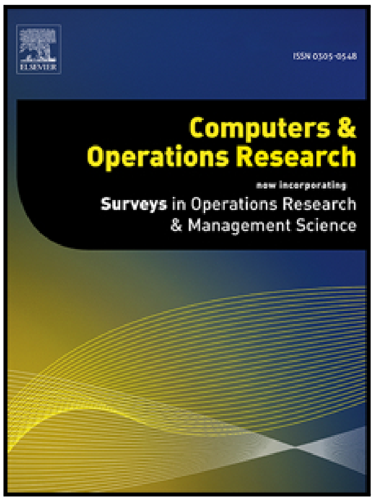
Accepted Manuscript



A CPU-GPU Local Search Heuristic for the Maximum Weight Clique Problem on Massive Graphs

Bruno Nogueira, Rian G.S. Pinheiro

PII: S0305-0548(17)30257-5
DOI: 10.1016/j.cor.2017.09.023
Reference: CAOR 4335

To appear in: *Computers and Operations Research*

Received date: 10 October 2016
Revised date: 22 September 2017
Accepted date: 25 September 2017

Please cite this article as: Bruno Nogueira, Rian G.S. Pinheiro, A CPU-GPU Local Search Heuristic for the Maximum Weight Clique Problem on Massive Graphs, *Computers and Operations Research* (2017), doi: 10.1016/j.cor.2017.09.023

Highlights

- We propose a new neighborhood structure for the problem. The results demonstrate that our neighborhood structure is better than the current ones, and it has the additional benefit that it can be explored using a GPU-based massivelly parallel architecture.
- We are the first to study the use of a GPU on the problem. The results indicate that an up to 12x speedup can be achieved.
- We compare our heuristic with the state-of-the-art ones and show that, even when the heuristic executes without using a GPU, it outperforms them. Moreover, the results also indicate that GPULS

A CPU-GPU Local Search Heuristic for the Maximum Weight Clique Problem on Massive Graphs

Bruno Nogueira[a,*], Rian G. S. Pinheiro[a]

[a] *Universidade Federal Rural de Pernambuco, Unidade Acadêmica de Garanhuns, Brazil*

Abstract

Given an undirected graph with positive weights on the vertices, the Maximum Weight Clique Problem (MWCP) consists in finding a clique with maximum total weight. In this paper, we present a CPU-GPU local search heuristic for solving the MWCP on massive graphs. The heuristic is based on two new neighborhood structures for the problem. These neighborhoods are explored using an efficient procedure that is suitable to be mapped onto a GPU-based massively parallel architecture. We test the proposed heuristic on real-world massive graphs with millions of edges and vertices. The results indicate that, even when the heuristic is executed on a CPU-only architecture, it is able to outperform the best-known heuristics for the MWCP. Moreover, the hybrid CPU-GPU implementation obtained an average speedup of up to 12 times over the CPU-only implementation.

Keywords: Combinatorial Optimization, Clique, Maximum Weight Clique, GPU, Local Search, Metaheuristics

1. Introduction

Given an undirected graph $G = (V, E)$, a clique C of G is a subset of V whose elements are pairwise adjacent, i.e., $\forall v_i, v_j \in C, (v_i, v_j) \in E$. The Maximum Clique Problem (MCP) is a classical combinatorial problem [1], it aims at finding a maximum-cardinality clique of G. In this work, we focus on a prominent generalization of this problem known as the Maximum Weight Clique Problem (MWCP), which consists of: given a weighting function $w : V \to \mathbb{R}_{>0}$ that associates with each vertex $v \in V$ a positive value, determine a clique $C \subseteq V$ with maximum total weight, $Weight(C) = \sum_{v \in C} w(v)$. The MWCP has important applications in many different domains, such as combinatorial auctions [2], pattern recognition/robotics [3, 4], community (cluster) detection [5], portfolio selection [6], bioinformatics [7] and scheduling [8]. Very often data coming from these domains are massive, hence efficient methods for solving very large MWCP instances are required.

*Corresponding author

Email addresses: nogueirabruno@gmail.com; bruno.nogueira@ufrpe.br (Bruno Nogueira), rian.gabriel@ufrpe.br (Rian G. S. Pinheiro)

Both MCP and MWCP are known to be $\mathcal{NP}$-hard for general graphs [1]. Moreover, research on computational complexity has shown that they are also hard to approximate. The best-known polynomial-time approximation algorithm for MCP has an approximation factor of $n.(\log\log n)^2/(\log n)^3$ [9]. Besides, it has been demonstrated that it is $\mathcal{NP}$-hard to approximate MCP within a factor better than $n^{1-\epsilon}$, for any $\epsilon > 0$ [10]. These results indicate that the MWCP is indeed very difficult to solve. Hence, for large and dense instances, heuristics are commonly adopted to obtain good solutions within reasonable execution times.

The state-of-the-art heuristics for the MWCP use local search as their main ingredient [11, 12, 13, 14, 15, 16]. These heuristics start from a candidate clique which is then iteratively transformed into a new clique by one of the following move operators: (i) *(0,1)-swap*, in which one vertex is inserted into the clique; (ii) *(1,0)-swap*, in which one vertex is removed from the clique; (iii) *(1,1)-swap*, in which one vertex in the clique is exchanged by another vertex not in the clique, and (iv) *(ω,1)-swap*, in which one vertex is added and all vertices that are not adjacent to the inserted vertex are removed from the clique. We remark that these move operators appeared in the literature under different names. For instance, the moves *(0,1)-swap* and *(1,0)-swap* are called *ADD* and *DROP* in [12], whereas the move *(ω,1)-swap* is called *PUSH* in [15].

