A robust and cooperative parallel tabu search algorithm for the maximum vertex weight clique problem

Abstract

The maximum vertex weight clique problem (MVWCP) is a challenging NP-Hard combinatorial optimization problem that searches for a clique with maximum total sum of vertices' weights. In this study, we propose a robust and cooperative parallel tabu search algorithm (PTC) for the MVWCP. Our proposed algorithm uses a dedicated tabu search algorithm with a multistart strategy for the diversification of search space on a parallel computation environment. An effective seeding mechanism is developed with respect to the rank of the processors to choose diversified starting points for better exploration areas. Classical *add*, *swap* and *drop* operators of tabu search are improved for parallel computation with a combined neighborhood approach. The PTC algorithm is evaluated on a set of 120 problem instances from DIMACS-W and BHOSLIB-W benchmarks. Computational results show that the PTC algorithm competes with state-of-the-art heuristic algorithms by reporting average best (optimal) result hit ratios up to 99.0%.

Keywords: Maximum clique; tabu search; parallel computation; cooperation; heuristics.

1. Introduction

The maximum clique problem (MCP) is to find a complete sub-graph with a maximum cardinality in a given general graph [1]. The MCP can be used to design and solve several problems in computer vision, pattern matching, image matching, economics, examination planning, financial networks, social network analysis, wireless network telecommunications, bioinformatics and chemoinformatics [2][3]. In addition to this, clique partitioning, max-min diversity, graph vertex coloring, sum coloring and optimal winner determination are some of the other important combinatorial optimization problems that are related with the MCP [4, 5]. The maximum vertex weight clique problem (MVWCP) is a generalization of the classical MCP. When the vertices of the graph are assigned value 1, MVWCP is equivalent to MCP that finds a maximum cardinality clique [6][7]. The decision version of MCP is NP-complete and the generalized MVWCP is at least as hard as MCP [8][9]. More formaly, given an undirected graph $G = (V, E)$ with vertex set $V = \{1,..., n\}$ and edge set $E \subseteq V \times V$. Let $w : V \rightarrow Z^+$ be a weighting function that assigns to each vertex $i \in V$ a positive integer. The MVWCP determines a clique of maximum weight.

Tabu Search is an efficient meta-heuristic with a local heuristic search procedure to explore a solution space and it has been successfully applied to the solution of several combinatorial optimization problems [10]. In this study, we enhance the capacity of a classical tabu search algorithm [7] with parallel computation and adapt classical operators *add*, *swap*, and *drop* operators to the parallel computation environment. Restarting mechanism of the proposed algorithm (PTC) has a significant improvement on the average success hit ratio of the algorithm. The PTC algorithm is aware of stagnation and restarts the optimization process from different initial cliques when it gets stuck into local optima. This procedure is coordinated in the parallel computation environment and slave nodes use their exploration areas during the optimization. Our proposed PTC algorithm shows a more robust behaviour than other heuristic algorithms. The parallel version of tabu search enhances the robustness property of the algorithm by obtaining up to 99% of the reported best (optimal) result hit ratios in the literature.

In Section 2, related studies for the state-of-the-art MVWCP are summarized. In Section 3, our proposed algorithm, PTC, is introduced. Section 4 gives the details for the performance evaluation of the experimental results and comparison of the PTC algorithm to the state-of-the-art algorithms on benchmark instances DIMAC-W and BHOSLIB-W. Concluding remarks and future work are provided in the last Section.

2. Related work

In this section, we give information about the state-of-the-art algorithms proposed for the solution of the MVWCP [6]. Several algorithms have been proposed for solving MVWCP for the last 20 years. These algorithms can be examined in two parts, exact and heuristic algorithms. Östergärd propose a branch-and-bound (B&B) algorithm in which the vertices are processed according to the order provided by a vertex coloring of the given graph [11]. Kumlander develop a backtrack tree search algorithm which relies on a heuristic coloring-based vertex order [12]. Warren & Hicks propose three B&B algorithms that use weighted clique covers to generate upper bounds and branching rules [13]. Wu & Hao develop an algorithm that uses new bounding and branching techniques with specific vertex coloring and sorting [4].

Heuristic methods are recent popular algorithms to obtain (near-)optimal solutions in practical computing times when the search space of the problem space is too large to be calculated by an exact algorithm. Some of the most important heuristic algorithms are listed below. Mannino & Stefanutti propose a tabu search algorithm based on edge projection and augmenting sequence [14]. Fang et al. present an algorithm that uses Maximum Satisfiability Reasoning as a bounding technique [15]. Bomze et al. design MVWCP as a continuous problem that is solved by a parallel algorithm using a distributed computational network model [23]. Busygin proposes a trust region algorithm [25]. Singh & Gupta introduce a hybrid method combining a genetic algorithm [17]. A partially enumerative algorithm is presented for the maximum clique problem which is very simple to implement by Carraghan & Pardalos [16]. Pullan & Hoos develop the Phase Local Search (PLS) algorithm for the classical MCP to MVWCP [18]. Wu et al. develop a tabu search algorithm by integrating multiple neighborhoods [7]. Benlic & Hao present the Breakout Local Search algorithm that explores multiple neighborhoods and applies both directed and random perturbations [19]. The general binary quadratic programming (BQP) model has been widely applied to solve a number of combinatorial optimization problems. Wang et al. recast the MVWCP into a model which is solved by a probabilistic tabu search algorithm designed for the BQP [20].

Grosso et al. formulate and analyze iterated local search algorithms for the MCP. The basic components of such algorithms are fast neighbourhood search, diversification techniques and restart rules [21]. Zhou et al. introduce a generalized move operator called PUSH that generalizes the conventional ADD and SWAP operators commonly used in the literature and can be integrated in a local search algorithm for MVWCP [22]. Nogueira et al. present a hybrid iterated local search (ILS) algorithm for the maximum weight independent set (MWIS) problem. MWIS is a problem that is closely related to MVWCP. In their study, two new neighborhood structures are introduced and they are explored using the variable neighborhood descent procedure. The results show that the hybrid ILS is capable of finding all known optimal solutions [24]. Malladi et al. introduce the Clustered Maximum Weight Clique Problem (CCP), a generalization of the MVWCP, that models an image acquisition scheduling problem for a satellite constellation [34]. Wider information about the algorithms to solve MVWCP can be found in a literature survey by Wu & Hao [6]. El Baz et al. propose a parallel ant colony optimization based metaheuristic (PACOM) for solving MVWCP [28]. This is the only parallel heuristic algorithm that we have found out in literature for MVWCP. Its results are given for DIMACS-W problem instances. To the best of our knowledge, there is no parallel tabu search algorithm like ours designed for the solution of MVWCP.

Contemporary software/hardware support can provide instruction level parallelism and gain performance increases. However, they do not perform as well as the human designed parallel algorithms. Our proposed algorithm, PTC, intends to explore different areas of search space concurrently rather than only speeding-up the search velocity of a single tabu-search process. There are recent and successful approaches of parallel heuristic algorithms to solve combinatorial optimization problems such as quadratic assignment and bin packing problems [26, 27, 29]. Our approach is one of its first examples that is applied to the solution of the MVWCP.

3. Proposed algorithm, Parallel-Tabu-Clique (PTC)

In this section, we present our proposed Parallel-Tabu for Clique (PTC) algorithm for the MVWCP. PTC is a parallel tabu search algorithm implemented by using C++ programming language and Message Passing Interface (MPI) libraries. The PTC introduces a novel diversification mechanism to improve the efficiency of the algorithm by making use of alternative seeding support for the starting points and randomization of the search process. This mechanism is designed with respect to the rank number of the processors. A distributed probabilistic restarting process is employed with the proposed PTC algorithm. Classical *add*, *swap*, and *drop* operators of tabu search that are used for constructing a maximum clique are adapted to the distributed computation environment. A master and slave communication topology is used during the communication of processors.

3.1. Motivation for parallel heuristic optimization algorithms

Tabu search is really an efficient heuristic that has been used for the optimization of several NP-Hard combinatorial optimization problems [7][10]. However, as most of the heuristics, tabu search tries to find the optimal value by calculating the fitness value of each neighbor solution. This is a process event that consumes a serious amount of time. Classical single-processor heuristic algorithms have this disadvantage of calculating the fitness values more slowly than parallel heuristic algorithms [30][32][33]. Techniques such as dynamic programming can reduce this cost significantly but it is not always possible to apply these techniques. In our opinion, a scalable parallel algorithm that quickly calculates the fitness of the new solutions is a very important means of computation for better optimization. There is a classical way of understanding the parallel algorithms as tools of speeding-up the computation. However, with the heuristic methods it provides more than just being a faster way of execution. With the parallel heuristic algorithm we propose, we intend to increase the possibility of obtaining the optimal results through faster computation and diversified search methods of parallel implementation. All the processors seed themselves from different initial points, which is a very effective way of exploring the search space.

3.2. Fitness value of a possible MVWCP solution

In this section we give information about the fitness value of a solution. The MVWCP is a maximization problem in a given graph $G = (V, E, w)$ where V is the set of vertices, E is the set of edges and w is a weight function that assigns a positive number to each vertex. The PTC algorithm searches space Ω (set of all the cliques in G) for a maximum total sum. For a solution (clique) $C \in \Omega$, its fitness value can be given as a function of $W(C) = \sum_{i \in C} w_i$ where $W : \Omega \rightarrow Z^+$. The neighbors of C are possible elements of V that are not in the set of C and when a new clique C' is constructed by adding a new vertex and $W(C') > W(C)$ then a better solution is obtained during the optimization process of the PTC algorithm.

