS can be improved by using CUDA programming. Hyper-heuristics that make use of several heuristic approaches is a hot topic and it can also be applied to the MVWCP.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

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

Informed consent There is no individual participant included in the study.

References

Alidaee B, Glover F, Kochenberger G, Wang H (2007) Solving the maximum edge weight clique problem via unconstrained quadratic programming. Eur J Oper Res 181:592–597

Balas E, Yu CS (1986) Finding a maximum clique in an arbitrary graph. SIAM J Comput 15:1054–1068

Benlic U, Hao J-K (2013) Breakout local search for maximum clique problems. Comput Oper Res 40:192–206

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

Cantú-Paz E (1998) A survey of parallel genetic algorithms. Calculateurs paralleles, reseaux et systems repartis 10:141–171

Dijkhuizen G, Faigle U (1993) A cutting-plane approach to the edge-weighted maximal clique problem. Eur J Oper Res 69:121–130

Dokeroglu T (2015) Hybrid teaching-learning-based optimization algorithms for the quadratic assignment problem. Comput Ind Eng 85:86–101

Dokeroglu T, Mengusoglu E (2017) A self-adaptive and stagnation-aware breakout local search algorithm on the grid for the steiner tree problem with revenue, budget and hop constraints. Soft Comput 22:1–19

Dokeroglu T, Sevinc E, Cosar A (2019) Artificial bee colony optimization for the quadratic assignment problem. Appl Soft Comput 76:595–606

El Baz D, Hifi M, Wu L, Shi X (2016) A parallel ant colony optimization for the maximum-weight clique problem. In: IEEE international parallel and distributed processing symposium workshops, 2016, IEEE, pp 796–800

Hansen P, Mladenović N, Brimberg J, Pérez JAM (2010) Variable neighborhood search. In: Gendreau M, Potvin JY (eds) Handbook of metaheuristics. Springer, Berlin, pp 61–86

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

Jiang H, Li C-M, Liu Y, Manya F (2018) A two-stage maxsat reasoning approach for the maximum weight clique problem. In: AAAI

Kiziloz HE, Dokeroglu T (2018) A robust and cooperative parallel tabu search algorithm for the maximum vertex weight clique problem. Comput Ind Eng 118:54–66

Kucukyilmaz T, Kiziloz HE (2018) Cooperative parallel grouping genetic algorithm for the one-dimensional bin packing problem. Comput Ind Eng 125:157–170

Kumlander D (2004) A new exact algorithm for the maximum-weight clique problem based on a heuristic vertex-coloring and a backtrack search. In: Proceedings of 5th international conference on modelling, computation and optimization in information systems and management sciences, Citeseer, pp 202–208

Li C-M, Liu Y, Jiang H, Manyà F, Li Y (2018) A new upper bound for the maximum weight clique problem. Eur J Oper Res 270:66–77

Lourenço HR, Martin OC, Stützle T (2010) Iterated local search: framework and applications. In: Gendreau M, Potvin JY (eds) Handbook of metaheuristics. Springer, Berlin, pp 363–397

Ma T, Latecki L J (2012) Maximum weight cliques with mutex constraints for video object segmentation. In: IEEE Conference on computer vision and pattern recognition (CVPR), 2012 IEEE, pp 670–677

Mascia F, Cilia E, Brunato M, Passerini A (2010) Predicting structural and functional sites in proteins by searching for maximum-weight cliques. In: AAAI

Nogueira B, Pinheiro RG (2018) A CPU-GPU local search heuristic for the maximum weight clique problem on massive graphs. Comput Oper Res 90:232–248

Nogueira B, Pinheiro RG, Subramanian A (2017) A hybrid iterated local search heuristic for the maximum weight independent set problem. Optim Lett 12:1–17

Pullan W (2008) Approximating the maximum vertex/edge weighted clique using local search. J Heuristics 14:117–134

Pullan W, Hoos HH (2006) Dynamic local search for the maximum clique problem. J Artif Intell Res 25:159–185

Tepper M, Sapiro G (2013) Ants crawling to discover the community structure in networks. In: Iberoamerican congress on pattern recognition. Springer, Berlin, pp 552–559

Wang Y, Hao J-K, Glover F, Lü Z, Wu Q (2016a) Solving the maximum vertex weight clique problem via binary quadratic programming. J Comb Optim 32:531–549

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

Warren JS, Hicks IV (2006) Combinatorial branch-and-bound for the maximum weight independent set problem. Relatório Técnico, Texas A&M University, Citeseer 9:17

Wu Q, Hao J-K (2015a) Solving the winner determination problem via a weighted maximum clique heuristic. Exp Syst Appl 42:355–365

Wu Q, Hao J-K (2015b) A review on algorithms for maximum clique problems. Eur J Oper Res 242:693–709

Wu Q, Hao J-K (2016) A clique-based exact method for optimal winner determination in combinatorial auctions. Inf Sci 334:103–121

Wu Q, Hao J-K, Glover F (2012) Multi-neighborhood tabu search for the maximum weight clique problem. Ann Oper Res 196:611–634

Zheng C, Zhu Q, Sankoff D (2007) Removing noise and ambiguities from comparative maps in rearrangement analysis. IEEE/ACM Trans Comput Biol Bioinf 4:515–522

Zhian H, Sabaei M, Javan N T, Tavallaie O (2013) Increasing coding opportunities using maximum-weight clique. In: 5th computer science and electronic engineering conference (CEEC), 2013, IEEE, pp 168–173

Zhou Y, Hao J-K, Duval B (2017a) Opposition-based memetic search for the maximum diversity problem. IEEE Trans Evol Comput 21:731–745

Zhou Y, Hao J-K, Goëffon A (2017b) Push: a generalized operator for the maximum vertex weight clique problem. Eur J Oper Res 257:41–54