Apart from the local search based heuristics, other representative approaches for the problem include a genetic algorithm [17], a heuristic based on replication dynamics [18], and an approach based on a binary quadratic programming model that is solved by a tabu search heuristic [19]. Cai and Lin [20] proposed a heuristic that relies on graph reductions to solve large and sparse MWCP instances. Their heuristic alternates between clique construction and graph reduction. The reduction procedure tries to find (and remove) vertices that are impossible to be in any clique of the optimal weight. Recently, Jiang et al. [21] proposed an exact branch-and-bound algorithm that also targets large and sparse instances.

This work presents a new CPU-GPU local search heuristic for solving the MWCP on massive graphs. We consider massive graphs as graphs with at least 1000 vertices. Our approach is based on two new neighborhood structures for the problem. These neighborhoods are evaluated and explored using an efficient procedure that is suitable to be mapped onto a GPU-based massively parallel architecture. Another feature of the heuristic is the use of the reduction procedures proposed in [20]. We test our heuristic on real-world massive graphs with millions of edges and vertices. The results indicate that, even when the heuristic is executed on a CPU-only architecture, it is able to outperform the best-known heuristics for the MWCP. Moreover, the hybrid CPU-GPU implementation obtained an average speedup of up to 12 times over the CPU-only implementation. To the best of our knowledge, we are the first to investigate the use of a massively parallel architecture on the MWCP.

The remainder of the paper is organized as follows. Section 2 describes the proposed neighborhood structures. Section 3 details our CPU-GPU local search heuristic. Section 4 shows computational results for test problems from the literature, and Section 5 concludes the paper.

2. Neighborhood structures

Local search algorithms are based on the iterative exploration of the solution space: at each iteration, the algorithm moves from the current solution to a neighbor solution. This work proposes two neighborhood structures for the MWCP: $\mathcal{N}_1$ and $\mathcal{N}_2$. The neighborhood $\mathcal{N}_1$ of a candidate clique C is given by the set of new cliques that can be generated by applying to C one of the following moves: *(0,1)-swap*, *(1,0)-swap*, and *(ω, 1)-swap*. On the other hand, the neighborhood $\mathcal{N}_2$ of C is generated by applying to C the *(1,2)-swap* move, which consists in removing one vertex from C and inserting another two into it.

Let $G = (V, E, w)$ be a MWCP instance, Ω the solution space (i.e., the set of all possible cliques of G), and $N(v) \subseteq V$ the adjacent vertices of vertex $v \in V$. Given a clique $C \in \Omega$, we can partition the vertices of graph G into three (disjoint) sets: C, $V_C^\bullet$, and V_C°, where

$$\begin{aligned} V_C^\bullet &= \{v : v \in V \setminus C, |N(v) \cap C| = |C|\} \text{ and} \\ V_C^\circ &= \{v : v \in V \setminus C, |N(v) \cap C| < |C|\}. \end{aligned} \tag{1}$$

Note $V_C^\bullet$ is the set of vertices that are not in the clique C, but are adjacent to all vertices in C. Besides, every vertex $v \in V_C^\circ$ is not adjacent to at least one vertex in the clique C. Given any $v \in C$, we also denote by $C_v \subseteq V_C^\circ$ the set of vertices that are adjacent to all vertices of $C \setminus \{v\}$, i.e., $C_v = \{y : |N(y) \cap C \setminus \{v\}| = |C| - 1, v \in C, y \in V_C^\circ\}$.

The move operators *(0,1)-swap*, *(1,0)-swap*, *(ω, 1)-swap*, and *(1, 2)-swap* are defined as

$$\begin{aligned} \textit{(0,1)-swap}(v, C) &= C \cup \{v\} && \forall v \in V_C^\bullet, \\ \textit{(1,0)-swap}(v, C) &= C \setminus \{v\} && \forall v \in C, \\ \textit{(}\omega\textit{,1)-swap}(v, C) &= (C \cup \{v\}) \setminus (C \setminus N(v)) && \forall v \in V_C^\circ, \\ \textit{(1,2)-swap}(v, x, y, C) &= (C \setminus \{v\}) \cup \{x, y\} && \forall v \in C, \forall (x, y) \in E, \{x, y\} \subseteq C_v, x, y \notin C; \end{aligned} \tag{2}$$

and the proposed neighborhoods are given by:

$$\begin{aligned} \mathcal{N}_1(C) =& \{\textit{(0,1)-swap}(v, C) : v \in V_C^\bullet\} \cup \\ & \{\textit{(1,0)-swap}(v, C) : v \in C\} \cup \\ & \{\textit{(}\omega\textit{,1)-swap}(v, C) : v \in V_C^\circ\}, \end{aligned} \tag{3}$$

$$\mathcal{N}_2(C) = \{\textit{(1,2)-swap}(v, x, y, C) : v \in C, (x, y) \in E, \{x, y\} \subseteq C_v\}. \tag{4}$$

From Definitions (2) and (3), it is possible to observe that each vertex $v \in V$ is associated with exactly one of the move operators that generate the neighborhood $\mathcal{N}_1$, hence we can associate with each vertex a move gain (i.e., the weight change after the corresponding move is applied on the clique C):

$$\Delta_v^{\mathcal{N}_1} = \begin{cases} w(v), & \text{if } v \in V_C^\bullet \\ -w(v), & \text{if } v \in C \\ w(v) - \sum_{v' \in C \setminus N(v)} w(v') & \text{if } v \in V_C^\circ. \end{cases} \tag{5}$$

Definition (5) indicates the *(1,0)-swap* move always lead to a negative improvement, which might seem counterintuitive. However, non-improving moves can be very useful to help a heuristic scape from local minima [22].