3.3. Initialization of a candidate maximum vertex weight clique solution

For each processor in a parallel computation environment, the PTC algorithm begins with a different starting clique, C. This is provided by a mechanism that generates random numbers with respect to the rank of the processor in the parallel computation environment. Therefore, the larger amount of processors you have, the more likely the PTC algorithm will explore the search space better. The PTC algorithm uses a parallel version of the tabu-search to improve the weight of clique C. In order to construct an initial C, the PTC algorithm selects a vertex i from G and initializes the clique C to the set consisting of this single vertex that is selected as a different starting point for each processor in the environment. Later, another vertex $v \notin C$ that has a connection with every vertex in C is selected continuously. This iteration goes on until no such vertex v exists. This is a simple and fast process with diversified initial solutions for each iteration of the tabu search procedure at each different slave. When two possible vertices are considered then one of them is selected randomly with respect to the seeding mechanism of the processor.

3.4. Operators of the PTC algorithm

The PTC algorithm explores the optimal value by using three basic operators (*add*, *swap* and *drop*) jointly.

Add operator increases the total sum of the weights by introducing a new vertex to the clique. Therefore, *add* operator has always a positive effect on the existing solution. But it is not always possible to *add* a new vertex to the existing clique and it is a costly operation to verify this. The complexity of *add* operation is O(n) where n is the number of vertices.

Swap operator exchanges one of the vertices in a clique with another one that is outside of the solution. This operation may lead to a better result or not. It depends on the values of the vertices that are *swapped*. But this exchange of the vertices may lead to a better maximum vertex weight clique. Sometimes, it is possible to have better results than applying *add* operator. In Figure 1, we can see how a *swap* operator produces a better solution than an *add* operator (see the total cost of each clique after *add* and *swap* operations. Original clique has a total cost of 15 after *add* operation whereas the total cost is 21 with *swapping*.).

Drop operator removes a vertex from the existing solution. It always decreases the total cost of the maximum vertex weight clique. However, it does not mean that it will not help us explore better solutions. Instead, it has a positive effect for the diversification of the search.

It is not wise to say that *add* operator has always a better influence for the optimization of the problem. There may be situations that an *add* operation cannot be executed. In such cases, *swap* and *drop* operators can easily diversify our search space and rescue us from local optima. Therefore, we cannot talk about supremacy of a specific operator. The best operator depends on the landscape of the search area and initial cliques. Moreover, using these operators in a union fashion is reported to be an efficient way of optimization of the MVWCP [7].

Parameter L (search depth) limits the search process for consecutive iterations without improving the clique weight. It means when there is no improvement in the result, the tabu search does not further keep on spending its exploration on this constructed solution instead it restarts a new solution.

Figure 1. The *swap* operator exchanges the vertices 1 and 2 from clique $C = \{1, 5, 6\}$ to produce a new maximum vertex weight clique $C' = \{2, 5, 6\}$. *add* operator that introduces vertex 3 to the existing clique does not produce a better maximum vertex weight clique.

The PTC algorithm uses a distributed tabu list to restrict the revisiting of previously searched solutions. At every processor's memory there is a tabu list that works independently for each tabu search process. When a vertex leaves the current clique by using a *swap* or *drop* operator it is forbidden to include this vertex into the same solution clique for the defined iteration times. This iteration is called tabu tenure. A vertex can leave the clique by using *drop* or *swap* operator without any constraint. A move is called as tabu if one of its attributes is tabu and use the aspiration criterion that lets a move to be executed even if it is a tabu and it generates a solution better than any existing solution. A move that satisfies the aspiration criterion and not tabu is permitted to be executed in tabu search.

PTC algorithm explores three neighborhoods at each iteration and moves toward a move that produces the best solution. The three neighborhoods enhances the algorithm to make a better exploration of the candidate solutions. Tabu list provides an area that should not be visited again and provides a good diversification mechanism. This is performed concurrently by several processors during the optimization. Multistart is another diversification process that is employed by the PTC algorithm. The details of the tabu search and parallel execution of the processors can be seen in Algorithms 1 and 2 respectively. Master processor is receiving the results from slave processors while it is also making a tabu search and improving the quality of the solution. In Figure 2, the flowchart of the PTC algorithm is presented.

Figure 2. Flowchart of the PTC algorithm.

3.5. Communication topology of the PTC algorithm

The PTC algorithm uses a master and slave communication topology during the execution of the algorithm. All the processors in the environment (including the master) start their tabu search process. After executing tabu search optimization 100 times, they send their solutions to the master node. Master node receives all the results coming from the slaves and report the best solutions and the other statistics obtained during the optimization process.

Algorithm 1: The multi-neighborhood tabu search algorithm for MWCP [7]

Input: A weighted graph $G = (V, E, w)$, integer L (search depth), $Iter_{max}$ (max. number of iterations)
Ensure: C^* is a clique with weight $W(C^*)$
$Iter$ = null // counter for iterations
$C* = \emptyset$
/* number times that tabu search is restarted from different vertices*/
while *($Iter < Iter_{max}$)* **do**
    C = Initialize()
    Initialize *tabu_list*
    N = 0 // iterations that $W(C^*)$ is not improved
    $C_{local\_best} = C$ // C_{local_best} is the local best Clique found until now
    while $N < L$ **do**
        Construct neighborhoods N_1, N_2, and N_3 from C
        Choose the best neighbor $C' \in N_1 \cup N_2 \cup N_3$
        $C = C'$ // current solution is new solution
        $N = N + 1$
        $Iter++$
        Update tabu_List
        if $(W(C) > W(C_{local\_best}))$ **then**
            $N = 0$
            $C_{local\_best} = C$
    if $(W(C_{local\_best}) > W(C^*))$ **then**
        $C^* = C_{local\_best}$
Return (Clique (C^*))

Algorithm 2: Pseudocode for the Master and Slave nodes of the PTC algorithm

Master side execution
select_an_initial_point(p);
tabu_search(s); // execute search process at master
for $i \leftarrow$ *1 to number_of_iterations* **do**
    for $k \leftarrow$ *1 to number_of_slaves* **do**
        receive the current solution from slave $processor_k$
        update the global best solution
report the global best result;
Slave side execution
s: current solution;
read_graph_data();
for $i \leftarrow$ *1 to number_of_iterations* **do**
    select_an_initial_point(p); // seed with the rank of the processor
    tabu_search(s); // execute search process at this processor
    send_result_to_master (s)

4. Performance Evaluation of the Experimental Results

In this section, we give information about our High Performance Cluster (HPC) experimental environment, problem instances of the MVWCP, performance evaluations (in terms solution quality and execution times), speed-up, and scalability issues of the proposed algorithm. We report the results of the proposed algorithm with the benchmark problem instances and compare its performance on those of other state-of-the-art algorithms.

4.1. Problem instances and experimental setup

We observe the efficiency of our PTC algorithm on DIMACS-W and BHOSLIB-W instances [31]. The DIMACS-W benchmark has 80 instances from a variety of real life applications. The DIMACS-W benchmark has graphs generated randomly and maximum clique has been hidden by incorporating low-degree vertices. The problem instances range from 50 vertices and 1,000 edges to 3,300 vertices and 5,000,000 edges. The BHOSLIB-W problem instances have been known to be difficult for maximum clique algorithms. Both DIMACS-W and BHOSLIB-W benchmarks have been widely used to test new MVWCP heuristics. The weighted DIMACS-W benchmarks are produced from the DIMACS benchmark instances by allocating weights to vertices. For each vertex k, w_k is set equal to (k mod 200)+1 [35]. Each problem instance is solved 100 times by our proposed algorithm with different random seeds with respect to the rank of the processors. The maximum allowed iterations Iter$_{max}$ for each run and instance is set to 10^8. For the search depth (L), L = 4,000 for the instances of the MWCP.

Our experiments are performed on a high performance cluster (HPC) computer. The machine has 46 nodes, each with 2 CPUs giving a total of 92 CPUs. Each CPU has 4 cores with a total of 368 cores. Each node has two 24 port Gigabit ethernet switches and one 24 port high performance switch. The software comprises; a Scientific Linux v4.5 64-bit operating system, Lustre v1.6.4.2 parallel file system, Torque v2.1.9 resource manager, Maui v3.2.6 job scheduler, Open MPI v1.2.4, and the C++. The instances are optimized by using 6 nodes, which means that 48 processors are used during the optimization process. For difficult instances, MANN_a27, MANN_a45, and MANN_a81, 128 processors are used to provide better results.

4.2. The effect of increasing the number of processors

We analyze the performance of the PTC algorithm with increasing number of processors. This experiment shows us whether additional processors will provide any advantage (better results) or not. Therefore, the results of this experiment are really crucial for scalability and effectiveness of our parallel heuristic algorithm. As we have explained earlier, main purpose of a parallel heuristic algorithm is not just to speed-up the calculation of the instructions but also explore the search space from possible different starting and diversified points of the problem. We select one of the hardest DIMACS-W problem instances (MANN_a27) as our test case instance. During our experiments we observe that this instance is really a difficult one and it would be interesting to observe the performance of our algorithm on this problem instance. Figures 3 and 4 show the number optimal solution hits and the average of the obtained results after 100 runs respectively. It can be easily observed that when the number of best result hits is 3 with a single processor, it becomes 74 with 128 processors. Similarly, a significant amount of increase is observed on the average number of the hits during the experiments.