Figure 1a and the table in Figure 1b depicts a MWCP instance as well as a candidate clique C. Column *(a)* in Figure 1b shows the move gain associated with each vertex, according to Definition (5). As shown in Figure 1c, the move *(1,0)-swap*(v_8, C) removes vertex v_8 and returns clique C'. The resulting clique C'' after the move *(ω,1)-swap*(v_2, C') is illustrated in Figure 1d. Finally, in Figure 1e, the clique C''' is obtained after the insertions of vertices v_3 and v_1. Columns *(c)–(e)* in Figure 1b show the corresponding vertex gain after each move indicated in Figures 1c–1e.

The move gain of the move operator *(1,2)-swap* is:

$$\Delta^{\mathcal{N}_2}_{v,x,y} = -w(v) + w(x) + w(y) \quad \forall v \in C, \forall (x,y) \in E, \{x,y\} \subseteq C_v. \tag{6}$$

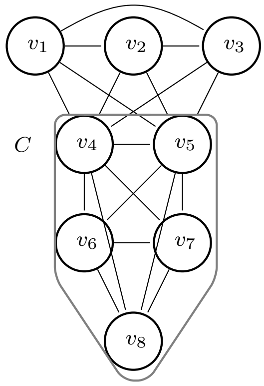
An example of the *(1,2)-swap* operator is illustrated in Figure 2, in which the original clique C is modified by removing vertex v_1 and adding vertices v_4 and v_5.

3. CPU-GPU local search heuristic

This section presents the proposed CPU-GPU local search heuristic. Our approach leverages the GPU massively parallel power and high memory bandwidth to efficiently solve the MWCP. Algorithm 1 shows the pseudocode of the proposed heuristic, called GPULS (GPU Local Search). Operations related to the GPU are denoted as << *operation* >> (see, e.g., lines 2, 7, and 8).

GPULS is a multi-start local search heuristic [23] in which the initial solution is randomly generated. The heuristic considers the two neighborhood structures described in the previous section, namely $\mathcal{N}_1$ and $\mathcal{N}_2$. GPULS adopts a tabu search approach [22] for exploring $\mathcal{N}_1$: it selects from the available moves of $\mathcal{N}_1$, the move with maximum gain that is not on the tabu list (even if the gain is negative). The tabu list is used to prevent the heuristic from reversing recent moves and to ensure diversification. The neighborhood $\mathcal{N}_2$, on the other hand, is explored in a first-improvement fashion, i.e., GPULS selects from the available moves of $\mathcal{N}_2$ the first improving move (if such a move exists). Moreover, when the best clique found so far is improved, the heuristic uses the two vertex removal procedures proposed in [20] to reduce the problem instance.

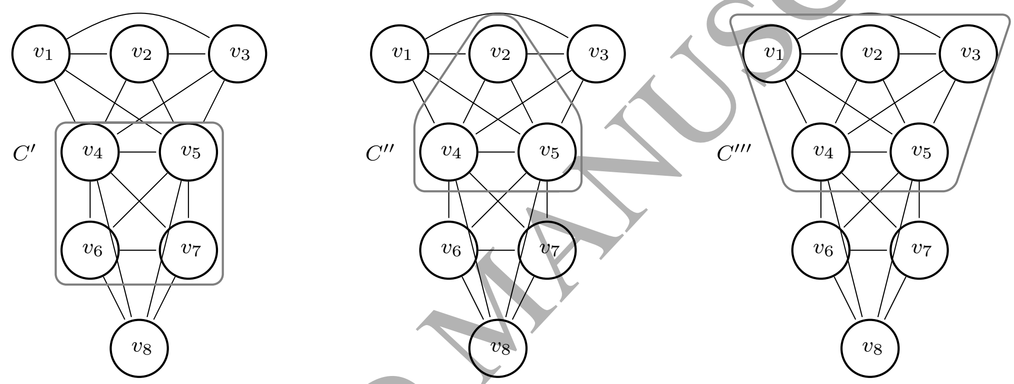
Our heuristic starts by initializing on the GPU the data structures that represent the MWCP instance G (line 2 of Algorithm 1). These data structures and the other ones adopted by the heuristic are further detailed in Subsection 3.3. Next, the best clique found so far C_{best} is initialized and the outer loop is executed (lines 3-27). At each iteration of this loop, the heuristic defines a new current clique C and sets this clique as the new local best clique $C_{localbest}$ (lines 5 and 6). C is randomly initialized, i.e., initially $C = \emptyset$, and then the heuristic iteratively adds to C a randomly chosen vertex that is adjacent to all vertices already in C until no such vertex exists. Next, C, $C_{localbest}$ and an empty tabu list are copied to the GPU memory (lines 7 and 8).



(a) Initial clique C, $Weight(C) = 27$

Vertex	Weight	Move Gain			
		(a)	(c)	(d)	(e)
v_1	5	-12	-7	5	-5
v_2	8	-9	-4	-8	-8
v_3	5	-12	-7	5	-5
v_4	5	-5	-5	-5	-5
v_5	5	-5	-5	-5	-5
v_6	6	-6	-6	-2	-12
v_7	6	-6	-6	-2	-12
v_8	5	-5	5	-3	-13

(b) Move Gain

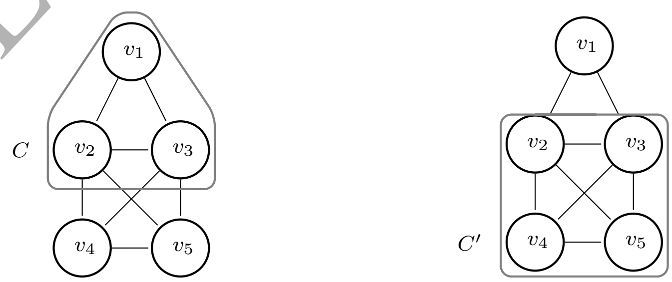


(c) *(1,0)-swap*(v_8, C), $Weight(C') = 22$

(d) $(\omega, 1)$-*swap*(v_2, C'), $Weight(C'') = 18$

(e) *(0,1)-swap*$(v_1$,*(0,1)-swap*$(v_3, C''))$, $Weight(C''') = 28$

Figure 1: Move gain after a sequence of move operations of the neighborhood $\mathcal{N}_1$.



(a) Initial clique C

(b) *(1,2)-swap* (v_1, v_4, v_5, C)

Figure 2: Move operation of the neighborhood $\mathcal{N}_2$.

Input : Weighted graph $G = (V, E, w)$
Output: A maximal clique C_{best}

```
GPULS(G)
    << Copy G to GPU memory >>
    C_best = ∅                                           // Best clique found so far
    while (stop criterion) do
        C = RandomSolution(G)                            // Generate a random clique
        C_localbest = C                                   // Local best clique found
        << Copy C and C_localbest to GPU memory >>
        << Create an empty tabu list on GPU memory >>
        no_improv = 0    // Number of consecutive iterations without improvement
        while (no_improv < K_1) do
            // Find in neighborhood N_1 of C the move with maximum gain that is
               not on the GPU tabu list (Subsection 3.1)
            C' = TabuIteration(C, G, Weight(C_localbest) , iter)
            if (Weight(C') > Weight(C)) then
                C_localbest = C', no_improv = 0
                << Update C_localbest on GPU memory >>
            else
                no_improv = no_improv + 1
                if (no_improv mod K_2 = 0) then
                    // Find an improving move in neighborhood N_2 of C_localbest
                       (Subsection 3.2)
                    C' = SwapLocalSearch(C_localbest, G)
                    while (Weight(C') > Weight(C_localbest)) do
                        no_improv = 0, C_localbest = C'
                        << Update C_localbest on GPU memory >>
                        C' = SwapLocalSearch(C_localbest, G)
                    // Set the local best solution as the current solution and
                       reset the tabu list
                    C = C_localbest
                    << Update C and reset the tabu list on GPU memory >>
        if (Weight(C_localbest) > Weight(C_best)) then
            C_best = C_localbest, G = GraphReduce(G, C_best)
            << Update G on GPU memory >>
    return C_best
```

Algorithm 1: GPU-CPU heuristic for the MWCP

The inner loop (lines 10-24) is responsible for the local search on neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_2$. It repeats until no improvement in $C_{localbest}$ is obtained after K_1 consecutive iterations. At each inner loop iteration, function *TabuIteration* tries to find in the neighborhood $\mathcal{N}_1$ of C the move with maximum gain that is not on the tabu list (line 11). The heuristic then applies the selected move, and if the new current clique improves the local best clique $C_{localbest}$, the former is set as the new local best clique (lines 12-14). After K_2 consecutive inner loop iterations without improvement in $C_{localbest}$, the heuristic iteratively tries to find improving moves in neighborhood $\mathcal{N}_2$ of $C_{localbest}$ (lines 18-22). After trying to find improvements in neighborhood $\mathcal{N}_2$, $C_{localbest}$ is set as the new current solution and the tabu list is restarted (lines 23 and 24). We remark that after preliminary tests, we observed that the values $K_1 = 500$ and $K_2 = 100$ yielded a good trade-off between solution quality and CPU time. Hence, we used this setting in our experiments.

Finally, at the end of the outer loop, the heuristic updates the best clique found C_{best}, and applies the two graph reduction procedures proposed in [20] (lines 25-27). If the graph G becomes empty after the reduction procedures, then the best found solution C_{best} is proved to be optimal.

3.1. Neighborhood $\mathcal{N}_1$ exploration and tabu list management

The neighborhood $\mathcal{N}_1$ is explored by the function *TabuIteration* (line 11 of Algorithm 1), which selects the move with maximum gain that is not on the tabu list. The heuristic updates the tabu list in the following way: when a vertex v is added to the current clique by a *(0,1)-swap* move, v is forbidden to be removed from the clique by a *(1,0)-swap* move for the next γ_1 iterations. Similarly, if v is removed from the clique by a *(1,0)-swap* move, it cannot be added back by a *(0,1)-swap* or *(ω, 1)-swap* move in the next γ_2 iterations. When a *(ω, 1)-swap* move is applied, the removed vertices are forbidden be added back by the moves *(0,1)-swap* or (ω, 1)-swap for the next γ_3 iterations, but the inserted vertex can be removed without restrictions. Based on previous research [12, 13], we empirically defined the following values for γ_1, γ_2, and γ_3: $\gamma_1 = 7$, $\gamma_2 = 5$, and $\gamma_3 = 7 + random(|C|)$, where *random* is a function that returns at random a value ranging from 1 to $|C|$. We revoke the tabu restrictions when the move on the tabu list improves the current local best weight (this approach is known as aspiration criterion [22]).