Figure 3. The effect of increasing the number of processors on number of best results.

Figure 4. The effect of increasing the number of processors on the average of hits.

Tables 1 and 2 present detailed results of PTC algorithm on DIMACS-W and BHOSLIB-W benchmark problem instances respectively. Columns 1-3 define the properties of the problem instance. ω is the size of the largest known clique. Node is the number of vertices in the graph. At the other columns, we give some computational statistics about the optimization of the problem. The maximum weight obtained by PTC over the 100 runs (W_best), the average weight over the 100 trials (W_{avg}), the number of successful trials W_{best} (Success), the average time (sec.) and the average iterations [7]. W_{best} results are obtained from the study of Wu et al. [7] as most of the related studies do. When making comparisons with state-of-the-art algorithms we keep these standard values of Wu et al..

By looking at the results of the solutions, we can see that the cardinality of the obtained maximum weighted clique $|C|$ does not always have the same value with the maximum clique size (ω). Because maximum clique size ω does not mean that it will have the maximum weight of the whole graph G. There may be other smaller cliques with a larger total sum of vertices.

The hit ratio of the PTC algorithm is 100 (except for the instance MANN_a27) for all DIMACS-W instances. The PTC algorithm consumes a lot of time while solving the instances C2000.9, MANN_a27, MANN_a45, MANN_a81, and p_hat1500-3 in DIMACS-W benchmark. The average of the execution times is reasonable with the instances in this set. The execution times are observed to be higher for the BHLOBS-W benchmark instances in Table 2. That is because the number of nodes and edges are larger in this set of problems. The hit ratio of the PTC algorithm also decreases when the number of nodes and the edges are increasing in BHLOBS-W benchmark. For the problem instance frb30-15-1 with 450 nodes, the hit ratio is 100 but for the instance frb59-26-5 with 1,534 nodes, the hit ratio goes down to a ratio 6.

The most important lesson we can get after these experiments is that the PTC algorithm provides a robust performance due to its parallel and diversified exploration capability. It can be concluded that when the algorithm is run 100 times with 48 processors (slave nodes), it can grantee the optimal solutions given in the stud of Wu et al..

Table 1. Detailed results of PTC 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-found** is the best value found by the algorithm, **avg-sum** is the average of the results, **hit** is the number of obtained best results, $|C|$ is the cardinality of the obtained maximum weighted clique, **avg-iter.** is the average iteration performed during optimization.

Table 2. Detailed results of PTC algorithm on 40 BHLOBS-W benchmark instances.

4.3. Comparison with state-of-the-art algorithms

In this part of the experiments, we compare our results with those of state-of-the-art heuristic algorithms in literature. We compare our PTC algorithm with six recent heuristic algorithms designed for the solution of the MVWCP. The algorithms are the Phased Local Search (PLS) [35], Multi-Neighborhood Tabu Search (MN/TS) [7], Breakout Local Search (BLS) [19], ReTS-I [22], Iterated Local Search Variable Neighborhood Descent (ILS-VND) [24], parallel ant colony optimization based metaheuristic (PACOM) [28] and Binary Quadratic Programming (BQP) problem with the Probabilistic Tabu Search algorithm (BQP-PTS) [20]. All these algorithms are sequential and executed with a single processor. Because it is beyond our study to write parallel versions of all these algorithms and make a more fair comparison, we give the published experimental results of these algorithms.

For the other 118 instances, the PTC is able to provide better or the same results that have been found by the other algorithms. Algorithms ILS-VND, ReTS-I and BLS have the best observed results for the instances MANN_a45 and MANN_a81. These are the only two instances that PTC is not able to provide the best known results. When we take the average of the best results without including these instances, ILS-VND, ReTS-I, BLS and PTC have the same average values (4,575.4). The PTC algorithm has the best average hit success of all the algorithms for the DIMAC-W (99.0%) and BHOSLIB-W (85.1%) instances. Tables 3 and 4 present the comparison of PTC algorithm with those of the state-of-the-art heuristic algorithms.

Table 3: Comparison with state-of-the-art algorithms on DIMACS-W problem instances

Table 4: Comparison with state-of-the-art algorithms on BHLOBS-W problem instances

Table 5 gives the execution times of state-of-the-art algorithms for selected problem instances from DIMACS-W and BHOSLIB-W benchmark. Its execution time is closer to sequential tabu search algorithm MN/TS. Generally all the state-of-the-art heuristic algorithms have practical optimization times for the instances. ReTS-I and ILS-VND algorithms have the best (shortest) execution times whereas BQP-PTS algorithm has the longest running time. The PTC algorithm has reasonable execution times on the average. The problem instances of our experiments range from 50 vertices and 1,000 edges to 3,300 vertices and 5,000,000 edges. Therefore, when an instance is small, the computation time of the optimization does not take much. However, due to the intractable complexity behavior of the problem, it grows exponentially. This causes the fitness evaluation of the new solutions to consume a lot of computation power while using the operators, *add*, *swap* and *drop*.

Table 5: Execution times of state-of-the-art algorithms for some selected problem instances. Units are given as seconds.

A study by El Baz et al. that proposes a parallel ant colony optimization based metaheuristic (PACOM) for solving the MVWCP was the only parallel algorithm that we have come across during our literature survey [28]. The performance of the PACOM is evaluated on the set of DIMACS-W problem instances by using 8 processors. For 77 instances in this problem set we report the same results. For MANN_a81, PACOM performs better than PTC whereas PTC reports better solutions for MANN_a27 and MANN_a45.

Genetic algorithms make use of a top-down approach in their operators while tabu-search like trajectory heuristics use a constructive manner from bottom to the top. They introduce a chromosome that shows the selected vertices with 0s and 1s to construct a clique. It is a difficult job for genetic algorithms to provide valid chromosomes after each crossover and mutation. Therefore, they spend much of their time for valid chromosome verifications. When the best reported references are considered for the solution of the MVWCP, there is no genetic algorithm. This shows the drawback of the genetic algorithms for this specific problem. It is clear from the given experimental results in our tables, the operators developed for the MVWCP are reported to obtain best/optimal solutions.

Speed up and scalability are crucial points to be considered for a parallel algorithm. The PTC algorithm runs on several processors simultaneously. Its nature is island parallel, which means that each processor does not wait or send any data to each other (except the master processor). This behavior of the PTC algorithm provides a cohesive structure during the optimization of each processor. Therefore, it is scalable that as many as new processors are added to the environment, the performance of the algorithm improves whether the execution time is increasing linearly. This is a very good virtue for parallel algorithms. As we have explained before, the main focus of the PTC algorithm is to increase the probability of finding the optimal value rather than speeding up the optimization. Restarting a new solution from a different clique is a very effective way of escaping from local optima and for exploring diversified places by using a parallel computation environment.

Local heuristic algorithms consume most of their time for the fitness evaluation of the new solutions while they are exploring the search space. This is one of the most important disadvantages of the heuristic algorithms. On the average, a fitness evaluation is calculated millions of times even during an ordinary optimization period. Speeding up this process and evaluating the neighbors quickly enhances the performance of the heuristic algorithms significantly.

5. Conclusions and future work

We proposed a novel robust and cooperative parallel heuristic algorithm for the MVWCP. The proposed PTC algorithm represents one of the first parallel heuristic algorithms to solve the MVWCP. The PTC competes with state-of-the art algorithms on most of the problem instances with its robust solution quality. Among the algorithms that use the same operators (*add*, *swap* and *drop*), the PTC algorithm can be evaluated as the best performing one with significant improvements. Although parallel heuristic algorithms are very effective approaches, new optimization operators like PUSH and novel intelligent approaches are still crucial tools as well as parallel computation for heuristic algorithms. For future work, we plan to study on an automated parameter tuning parallel tabu search algorithm for the MVWCP. It is also interesting to execute several heuristic approaches (hyper-heuristics) simultaneously in a parallel high performance environment. Hyper-heuristics is a novel approach that can optimize a problem instance by using best of different heuristics. We also believe that finding a chance to optimize the MVWCP by using thousands/millions of processors can provide new best results.

References

[1] Karp, R. M. (1972). Reducibility among combinatorial problems. In Complexity of computer computations (85-103).

[2] Ballard, D. H. , & Brown, C. M. (1982). Computer vision . Prentice-Hall Englewood Cliff.

[3] Bomze, I. M., Budinich, M., Pardalos, P. M., & Pelillo, M. (1999). The maximum clique problem. In Handbook of combinatorial optimization (1-74). Springer US.

[4] Wu, Q. , & Hao, J. K. (2016). A clique-based exact method for optimal winner determination in combinatorial auctions. Information Sciences, 334 , 103121 .

[5] Zhou, Y., Hao, J. K., & Duval, B. (2017). Opposition-based Memetic Search for the Maximum Diversity Problem. IEEE Transactions on Evolutionary Computation.

[6] Wu, Q., & Hao, J. K. (2015). A review on algorithms for maximum clique problems. European Journal of Operational Research, 242(3), 693-709.

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

[8] Alidaee, B., Glover, F., Kochenberger, G., & Wang, H. (2007). Solving the maximum edge weight clique problem via unconstrained quadratic programming. European Journal of Operational Research, 181(2), 592-597.

[9] Dijkhuizen, G., & Faigle, U. (1993). A cutting-plane approach to the edge-weighted maximal clique problem. European Journal of Operational Research, 69(1), 121-130.

[10] Glover, F., & Laguna, M. (2013). Tabu Search. In Handbook of Combinatorial Optimization (3261-3362).