Besides not allowing moves that are on the tabu list, we also forbid *(ω, 1)-swap* moves when the number of vertices to be removed is greater than one. This restriction is also revoked if the corresponding *(ω, 1)-swap* improves the current local best weight. We adopt this approach because when the number of vertices removed by a *(ω, 1)-swap* is too high, this move may become computationally expensive and it may also destroy the good characteristics of the current clique.

We will now show how the moves that generate the neighborhood $\mathcal{N}_1$ of C can be evaluated in $\mathcal{O}(1)$ time. Since the gains of the moves *(0,1)-swap(v,C)* and *(1,0)-swap(v,C)* are simply $w(v)$ and $-w(v)$, it is easy to see that such moves can be evaluated in $\mathcal{O}(1)$. To evaluate the gain of a *(ω, 1)-swap* move in $\mathcal{O}(1)$ time, we proceed as follows: let μ be an array that stores, for each vertex $v \in V$, the weight of v plus the sum of the weights of all adjacent vertices of v that are in the current clique C, i.e., $\mu[v] = w(v) + \sum_{y \in C \cap N(v)} w(y)$.

Initially, we set $\mu[v] = w(v)$, $\forall v \in V$; and whenever a vertex is inserted into or deleted from the clique, μ is updated as follows: to insert (delete) a vertex y to (from) C, for each adjacent vertex y of v, we increase (decrease) $\mu[y]$ by $w(v)$. Thus, given array μ, the move gain of a *(ω, 1)-swap(v,C)* move can be evaluated in $\mathcal{O}(1)$ time with the following expression: $\mu[v] - Weight(C)$.

Algorithm 2 outlines the *TabuIteration* function, which has been parallelized on the GPU. This function is composed of a series of GPU kernel calls. A GPU kernel is a function callable from the CPU and executed, simultaneously by many threads in parallel, on the GPU device. The proposed kernels are detailed in Subsection 3.3.

Input : current clique C, weighted graph $G = (V, E, w)$, local best weight $best\_weight$, iteration number $iter$
Output: A maximal clique C'

1 `TabuIteration`$(C, G, best\_weight, iter)$
2   << Launch `MoveGainEval` kernel to calculate the move gain associated with each vertex >>
3   << Launch `Reduce` kernel to obtain the vertex with maximum move gain max_v >>
4   << Copy max_v from the GPU memory >>
5   **if** $(max\_v \in C)$ **then**
6     $C' =$ *(1,0)*-`swap`(max_v, C)
7   << Launch `Deletion` kernel to remove vertex max_v and update the tabu list on the GPU >>
8   **else if** $((max\_v \in V \setminus C) \wedge (|N(max\_v) \cap C| = |C|))$ **then**   `// if` $max\_v \in V_C^{\bullet}$
9     $C' =$ *(0,1)*-`swap`(max_v, C)
10   << Launch `Insertion` kernel to add vertex max_v and update the tabu list on the GPU >>
11   **else**   `// if` $max\_v \in V_C^{\circ}$
12     $C' =$ *(ω,1)*-`swap`(max_v, C)
13   **foreach** $(v \in C \setminus N(max\_v))$ **do**
14   << Launch `Deletion` kernel to remove vertex v and update the tabu list >>
15   << Launch `Insertion` kernel to add vertex max_v >>
16   **return** C'

Algorithm 2: Neighborhood $\mathcal{N}_1$ exploration algorithm

In the beginning of the *TabuIteration* function, the call to the *MoveGainEval* kernel makes the GPU evaluate each candidate move of the neighborhood $\mathcal{N}_1$ of C (line 2). As will be shown in Subsection 3.3, the *MoveGainEval* kernel adopts the $\mathcal{O}(1)$ procedure described earlier to evaluate such moves. Next, the *Reduce* kernel is called to read the output of the *MoveGainEval* kernel and, by means of a parallel reduction [24], select the vertex associated with the maximum move gain (line 3). The selected vertex is then copied from the GPU memory to the CPU memory (line 4). Finally, the move associated with the selected vertex is used to update the current clique on both CPU and GPU memories (lines 5-15). The

Insertion and *Deletion* kernels are responsible for updating the clique on the GPU memory. Except for the few bytes transferred at line 3, no other data transfer occurs between the CPU and GPU in the *TabuIteration* function. Hence, the proposed implementation is almost free of one of the major bottlenecks of CPU-GPU programs: the overhead due to the CPU-GPU communication.

3.2. Neighborhood $\mathcal{N}_2$ exploration

The neighborhood $\mathcal{N}_2$ is explored through the function *SwapLocalSearch*, which tries to find a *(1,2)-swap* move that leads to an improvement in the current local best solution $C_{localbest}$ (lines 18-22 of Algorithm 1). To find an improving *(1,2)-swap*, we need to find a vertex $v \in C$ and two vertices $x, y \notin C$, such that the removal of v and the insertion of x and y would lead to a clique with greater weight. Given a maximal clique, such a move only exists if the following holds [25]: (i) $w(x)+w(y) > w(v)$; (ii) x and y are adjacent, (iii) both x and y are adjacent to all vertices in $C \setminus v$; and (iv) neither x nor y are adjacent to v (or else C would not be maximal). Algorithm 3 describes the proposed approach for finding an improving *(1,2)-swap*.