[11] Östergärd , P. R. (2002). A fast algorithm for the maximum clique problem. Discrete Applied Mathematics, 120(1), 197-207.

[12] Kumlander, D. (2004). A new exact algorithm for the maximum-weight clique problem based on a heuristic vertex-coloring and a backtrack search. In Proc. 5th Intl Conf. on Modelling, Computation and Optimization in Information Systems and Management Sciences (pp. 202-208).

[13] Warren, J. S., & Hicks, I. V. (2006). Combinatorial branch-and-bound for the maximum weight independent set problem. Relatorio tecnico, Texas A&M University.

[14] Mannino, C., & Stefanutti, E. (1999). An augmentation algorithm for the maximum weighted stable set problem. Computational Optimization and Applications, 14(3), 367-381.

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

[16] Carraghan, R., & Pardalos, P. M. (1990). An exact algorithm for the maximum clique problem. Operations Research Letters, 9(6), 375-382.

[17] Singh, A., & Gupta, A. K. (2006). A hybrid heuristic for the maximum clique problem. Journal of Heuristics, 12(1), 5-22.

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

[19] Benlic, U., & Hao, J. K. (2013). Breakout local search for maximum clique problems. Computers & Operations Research, 40(1), 192-206.

[20] Wang, Y., Hao, J. K., Glover, F., Lu, Z., & Wu, Q. (2016). Solving the maximum vertex weight clique problem via binary quadratic programming. Journal of Combinatorial Optimization, 32(2), 531-549.

[21] Grosso, A., Locatelli, M., & Pullan, W. (2008). Simple ingredients leading to very efficient heuristics for the maximum clique problem. Journal of Heuristics, 14(6), 587-612.

[22] Zhou, Y., Hao, J. K., & Goeffon, A. (2017). PUSH: A generalized operator for the Maximum Vertex Weight Clique Problem. European Journal of Operational Research, 257(1), 41-54.

[23] Bomze, I. R., Pelillo, M., & Stix, V. (2000). Approximating the maximum weight clique using replicator dynamics. IEEE Transactions on neural networks, 11(6), 1228-1241.

[24] Nogueira, B., Pinheiro, R. G., & Subramanian, A. (2017). A hybrid iterated local search heuristic for the maximum weight independent set problem. Optimization Letters, 1-17.

[25] Busygin, S. (2006). A new trust region technique for the maximum weight clique problem. Discrete Applied Mathematics, 154(15), 2080-2096.

[26] Dokeroglu, T. (2015). Hybrid teaching-learning-based optimization algorithms for the quadratic assignment problem. Computers & Industrial Engineering, 85, 86-101.

[27] Dokeroglu, T., & Cosar, A. (2016). A novel multistart hyper-heuristic algorithm on the grid for the quadratic assignment problem. Engineering Applications of Artificial Intelligence, 52, 10-25.

[28] El Baz, D., Hifi, M., Wu, L., & Shi, X. (2016). A Parallel Ant Colony Optimization for the Maximum-Weight Clique Problem. In Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International (pp. 796-800).

[29] 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 Computing, 1-19.

[30] James, T., Rego, C., & Glover, F. (2009). A cooperative parallel tabu search algorithm for the quadratic assignment problem. European Journal of Operational Research, 195(3), 810-826.

[31] Johnson, D. S., & Trick, M. A. (Eds.). (1996). Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993 (Vol. 26).

[32] Gendreau, M., Guertin, F., Potvin, J. Y., & Taillard, E. (1999). Parallel tabu search for real-time vehicle routing and dispatching. Transportation science, 33(4), 381-390.

[33] Lu, Y., Cao, B., & Glover, F. (2017). A Tabu Search based clustering algorithm and its parallel implementation on Spark. arXiv preprint arXiv:1702.01396.

[34] Malladi, K. T., Mitrovic-Minic, S., & Punnen, A. P. (2017). Clustered maximum weight clique problem: Algorithms and empirical analysis. Computers & Operations Research, 85, 113-128.

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

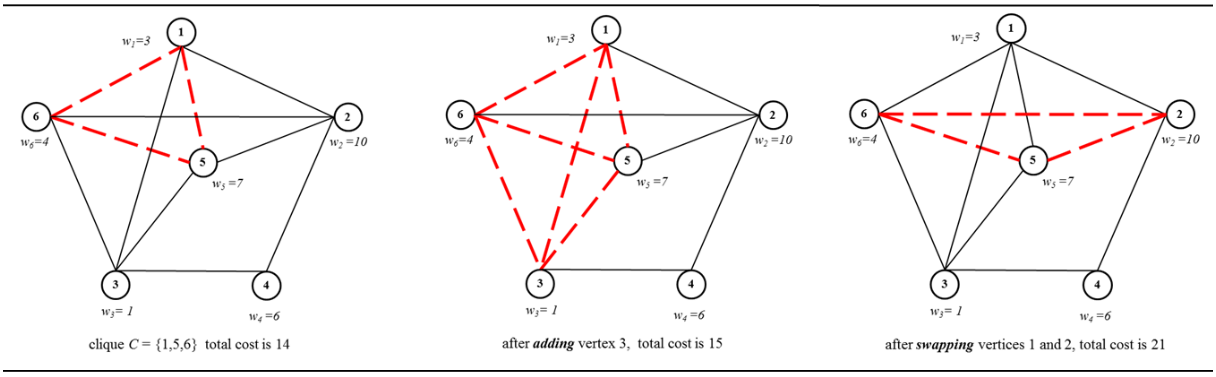

Figure 1: The *swap* operator exchanges the vertices 1 and 2 from clique $C = \{1, 5, 6\}$ to produce a new maximum vertex weight clique $C' = \{2, 5, 6\}$. *add* operator that introduces vertex 3 to the existing clique does not produce a better maximum vertex weight clique.

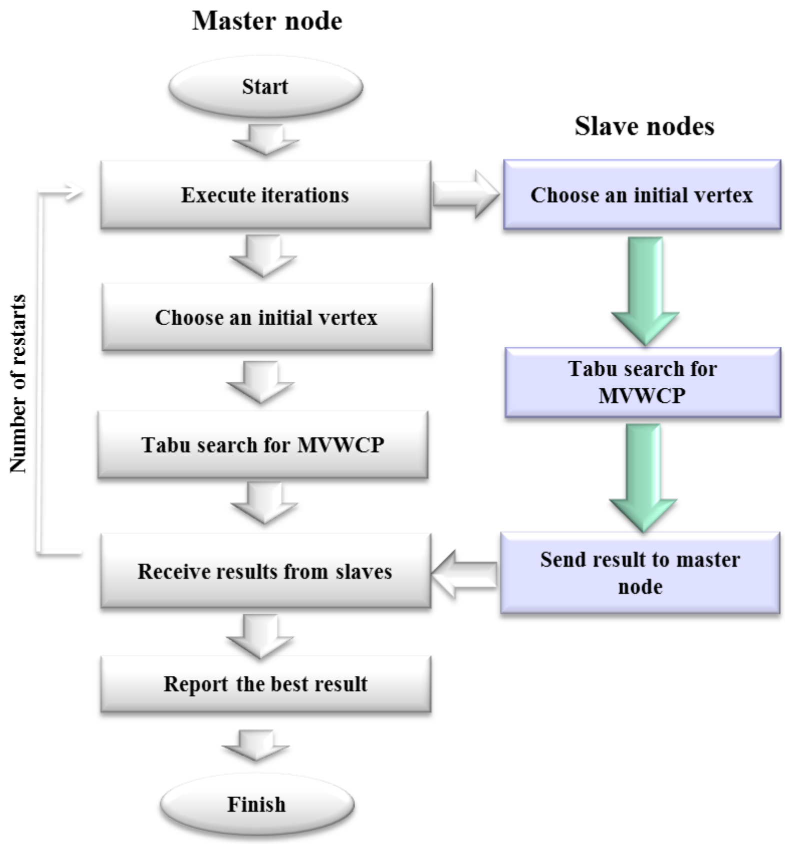

Figure 2: Flowchart of the PTC algorithm.

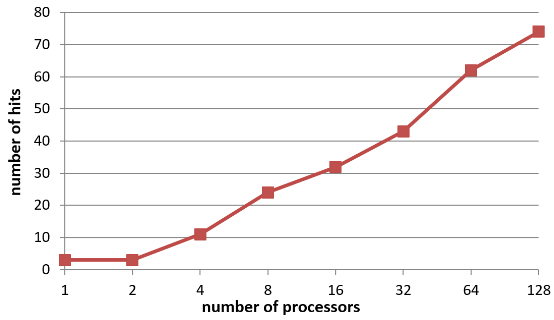

Figure 3: The effect of increasing the number of processors on number of best results.

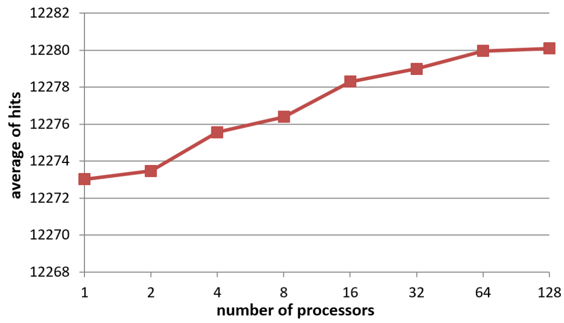

Figure 4: The effect of increasing the number of processors on the average of hits.