The algorithm starts by copying from the GPU memory an array denoted by τ. This array stores, for each vertex $v \in V$, the number of vertices in the clique that are adjacent to v, i.e., $\tau[v] = |N(v) \cap C|$. Whenever a vertex is inserted or removed from the clique, this array is updated by the *Insertion*/*Deletion* GPU kernels as follows: to insert (remove) a vertex v to (from) the clique, for each adjacent vertex y of v, the GPU increases (decreases) $\tau[y]$ by one. After copying the τ array, for each vertex $v \in C$, we define two data structures: L_v and A_v (lines 3-13). L_v is a list that stores the vertices that are not adjacent to v but are adjacent to all other vertices in C; and A_v is an array that indicates if a vertex v_i is included in L_v, i.e., $A_v[v_i] = 1$ if the vertex is in L_v, and $A_v[v_i] = 0$, otherwise. The algorithm assumes the vertices are labeled from 1 to $|V|$, and that the adjacency list of any vertex is sorted by increasing order of labels. After creating L_v and A_v, the size of L_v is checked. If its size is not greater than one, v cannot be part of a *(1,2)-swap* (lines 14 and 15). At lines 16-23, the algorithm tries to find two vertices $x, y \in L_v$, such that $y \in N(x)$ and $w(x) + w(y) > w(v)$. If such vertices are found, the algorithm removes v and inserts x and y (lines 19-22).

Algorithm 3 indicates that finding an improving *(1,2)-swap* consists of two steps: (i) constructing L_v and A_v for each vertex $v \in C$ (lines 6-13), which takes $\mathcal{O}(|C||V|)$ time, and (ii) traversing the adjacency list of each vertex $x \in \{x' : x' \in V \setminus C, \tau[x'] = |C| - 1\}$ (lines 16-18), which takes no more than $\mathcal{O}(|E|)$ time as each adjacency list is traversed at most once. Therefore, the proposed algorithm can find an improving *(1,2)-swap* move (or prove no such a move exists) in $\mathcal{O}(|C||V| + |E|)$ time.

3.3. GPU kernels

This subsection describes the proposed GPU kernels. On the GPU memory, the topology of the MWCP instance $G = (V, E, w)$ is stored as the well-known Compressed Sparse Row (CSR) graph format, which is a compact and efficient encoding scheme. CSR consists of two arrays E_a and V_a with sizes $|E|$ and $|V|$, respectively. E_a contains the concatenation of the adjacency lists of the graph, and V_a contains the indices indicating where each adjacency

Input : Local best clique C, weighted graph $G = (V, E, w)$
Output: A maximal clique C'

```
SwapLocalSearch(C, G)
    << Copy array τ from GPU memory to CPU memory (recall this array mantains
    τ[v] = |N(v) ∩ C|, ∀v) >>
    foreach (v ∈ C) do
        Let A_v be an array of size |V|
        L_v = ∅
        for (i = 1, j = 1; i ≤ |V|; i = i + 1) do
            A_v[v_i] = 0
            if (v_i = N(v, j)) then          // N(v, j) is the j-th adjacent vertex of v
                j = j + 1
            else
                if (τ[v_i] = |C| − 1 ∧ v_i ≠ v) then
                    L_v = v_i ∪ L_v
                    A_v[v_i] = 1
        if (Size(L_v) ≤ 1) then
            foreach (x ∈ L_v) do
                foreach (y ∈ N(x)) do
                    if ((A_v[y] = 1) ∧ (w(v) ≥ w(x) + w(y)) then
                        C' = (1,2)-swap(v, x, y, C)
                        << Launch Deletion kernel to remove vertex v >>
                        << Launch Insertion kernel to add vertex x >>
                        << Launch Insertion kernel to add vertex y >>
                        return C'
    return C
```

Algorithm 3: Neighborhood $\mathcal{N}_2$ exploration algorithm list starts. Figure 3 depicts an example. The weight of each vertex is stored in an additional array, denoted by V_w. Due to the reduction procedures, the graph instance might be reduced, hence, we also maintain the array V_{up}, which stores the vertices that have not been removed yet.

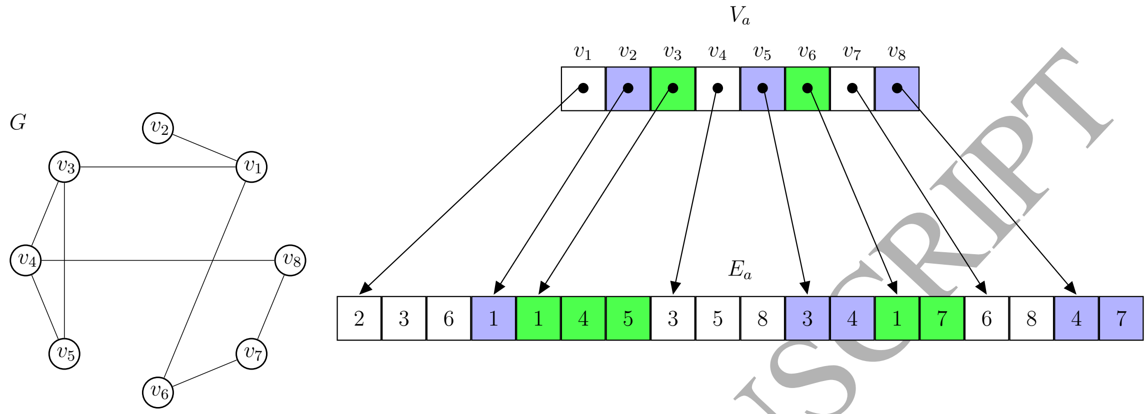


Figure 3: Graph G and its adjacent list representation.