Table 1: Detailed results of PTC 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-found** is the best value found by the algorithm, **avg-sum** is the average of the results, **hit** is the number of obtained best results, $|C|$ is the cardinality of the obtained maximum weighted clique, **avg-iter.** is the average iteration performed during optimization.

instance	**node**	ω	**W_best**	**best-found**	**avg-sum**	**hit**	$\|C\|$	**avg-iter.**	**time (sec.)**
brock200_1	200	21	2,821	2,821	2,821	100	19	874	0.7
brock200_2	200	12	1,428	1,428	1,428	100	9	533	0.4
brock200_3	200	15	2,062	2,062	2,062	100	13	525	0.1
brock200_4	200	17	2,107	2,107	2,107	100	13	1,096	0.8
brock400_1	400	27	3,422	3,422	3,422	100	21	5,016	4.6
brock400_2	400	29	3,350	3,350	3,350	100	21	4,537	4.3
brock400_3	400	31	3,471	3,471	3,471	100	23	5,163	4.9
brock400_4	400	33	3,626	3,626	3,626	100	33	2,478,760	2550.0
brock800_1	800	23	3,121	3,121	3,121	100	20	4,712	9.0
brock800_2	800	34	3,043	3,043	3,043	100	18	27,718	54.3
brock800_3	800	25	3,076	3,076	3,076	100	20	13,387	26.2
brock800_4	800	26	2,971	2,971	2,971	100	26	7,760,827	15,261.0
C125.9	125	34	2,529	2,529	2,529	100	30	38,465	22.0
C250.9	250	44	5,092	5,092	5,092	100	40	23,786	15.2
C500.9	500	57	6,955	6,955	6,955	100	48	17,942	15.4
C1000.9	1,000	68	9,254	9,254	9,254	100	61	2,231,531	2,848.3
C2000.5	2,000	16	2,466	2,466	2,466	100	14	56,029	321.2
C2000.9	2,000	80	10,999	10,999	10,999	100	72	74,945,585	345,318.4
C4000.5	4,000	18	2,792	2,792	2,792	100	16	1,403,031	15,205.3
DSJC500.5	500	13	1,725	1,725	1,725	100	12	6,169	0.0
DSJC1000.5	1,000	15	2,186	2,186	2,186	100	13	13,873	0.1
keller4	171	11	1,153	1,153	1,153	100	11	5,600	0.0
keller5	776	27	3,317	3,317	3,317	100	27	484	812.2
keller6	3,361	59	3,361	3,361	3,361	100	56	530,880,620	610.4
MANN_a9	45	16	372	372	372	100	16	80	0.1
MANN_a27	378	126	12,281	12,283	12,273.03	20	126	36,281,025	208,800.0
MANN_a45	1,035	345	34,192	34,205	>34,192	100	341	53,420,000	550,950.0
MANN_a81	3,321	1,100	111,128	111,146	>111,128	100	1095	91,824,000	3,008,420.0
hamming6-2	64	32	1,072	1,072	1,072	100	32	397	0.0
hamming6-4	64	4	134	134	134	100	4	4	0.0
hamming8-2	256	128	10,976	10,976	10,976	100	128	14,027	0.0
hamming8-4	256	16	1,472	1,472	1,472	100	16	127	0.0
hamming10-2	1,024	512	50,512	50,512	50,512	100	512	188,670	0.4
hamming10-4	1,024	40	5,129	5,129	5,129	100	35	418,091	1.2
gen200_p0.9_44	200	44	5,043	5,043	5,043	100	37	1,837	0.0
gen200_p0.9_55	200	55	5,416	5,416	5,416	100	52	294,403	0.2
gen400_p0.9_55	400	55	6,718	6,718	6,718	100	47	76,033	0.1
gen400_p0.9_65	400	65	6,940	6,940	6,940	100	48	12,692	0.0
gen400_p0.9_75	400	75	8,006	8,006	8,006	100	78	295,458	0.3
c-fat200-1	200	12	1,284	1,284	1,284	100	12	137,972	0.2
c-fat200-2	200	24	2,411	2,411	2,411	100	23	67,423	0.1
c-fat200-5	200	58	5,887	5,887	5,887	100	58	24,178	0.0
c-fat500-1	500	14	1,354	1,354	1,354	100	12	337,732	1.0
c-fat500-2	500	26	2,628	2,628	2,628	100	24	193,624	0.5
c-fat500-5	500	64	5,841	5,841	5,841	100	62	70,262	0.2
c-fat500-10	500	126	11,586	11,586	11,586	100	124	22,844	0.0
johnson8-2-4	28	4	66	66	66	100	4	3,185	0.0
johnson8-4-4	70	14	511	511	511	100	14	16	0.0
johnson16-2-4	120	8	548	548	548	100	8	181,088	0.0
johnson32-2-4	496	16	2,033	2,033	2,033	100	16	223,153	0.3
p_hat300-1	300	8	1,057	1,057	1,057	100	7	1,678	0.0
p_hat300-2	300	25	2,487	2,487	2,487	100	20	2,179	0.0
p_hat300-3	300	36	3,774	3,774	3,774	100	29	8,809	0.0
p_hat500-1	500	9	1,231	1,231	1,231	100	8	2,444	0.0
p_hat500-2	500	36	3,920	3,920	3,892	100	31	11,666	0.0
p_hat500-3	500	50	5,375	5,375	5,375	100	42	57,191	0.1
p_hat700-1	700	11	1,441	1,441	1,441	100	9	852	0.0
p_hat700-2	700	44	5,290	5,290	5,290	100	40	3,874	0.0

Table 1 Continued

instance	**node**	ω	**W_best**	**best-found**	**avg-sum**	**hit**	$\|C\|$	**avg-iter.**	**time (sec.)**
p_hat700-3	700	62	7,565	7,565	7,565	100	58	18,3757	0.2
p_hat1000-1	1,000	10	1,514	1,514	1,514	100	9	3,039	0.0
p_hat1000-2	1,000	46	5,777	5,777	5,777	100	42	7,066	0.0
p_hat1000-3	1,000	68	8,111	8,111	8,111	100	58	16,8145	0.3
p_hat1500-1	1,500	12	1,619	1,619	1,619	100	10	3,517	0.0
p_hat1500-2	1,500	65	7,360	7,360	7,360	100	58	55,019	0.2
p_hat1500-3	1,500	94	10,321	10,321	10,321	100	84	17,488,756	119,572.7
san200_0.7_1	200	30	3,370	3,370	3,370	100	30	52,271	0.1
san200_0.7_2	200	18	2,422	2,422	2,422	100	14	3,625	0.0
san200_0.9_1	200	70	6,825	6,825	6,825	100	70	123,211	0.2
san200_0.9_2	200	60	6,082	6,082	6,082	100	60	153,595	0.1
san200_0.9_3	200	44	4,748	4,748	4,748	100	34	4,420	0.0
san400_0.5_1	400	13	1,455	1,455	1,455	100	8	4,378	0.0
san400_0.7_1	400	40	3,941	3,941	3,941	100	40	3,859,008	37,669.3
san400_0.7_2	400	30	3,110	3,110	3,110	100	30	8,922,943	89,689.4
san400_0.7_3	400	22	2,771	2,771	2,771	100	18	7,454	0.0
san400_0.9_1	400	100	9,776	9,776	9,776	100	100	654,493	1.6
san1000	1,000	15	1,716	1,716	1,716	100	9	453,934	8.0
sanr200-0.7	200	18	2,325	2,325	2,325	100	15	478	0.0
sanr200-0.9	400	42	5,126	5,126	5,126	100	36	683	0.0
sanr400-0.5	400	13	1,835	1,835	1,835	100	11	1,681	0.0
sanr400-0.7	400	21	2,992	2,992	2,992	100	18	1,592	0.0

Table 2: Detailed results of PTC algorithm on 40 BHLOBS-W benchmark instances.

instance	**node**	ω	**W_best**	**best-found**	**avg-sum**	**hit**	$\|C\|$	**avg-iter.**	**time (sec.)**
frb30-15-1	450	30	2,990	2,990	2,990	100	27	126,687	0.2
frb30-15-2	450	30	3,006	3,006	3,006	100	28	1,067,223	1.4
frb30-15-3	450	30	2,995	2,995	2,995	100	27	1,407,315	1.9
frb30-15-4	450	30	3,032	3,032	3,032	100	28	35,719	0.0
frb30-15-5	450	30	3,011	3,011	3,011	100	27	667,587	0.9
frb35-17-1	595	35	3,650	3,650	3,650	100	33	26,631,649	9.6
frb35-17-2	595	35	3,738	3,738	3,738	100	33	36,933,198	85,666.0
frb35-17-3	595	35	3,716	3,716	3,716	100	33	2,903,709	4.3
frb35-17-4	595	35	3,683	3,683	3,683	100	35	43,527,142	1,007,709.0
frb35-17-5	595	35	3,686	3,686	3,686	100	33	2,963,616	9,108.0
frb40-19-1	760	40	4,063	4,063	4,063	100	37	11,547,245	100,402.0
frb40-19-2	760	40	4,112	4,112	4,112	100	36	28,624,110	114,118.0
frb40-19-3	760	40	4,115	4,115	4,115	100	36	44,720,006	143,785.9
frb40-19-4	760	40	4,136	4,136	4,136	100	37	18,264,443	110,780.9
frb40-19-5	760	40	4,118	4,118	4,118	100	36	31,568,964	95,982.0
frb45-21-1	945	45	4,760	4,760	4,760	100	41	44,554,714	211,587.6
frb45-21-2	945	45	4,784	4,784	4,748	100	42	40,172,601	154,734.0
frb45-21-3	945	45	4,765	4,765	4,765	100	43	56,334,801	211,794.0
frb45-21-4	945	45	4,799	4,799	4,799	100	42	31,784,789	1,533,997.0
frb45-21-5	945	45	4,779	4,799	4,779	100	43	29,248,923	1,590,248.0
frb50-23-1	1,150	50	5,494	5,494	5,491.4	64	47	24,620,081	110,090.5
frb50-23-2	1,150	50	5,462	5,462	5,462	100	47	43,007,000	182,190.0
frb50-23-3	1,150	50	5,486	5,486	5,486	100	47	53,901,774	237,783.0
frb50-23-4	1,150	50	5,454	5,454	5,454	100	46	44,146,546	284,538.0
frb50-23-5	1,150	50	5,498	5,498	5,498	100	47	36,065,699	195,685.0
frb53-24-1	1,272	53	5,670	5,670	5,670	100	50	66,197,822	3,038,090.0
frb53-24-2	1,272	53	5,707	5,707	5,707	100	49	32,412,000	152,460.0
frb53-24-3	1,272	53	5,640	5,655	5,640	100	49	77,652,000	360,430.0
frb53-24-4	1,272	53	5,714	5,714	5,704.4	64	48	32,135,000	1,442,252.0
frb53-24-5	1,272	53	5,659	5,659	5,658.2	60	49	96,896,198	435,070.0
frb56-25-1	1,400	56	5,916	5,916	5,904.5	82	53	39,412,141	442,150.0
frb56-25-2	1,400	56	5,872	5,872	5,864.3	73	52	24,211,152	194,171.0
frb56-25-3	1,400	56	5,859	5,859	5,849.3	76	51	17,845,998	267,952.0
frb56-25-4	1,400	56	5,892	5,892	5,891.4	82	51	31,244,856	181,049.0
frb56-25-5	1,400	56	5,839	5,839	5,834.6	85	52	87,455,412	543,146.0
frb59-26-1	1,534	59	6,591	6,591	6,583.3	52	53	54,389,333	88,456.6
frb59-26-2	1,534	59	6,645	6,645	6,617.4	46	55	46,250,666	133,040.0
frb59-26-3	1,534	59	6,608	6,608	6,582.74	6	55	66,908,000	328,810.0
frb59-26-4	1,534	59	6,592	6,592	6,542.13	8	54	83,803,679	432,260.0
frb59-26-5	1,534	59	6,584	6,584	6,562.4	6	53	62,640,547	318,230.0

Table 3: Comparison with state-of-the-art algorithms on DIMACS-W problem instances

	BQP-PTS		**PLSW**		**MN/TS**			**BLS**			**ReTS - I**			**ILS-VND**		**PTC**		
	Best	Succ.	Best	Succ.	Best	Succ.	Avg	Best	Succ.	Avg	Best	Succ.	Avg	Best	Avg	Best	Succ.	Avg
brock200_1	2,821	100	2,821	100	2,821	100	2,821	2,821	100	2,821	2,821	100	2,821	2,821	2,821	2,821	100	2,821
brock200_2	1,428	100	1,428	100	1,428	100	1,428	1,428	100	1,428	1,428	100	1,428	1,428	1,428	1,428	100	1,428
brock200_3	2,062	100	2,062	100	2,062	100	2,062	2,062	100	2,062	2,062	100	2,062	2,062	2,062	2,062	100	2,062
brock200_4	2,107	100	2,107	100	2,107	100	2,107	2,107	100	2,107	2,107	100	2,107	2,107	2,107	2,107	100	2,107
brock400_1	3,422	100	3,422	32	3,422	32	3,422	3,422	100	3,422	3,422	100	3,422	3,422	3,422	3,422	100	3,422
brock400_2	3,350	100	3,350	61	3,350	61	3,350	3,350	100	3,350	3,350	100	3,350	3,350	3,350	3,350	100	3,350
brock400_3	3,471	100	3,471	100	3,471	100	3,471	3,471	100	3,471	3,471	100	3,471	3,471	3,471	3,471	100	3,471
brock400_4	3,626	100	3,626	100	3,626	100	3,626	3,626	100	3,626	3,626	100	3,626	3,626	3,626	3,626	100	3,626
brock800_1	3,121	100	3,121	100	3,121	100	3,121	3,121	100	3,121	3,121	100	3,121	3,121	3,121	3,121	100	3,121
brock800_2	3,043	100	3,043	69	3,043	100	3,043	3,043	69	3,043	3,043	100	3,043	3,043	3,043	3,043	100	3,043
brock800_3	3,076	100	3,076	100	3,076	100	3,076	3,076	100	3,076	3,076	100	3,076	3,076	3,076	3,076	100	3,076
brock800_4	2,971	8	2,971	100	2,971	100	2,971	2,971	100	2,971	2,971	31	2,970.31	2,971	2,970.69	2,971	100	2,971
C125.9	2,529	100	2,529	100	2,529	100	2,529	2,529	100	2,529	2,529	100	2,529	2,529	2,529	2,529	100	2,529
C250.9	5,092	100	5,092	17	5,092	100	5,092	5,092	100	5,092	5,092	100	5,092	5,092	5,092	5,092	100	5,092
C500.9	6,955	100	6,822	-	6,955	100	6,955	6,955	100	6,955	6,955	100	6,955	6,955	6,955	6,955	100	6,955
C1000.9	9,254	100	8,965	5	9,254	100	9,254	9,254	100	9,254	9,254	100	9,254	9,254	9,254	9,254	100	9,254
C2000.5	2,466	71	2,466	18	2,466	100	2,466	2,466	100	2,466	2,466	100	2,466	2,466	2,466	2,466	100	2,466
C2000.9	10,999	72	10,028	-	10,999	22	10,972.32	10,999	74	10,989.9	10,999	92	10,996.44	10,999	10,973.6	10,999	100	10,999
C4000.5	2,792	19	2,792	-	2,792	100	2,792	2,792	100	2,792	2,792	100	2,792	2,792	2,790.3	2,792	100	2,792
DSJC500.5	1,725	100	1,725	100	1,725	100	1,725	-	-	-	1,725	100	1,725	1,725	1,725	1,725	100	1,725
DSJC1000.5	2,186	81	2,186	100	2,186	100	2,186	-	-	-	2,186	100	2,186	2,186	2,186	2,186	100	2,186
keller4	1,153	100	1,153	100	1,153	100	1,153	1,153	100	1,153	1,153	100	1,153	1,153	1,153	1,153	100	1,153
keller5	3,317	100	3,317	100	3,317	100	3,317	3,317	100	3,317	3,317	100	3,317	3,317	3,317	3,317	100	3,317
keller6	8,062	2	7,382	-	8,062	5	7,945.12	8,062	44	8,027.2	8,062	100	8,062	8,062	8,061.7	8,062	100	8,062
MANN_a9	372	100	372	100	372	100	372	-	-	372	372	100	372	372	372	372	100	372
MANN_a27	12,277	4	12,264	-	12,281	1	12,273	12,281	16	12,277	12,283	78	12,282.78	12,283	12,283	12,283	20	12,283
MANN_a45	34,194	2	34,129	-	34,192	1	34,180	34,229	1	34,211	34,259	1	34,253.6	34,265	34,263	34,205	100	34,192
MANN_a81	111,137	1	110,564	-	111,128	1	111,116	11,1237	1	111,188	111,370	1	111,351.19	111,400	111,394	11,1146	100	111,128
hamming6-2	1,072	100	1,072	100	1,072	100	1,072	1,072	100	1,072	1,072	100	1,072	1,072	1,072	1,072	100	1,072
hamming6-4	134	100	134	100	134	100	134	134	100	134	134	100	134	134	134	134	100	134
hamming8-2	10,976	100	10,976	100	10,976	100	10,976	10,976	100	10,976	10,976	100	10,976	10,976	10,870	10,976	100	10,976
hamming8-4	1,472	100	1,472	100	1,472	100	1,472	1,472	100	1,472	1,472	100	1,472	1,472	1,472	1,472	100	1,472
hamming10-2	50,512	67	50,512	100	50,512	100	50,512	50,512	100	50,512	50,512	100	50,512	50,512	50,420	50,512	100	50,512
hamming10-4	5,129	8	5,086	1	5,129	100	5,129	5,129	100	5,129	5,129	100	5,129	5,129	5,128.9	5,129	100	5,129
gen200_p0.9_44	5,043	100	5,043	100	5,043	100	5,043	5,043	100	5,043	5,043	100	5,043	5,043	5,043	5,043	100	5,043
gen200_p0.9_55	5,416	100	5,416	100	5,416	100	5,416	5,416	100	5,416	5,416	100	5,416	5,416	5,416	5,416	100	5,416
gen400_p0.9_55	6,718	100	6,718	2	6,718	100	6,718	6,718	2	6,718	6,718	100	6,718	6,718	6,718	6,718	100	6,718
gen400_p0.9_65	6,940	100	6,935	4	6,940	100	6,940	6,940	100	6,940	6,940	100	6,940	6,940	6,940	6,940	100	6,940
gen400_p0.9_75	8,006	100	8,006	100	8,006	100	8,006	8,006	100	8,006	8,006	100	8,006	8,006	8,006	8,006	100	8,006
c-fat200-1	1,284	100	1,284	100	1,284	100	1,284	-	-	-	1,284	100	1,284	1,284	1,284	1,284	100	1,284
c-fat200-2	2,411	100	2,411	100	2,411	100	2,411	-	-	-	2,411	100	2,411	2,411	2,411	2,411	100	2,411
c-fat200-5	5,887	100	5,887	100	5,887	100	5,887	-	-	-	5,887	100	5,887	5,887	5,887	5,887	100	5,887
c-fat500-1	1,354	100	1,354	100	1,354	100	1,354	-	-	-	1,354	100	1,354	1,354	1,354	1,354	100	1,354
c-fat500-2	2,628	100	2,628	100	2,628	100	2,628	-	-	-	2,628	100	2,628	2,628	2,628	2,628	100	2,628
c-fat500-5	5,841	100	5,841	100	5,841	100	5,841	-	-	-	5,841	100	5,841	5,841	5,841	5,841	100	5,841
c-fat500-10	11,586	100	11,586	100	11,586	100	11,586	-	-	-	11,586	100	11,586	11,586	11,586	11,586	100	11,586
johnson8-2-4	66	100	66	100	66	100	66	66	100	66	66	100	66	66	66	66	100	66
johnson8-4-4	511	100	511	100	511	100	511	511	100	511	511	100	511	511	511	511	100	511
johnson16-2-4	548	100	548	100	548	100	548	548	100	548	548	100	548	548	548	548	100	548
johnson32-2-4	2,033	40	2,033	100	2,033	100	2,033	2,033	100	2,033	2,033	100	2,033	2,033	2,033	2,033	100	2,033
p_hat300-1	1,057	100	1,057	100	1,057	100	1,057	1,057	100	1,057	1,057	100	1,057	1,057	1,057	1,057	100	1,057
p_hat300-2	2,487	100	2,487	100	2,487	100	2,487	2,487	100	2,487	2,487	100	2,487	2,487	2,487	2,487	100	2,487