A candidate clique C is represented by an binary array, called clique array C_a, in which the i-th array entry, $i \in \{1, ..., |V|\}$, indicates whether or not vertex v_i is in the clique. The tabu list TL is also an array, and it stores for each vertex $v \in V$ the least iteration v can be used again. Thus, to add vertex v to the tabu list for the next, say 5 iterations, we need to set $TL[v] = iter + 5$, where $iter$ denotes the current iteration number. To determine if a vertex v is in the tabu list, we simply check if the following expression evaluates to true: $TL[v] < iter$.

It should be highlighted that, whenever a vertex is inserted or removed from the clique, only the clique array C_a is updated on both CPU and GPU. The other adopted data structures are exclusively updated on the GPU by the *Insertion* and *Deletion* kernels.

Algorithm 4 shows the *MoveGainEval* kernel, which is used to evaluate each candidate move of the neighborhood $\mathcal{N}_1$ (line 2 of Algorithm 2). We remark that, in the presented kernels pseudocode, arguments that represent arrays are actually C pointers to the locations in GPU memory where the corresponding arrays are stored. Through the *MoveGainEval* kernel, each GPU thread evaluates the move gain associated with a single vertex. First, the gain is calculated (lines 2-10), and then the result is stored (lines 11-14). The *if* statement at line 11 indicates that a penalty is added to the gain if one of the following two cases occurs: (i) the number of removed vertices by a *(ω, 1)-swap* move is greater than one, (ii) the vertex is in the tabu list. As the *if* statement shows, the penalty is revoked if the candidate move improves the local best solution weight.

Algorithm 5 presents the *Deletion* kernel, which is used to remove a vertex from the clique. With this kernel, each GPU thread updates a single entry of the arrays τ and μ (lines 5 and 6). This kernel also updates the clique array C_a and the tabu list TL (lines 7-9). We omit the *Insertion* kernel because of its similarity with the *Deletion* kernel.

Input : active vertices array V_{up}, weight array V_w, clique array C_a, array μ (maintains $\mu[v] = w(v) + \sum_{y \in C \cap N(v)} w(y), \forall v$), array τ (maintains $\tau[v] = |N(v) \cap C|, \forall v$), local best weight $best\_weight$, current weight $weight$, current clique size $size$, tabu list TL, iteration counter $iter$, move gain array $gain$

Output: Inserts into idx-th entry of array $gain$ the move gain associated with vertex $V_{up}[idx]$

`MoveGainEval(`V_{up}, V_w, C_a, μ, τ, $best\_weight$, $weight$, $size$, TL, $iter$, $gain$`)`
- $idx =$ `ThreadId() // Get thread id`
- $v = V_{up}[idx]$
- `// Compute the gain of the move associated with vertex` v `(Section 2)`
- $\Delta_v = -\infty$
- **if** $(C_a[v] = 0 \land \tau[v] == size)$ **then**
  - $\Delta_v = V_w[v]$
- **else if** $(C_a[v] = 1)$ **then**
  - $\Delta_v = -V_w[v]$
- **else**
  - $\Delta_v = \mu[v] - weight$
- `// Add a penalty, if necessary; and store the result in array` $gain$
- **if** $((size - \tau[v] \leq 1) \land (iter \geq TL[v])) \lor (weight + \Delta_v > best\_weight)$ **then**
  - $gain[idx] = \Delta_v$
- **else**
  - $gain[idx] = -\infty$

Algorithm 4: Move gain evaluation kernel

Input : vertex to be deleted v, vertex array V_a, edge array E_a, weight array V_w, clique array C_a, array μ (maintains $\mu[v] = w(v) + \sum_{y \in C \cap N(v)} w(y), \forall v$), array τ (maintains $\tau[v] = |N(v) \cap C|, \forall v$), tabu list TL, current iteration $iter$, tabu tenure γ

Output: Updates the current clique C, tabu list TL, vector τ, vector μ

Deletion(v, V_a, E_a, W_a, C_a, τ, μ, TL, $iter$, γ)
    idx = ThreadId() // Get thread id
    // Get idx-th adjacent vertex of v
    $base = V_a[v]$
    $a_{idx} = E_a[base + idx]$
    // Update arrays τ and μ
    $\tau[a_{idx}] = \tau[a_{idx}] - 1$
    $\mu[a_{idx}] = \mu[a_{idx}] - V_w[a_{idx}]$
    // Remove v from the current clique and update the tabu list (the *if* statement determines only one thread should perform the update)
    . **if** ($idx = 0$) **then**
        $C_a[v] = 0$
        $TL[v] = iter + \gamma$

Algorithm 5: Deletion kernel

3.4. Asymptotic running time analysis

The computational cost of GPULS is strongly dominated by the functions *TabuIteration* and *SwapLocalSearch*. In this subsection we analyze the asymptotic running time of these functions.

Function *TabuIteration* consists of three steps. The first step is to calculate the move gain of each vertex (line 2 of Algorithm 2). As shown in Algorithm 4, this can be done in $\mathcal{O}(|V|/p)$ time, for $p \in \mathcal{O}(|V|)$, where p is the number of GPU processors. The second step performs a parallel reduction to find the best move among the $|V|$ candidate moves (line 3 of Algorithm 2), and it can be done in $\mathcal{O}((|V|log|V|)/p)$ time, for $p \in \mathcal{O}(|V|)$ [24]. The final step is to apply the selected move (line 5-15 of Algorithm 2). The *(ω,1)-swap* is the most costly move (lines 12-15 of Algorithm 2): in the worst case, it requires one insertion and $|V| - 1$ deletions. Since kernels *Insertion* and *Deletion* can be both done in $\mathcal{O}(\Delta/p)$ time, where Δ is the highest degree in the graph and $p \in \mathcal{O}(\Delta)$, the third step takes $\mathcal{O}(|V| \times \Delta/p)$ time in the worst case. Therefore, the worst-case running time of the *TabuIteration* function is $\mathcal{O}(|V| \times \Delta/p)$, for $p \in \mathcal{O}(\Delta)$.