Table 3: Continued

	BQP-PTS		PLSW		MN/TS			BLS			ReTS - I			ILS-VND		PTC		
	Best	Succ.	Best	Succ.	Best	Succ.	Avg	Best	Succ.	Avg	Best	Succ.	Avg	Best	Avg	Best	Succ.	Avg
p_hat300-3	3,774	100	3,774	47	3,774	47	3,774	3,774	47	3,774	3,774	100	3,774	3,774	3,774	3,774	100	3,774
p_hat500-1	1,231	100	1,231	100	1,231	100	1,231	1,231	100	1,231	1,231	100	1,231	1,231	1,231	1,231	100	1,231
p_hat500-2	3,920	100	3,925	-	3,920	100	3,920	3,920	100	3,920	3,920	100	3,920	3,920	3,920	3,920	100	3,920
p_hat500-3	5,375	100	5,361	-	5,375	100	5,375	5,375	100	5,375	5,375	100	5,375	5,375	5,375	5,375	100	5,375
p_hat700-1	1,441	100	1,441	100	1,441	100	1,441	1,441	100	1,441	1,441	100	1,441	1,441	1,441	1,441	100	1,441
p_hat700-2	5,290	100	5,290	100	5,290	100	5,290	5,290	100	5,290	5,290	100	5,290	5,290	5,290	5,290	100	5,290
p_hat700-3	7,565	100	7,565	12	7,565	100	7,565	7,565	100	7,565	7,565	100	7,565	7,565	7,565	7,565	100	7,565
p_hat1000-1	1,514	100	1,514	100	1,514	100	1,514	1,514	100	1,514	1,514	100	1,514	1,514	1,514	1,514	100	1,514
p_hat1000-2	5,777	100	5,777	87	5,777	87	5,777	5,777	87	5,777	5,777	100	5,777	5,777	5,777	5,777	100	5,777
p_hat1000-3	8,111	100	7,986	-	8,111	100	8,111	8,111	100	8,111	8,111	100	8,111	8,111	8,111	8,111	100	8,111
p_hat1500-1	1,619	95	1,619	100	1,619	100	1,619	1,619	100	1,619	1,619	100	1,619	1,619	1,619	1,619	100	1,619
p_hat1500-2	7,360	100	7,328	4	7,360	100	7,360	7,360	100	7,360	7,360	100	7,360	7,360	7,360	7,360	100	7,360
p_hat1500-3	10,321	9	10,014	-	10,321	96	10,320.5	10,321	100	1,0321	10,321	100	10,321	10,321	10,321	10,321	100	10,321
san200_0.7_1	3,370	100	3,370	100	3,370	100	3,370	3,370	100	3,370	3,370	100	3,370	3,370	3,370	3,370	100	3,370
san200_0.7_2	2,422	100	2,422	66	2,422	100	2,422	2,422	100	2,422	2,422	100	2,422	2,422	2,422	2,422	100	2,422
san200_0.9_1	6,825	100	6,825	100	6,825	100	6,825	6,825	100	6,825	6,825	100	6,825	6,825	6,825	6,825	100	6,825
san200_0.9_2	6,082	100	6,082	100	6,082	100	6,082	6,082	100	6,082	6,082	100	6,082	6,082	6,082	6,082	100	6,082
san200_0.9_3	4,748	100	4,748	72	4,748	72	4,748	4,748	100	4,748	4,748	100	4,748	4,748	4,748	4,748	100	4,748
san400_0.5_1	1,455	100	1,455	100	1,455	100	1,455	1,455	100	1,455	1,455	100	1,455	1,455	1,455	1,455	100	1,455
san400_0.7_1	3,941	100	3,941	100	3,941	100	3,941	3,641	98	3,640.64	3,941	97	3,932	3,941	3,941	3,941	100	3,941
san400_0.7_2	3,110	100	3,110	100	3,110	100	3,110	3,110	33	3,002.56	3,110	97	3,105.26	3,110	3,110	3,110	100	3,110
san400_0.7_3	2,771	99	2,771	100	2,771	100	2,771	2,771	100	2,771	2,771	100	2,771	2,771	2,771	2,771	100	2,771
san400_0.9_1	9,776	100	9,776	100	9,776	100	9,776	9,776	100	9,776	9,776	100	9,776	9,776	9,486.25	9,776	100	9,776
san1000	1,716	100	1,716	-	1,716	100	1,716	1,716	100	1,716	1,716	100	1,716	1,716	1,716	1,716	100	1,716
sanr200-0.7	2,325	100	2,325	100	2,325	100	2,325	2,325	100	2,325	2,325	100	2,325	2,325	2,325	2,325	100	2,325
sanr200-0.9	5,126	100	5,126	5	5,126	100	5,126	5,126	100	5,126	5,126	100	5,126	5,126	5,126	5,126	100	5,126
sanr400-0.5	1,835	100	1,835	100	1,835	100	1,835	1,835	100	1,835	1,835	100	1,835	1,835	1,835	1,835	100	1,835
sanr400-0.7	2,992	100	2,992	100	2,992	100	2,992	2,992	100	2,992	2,992	100	2,992	2,992	2,992	2,992	100	2,992
Average	6,373.9	88.5	6,333.3	83.9	6,373.8	91.6	6,371.6	6,778.2	91.0	6,684.9	6,377.7	96.2	6,377.2	6,378.1	6,371.6	6,374.2	**99.0**	6,373.4

Table 4: Comparison with state-of-the-art algorithms on BHLOBS-W problem instances

	BQP-PTS			**MN/TS**			**BLS**			**ReTS-I**			**ILS-VND**		**PTC**		
	Best	Succ.	Avg	Best	Succ.	Avg	Best	Succ.	Avg	Best	Succ.	Avg	Best	Avg	Best	Succ.	Avg
frb30-15-1	2,990	100	2,990	2,990	100	2,990	2,990	100	2,990	2,990	100	2,990.0	2,990	2,990	2,990	100	2,990
frb30-15-2	3,006	100	3,006	3,006	100	3,006	3,006	100	3,006	3,006	100	3,006.0	3,006	3,006	3,006	100	3,006
frb30-15-3	2,995	100	2,995	2,995	100	2,995	2,995	100	2,995	2,995	100	2,995.0	2,995	2,995	2,995	100	2,995
frb30-15-4	3,032	100	3,032	3,032	100	3,032	3,032	100	3,032	3,032	100	3,032.0	3,032	3,032	3,032	100	3,032
frb30-15-5	3,011	100	3,011	3,011	100	3,011	3,011	100	3,011	3,011	100	3,011.0	3,011	3,011	3,011	100	3,011
frb35-17-1	3,650	100	3,650	3,650	100	3,650	3,650	100	3,650	3,650	100	3,650.0	3,650	3,650	3,650	100	3,650
frb35-17-2	3,738	100	3,738	3,738	96	3,736.8	3,738	100	3,738	3,738	100	3,738.0	3,738	3,737.7	3,738	100	3,738
frb35-17-3	3,716	100	3,716	3,716	100	3,716.0	3,716	100	3,716	3,716	100	3,716.0	3,716	3,716	3,716	100	3,716
frb35-17-4	3,683	100	3,683	3,683	77	3,678.3	3,683	100	3,683	3,683	100	3,683.0	3,683	3,683	3,683	100	3,683
frb35-17-5	3,686	100	3,686	3,686	100	3,686.0	3,686	100	3,686	3,686	100	3,686.0	3,686	3,686	3,686	100	3,686
frb40-19-1	4,063	100	4,063	4,063	83	4,062.1	4,063	96	4,062.8	4,063	100	4,063.0	4,063	4,060.3	4,063	100	4,063
frb40-19-2	4,112	100	4,112	4,112	87	4,111.2	4,112	100	4,112,0	4,112	100	4,112.0	4,112	4,112	4,112	100	4,112
frb40-19-3	4,115	100	4,115	4,115	19	4,108.3	4,115	46	4,111.72	4,115	99	4,114.9	4,115	4,114.8	4,115	100	4,115
frb40-19-4	4,136	100	4,136	4,136	89	4,135.5	4,136	98	4,135.92	4,136	98	4,135.9	4,136	4,136	4,136	100	4,136
frb40-19-5	4,118	100	4,118	4,118	90	4,117.6	4,118	88	4,117.52	4,118	100	4,118.0	4,118	4,118	4,118	100	4,118
frb45-21-1	4,760	100	4,760	4,760	44	4,748.6	4,760	58	4,754.3	4,760	98	4,759.8	4,760	4,756.9	4,760	100	4,760
frb45-21-2	4,784	100	4,784	4,784	47	4,775.8	4,784	100	4,784,0	4,784	100	4,784.0	4,784	4,783.9	4,784	100	4,748
frb45-21-3	4,765	100	4,765	4,765	26	4,756.9	4,765	88	4,764.76	4,765	90	4,764.8	4,765	4,765	4,765	100	4,765
frb45-21-4	4,799	100	4,799	4,799	43	4,772.4	4,799	96	4,797.24	4,799	100	4,799.0	4,799	4,799	4,799	100	4,799
frb45-21-5	4,779	100	4,779	4,779	82	4,777.3	4,779	100	4,779,0	4,779	100	4,779.0	4,779	4,778.9	4,779	100	4,779
frb50-23-1	5,494	20	5,487.9	5,494	6	5,484.7	5,494	11	5,486.41	5,494	4	5,485.2	5,494	5,484.7	5,494	64	5,491.4
frb50-23-2	5,462	15	5,452.6	5,462	3	5,434.1	5,462	5	5,440.22	5,462	9	5,451.9	5,462	5,454.7	5,462	100	5,462
frb50-23-3	5,486	100	5,486.0	5,486	53	5,480.2	5,486	98	5,485.98	5,486	57	5,485.2	5,486	5,483.2	5,486	100	5,486
frb50-23-4	5,454	28	5,453.3	5,454	9	5,451.6	5,454	14	5,453.14	5,453	91	5,452.5	5,453	5,447.5	5,454	100	5,454
frb50-23-5	5,498	100	5,498.0	5,498	89	5,495.7	5,498	100	5,498,0	5,498	100	5,498.0	5,498	5,491.9	5,498	100	5,498
frb53-24-1	5,670	43	5,660.4	5,670	5	5,637.9	5,670	13	5,652.18	5,670	33	5,661.4	5,670	5,664.8	5,670	100	5,670
frb53-24-2	5,707	25	5,694.3	5,707	6	5,676.5	5,707	3	5,685.32	5,707	1	5,685.3	5,707	5,696.3	5,707	100	5,707
frb53-24-3	5,640	90	5,639.4	5,640	15	5,610.7	5,640	48	5,629.38	5,655	3	5,636.5	5,655	5,631.5	5,640	100	5,640
frb53-24-4	5,714	25	5,700.7	5,714	7	5,645.6	5,714	13	5,676.16	5,714	4	5,696.9	5,714	5,700.2	5,714	64	5,704.4
frb53-24-5	5,659	6	5,653.1	5,659	5	5,628.7	5,659	4	5,642.5	5,659	1	5,651.4	5,659	5,647.2	5,659	60	5,658.2
frb56-25-1	5,916	19	5,877.3	5,916	3	5,836.8	5,916	5	5,860.82	5,916	59	5,906.5	5,916	5,895.7	5,916	82	5,904.5
frb56-25-2	5,886	3	5,861.3	5,872	1	5,807.7	5,886	1	5,838.96	5,886	9	5,873.0	5,886	5,877.3	5,872	73	5,864.3
frb56-25-3	5,859	1	5,831.6	5,859	1	5,799.3	5,859	1	5,811.0	5,859	1	5,832.3	5,859	5,820.5	5,859	76	5,849.3
frb56-25-4	5,892	5	5,869.3	5,892	3	5,839.1	5,892	12	5,860.86	5,892	2	5,866.1	5,892	5,856.3	5,892	82	5,891.4
frb56-25-5	5,853	1	5,811.5	5,839	1	5,768.3	5,853	1	5,787.04	5,841	1	5,812.2	5,853	5,818.2	5,839	85	5,834.6
frb59-26-1	6,591	67	6,585.1	6,591	3	6,547.5	6,591	17	6,571.6	6,591	20	6,578.7	6,591	6,575.4	6,591	52	6,583.3
frb59-26-2	6,645	40	6,614.5	6,645	3	6,567.1	6,645	13	6,602.34	6,645	13	6,589.1	6,645	6,591.2	6,645	46	6,617.4
frb59-26-3	6,608	1	6,567.5	6,608	1	6,514.1	6,608	1	6,542.74	6,608	1	6,579.1	6,608	6,583.6	6,608	6	6,582.7
frb59-26-4	6,592	5	6,533.5	6,592	1	6,498.3	6,592	6	6,526.5	6,592	71	6,585.1	6,592	6,576.7	6,592	8	6,542.1
frb59-26-5	6,584	9	6,554.5	6,584	1	6,522.5	6,584	5	6,546.94	6,584	3	6,558.5	6,584	6,552.8	6,584	6	6,562.4
Average	4,903.7	65.1	4,894.2	4,903.0	45.0	4,877.9	4,903.7	56.0	4,888.1	49,03.8	61.7	4,895.6	4,904.1	4,894.5	4,903.9	**85.1**	4,897.6

Table 5: Execution times of state-of-the-art algorithms for selected problem instances (units are given in seconds).

instance	BQP-PTS	PLS	MN/TS	BLS	ReTS - I	ILS-VND	PTC
brock200_1	0.02	0.19	0.01	0.01	0	0.01	0.7
brock800_4	105.53	3.77	49.7	339.07	835.03	25.4	0.4
C125.9	0.02	8.08	0.02	0.01	0	0.01	0.8
C4000.5	19,902.77	-	80.56	179.89	116.05	216.1	96.4
DSJC500.5	3.82	0.95	0.04	-	0.13	0.03	0.1
DSJC1000.5	115.42	47.76	0.2	-	0.38	0.16	0.1
keller4	0.05	0.02	0.03	0.04	0	0.01	0.1
keller6	3,418.36	-	606.15	1,980.16	532.74	53.3	610.4
MANN_a9	0.01	0.01	0.01	-	0	0.01	0.1
MANN_a81	6,167.28	-	832.24	2,942.54	990.02	74.99	954.8
hamming6-2	0.01	0.01	0.001	0.01	0	0.01	0
hamming10-4	32.49	1,433.07	2.21	26.86	26.25	6.83	1.2
gen200_p0.9_44	0.02	4.44	0.01	0.01	0	0.01	0
gen400_p0.9_75	0.67	0.01	0.88	0.43	0.03	0.01	0.3
c-fat200-1	0.01	0.01	0.14	-	0	0.01	0.2
c-fat500-10	1.29	0.01	0.06	-	0.11	0.01	0
johnson8-2-4	0.01	0.01	0.01	0.01	0	0.01	0
johnson32-2-4	26.71	44.68	0.53	0.48	0.04	0.01	0.3
p_hat300-1	0.03	0.01	0.02	0.01	0	0.01	0
p_hat1500-3	34.14	-	188.38	1.78	2.06	0.06	246.3
san200_0.7_1	0.06	0.01	0.17	30.65	0.21	0.01	0.1
san400_0.9_1	0.31	0.01	1.29	6.25	2.38	11.4	0.2
san1000	40.93	-	13.01	4.94	71.07	0.13	8
sanr200-0.7	0.08	0.62	0.01	0.01	0	0.01	0
san400_0.7_3	42.54	4.41	0.05	0.05	0.41	0.1	0
frb30-15-1	4.9	-	0.35	1.12	1.43	1	0.4
frb30-15-5	2.13	-	3.01	3.64	0.8	0.05	3.2
frb35-17-1	6.59	-	25.8	68.45	5.1	5.66	26
frb35-17-5	3.73	-	8.09	20	2.7	1.07	9.5
frb40-19-1	87.72	-	85.57	291.14	51.68	15.57	96.2
frb40-19-5	96.63	-	178.89	343.82	34.72	2.62	199.5
frb45-21-1	896.25	-	126.26	982.32	161.39	18.98	144.6
frb45-21-5	34.17	-	193.82	206.6	11.23	11.67	212.3
frb50-23-1	1,911.49	-	186.62	1,221.72	154.05	52.83	188.4
frb50-23-5	751.84	-	110.85	388.18	118.21	21.2	124.1
frb53-24-1	981.33	-	233.22	1,056.82	349.95	20.27	246.3
frb53-24-5	2,802.83	-	294	278.91	777.93	10.97	304.2
frb56-25-1	1035	-	308.9	1,764.87	344.18	33.19	322.4
frb56-25-5	3,549.57	-	322.7	4,386.6	354.28	2.88	344.6
frb59-26-1	2,228.21	-	166.2	1,435.99	521.08	28.84	178.6
frb59-26-5	747.8	-	161.47	1,512.09	320.54	76.46	184.2
Average	1,098.4	77.4	102.0	541.0	141.1	16.9	109.9