In Subsection 3.2, we have shown that the function *SwapLocalSearch* can find an improving *(1,2)-swap* move in $\mathcal{O}(|C||V| + |E|)$ time. Since a *(1,2)-swap* requires two insertions and one deletion (lines 20-22 of Algorithm 3), the worst-case running time of the *SwapLocalSearch* function is $\mathcal{O}(|C||V| + |E| + \Delta/p)$, for $p \in \mathcal{O}(|V|)$.

Table 1: Neighborhood structure comparison.

Heuristic	Neighborhood structures
PLS: Phased Local Search [11]	$S_{0,1} \rightarrow S_{1,1}$
BLS: Breakout Local Search [13]	$S_{0,1} \cup S_{1,1}$
MN/TS: Multi-Neighborhood Tabu Search [12]	$S_{0,1} \cup S_{1,0} \cup S_{1,1}$
LSCC: Local search with SCC (Strong Configuration Checking strategy) [14]	$S_{0,1} \cup S_{1,0} \cup S_{1,1}$
ReTS-I: Restart Tabu Search [15]	$S_{0,1} \cup S_{\omega,1}$
FastWClq [20]	$S_{0,1}$
GPULS: GPU Local Search	$(S_{0,1} \cup S_{1,0} \cup S_{\omega,1}) \rightarrow S_{1,2}$

3.5. Neighborhood structures comparison

Like GPULS, most successful heuristics for the MWCP use local search as their main ingredient [11, 12, 13, 14, 15]. Table 1 compares GPULS neighborhood structures with those of the state-of-the-art local search based algorithms. In this table, $S_{x,y}$ denotes a neighborhood generated by applying the *(x,y)-swap* move. Hence, e.g., PLS approach consists in exploring two neighborhoods: first it explores *(0,1)-swap* moves followed by *(1,1)-swap* moves. BLS, on the other hand, explores a single neighborhood, which is composed of the union of *(0,1)-swap* and *(1,1)-swap* moves. Note $S_{\omega,1} = \cup_{x \geq 1} S_{x,1}$. Although the FastWClq heuristic is not based on local search, the construction phase of this heuristic uses the $S_{0,1}$ neighborhood structure.

Table 1 shows that the first neighborhood explored by GPULS is larger than any other neighborhood. Moreover, its second neighborhood is generated by a move, namely *(1,2)-swap*, that has never been applied before. Although larger neighborhoods require additional computational time, they may improve the quality of the obtained solutions. In GPULS, the additional computational time is relatively small because of its efficient data structures and algorithms. Section 4 presents computational evidence that the proposed enlarged neighborhood structures are indeed beneficial for the search.

Although, in principle, a GPU implementation could be devised for the heuristics above, the quality of a GPU implementation largely depends on the original sequential algorithm and its data structures. Sometimes, there is not enough sufficient parallelism to explore. For instance, the computation cost of the FastWClq heuristic is dominated by the function *ChooseAddVertex(CandSet, k)* (Algorithm 1 in [20]). To the best of our knowledge, a GPU implementation for this function would only be efficient for a large k, as otherwise there would not be enough parallel work to distribute among the GPU threads. However, in FastWClq, k is never greater than 64. Contrary to FastWClq and the other heuristics, GPULS was developed with a GPU implementation in mind, therefore we deliberately did not add features that could improve the algorithm but would jeopardize a GPU implementation (and vice versa).

4. Experimental results

This section reports computational results for GPULS. We have devised four versions for our heuristic:

- **GPULS(CPU)-R**: In this version, GPULS runs exclusively on the CPU and it does not include the graph reduction procedures (line 26 of Algorithm 1).
- **GPULS(GPU)-R**: This is the parallel version of GPULS(CPU)-R. It does not include the graph reductions procedures and runs on a hybrid CPU-GPU environment.
- **GPULS(CPU)**: The third version uses exclusively the CPU, but different from GPULS(CPU)-R, it includes the reduction procedures.
- **GPULS**: This is our main version and it also includes the reduction procedures. In this version, GPULS runs on a hybrid CPU-GPU environment. However, whenever the instance size (as measured by the number of vertices) becomes less than v_{max}, GPULS stops using the GPU and starts to use exclusively the CPU.

All versions were implemented in C++[1]. They were compiled with CUDA 7.5 and g++ 4.7 using the '-O3' optimization flag.

Our experimental platform is composed by a CPU Intel i7 3.6 GHz, with 16 GB of memory (only one CPU core was used), and a GPU GeForce GTX 780 Ti with 3 GB of memory. We ran the DIMACS Machine Benchmark[2], which can be used to compare speeds of different machines when comparing algorithms [12, 13, 19]. This benchmark was compiled with the '-O3' optimization and the CPU times in seconds to execute it were: 0.23 for r300.5, 0.80 for r400.5, and 3.14 for r500.5.

4.1. Instances

GPULS was tested on 130 real-world massive graphs from the Network Data Repository[3], the 16 larger instances from the DIMACS benchmark [4], and on the largest instance from the BHOSLIB benchmark [5] (`frb100-40`). Except for the BHOSLIB instance, these are the same instances used to evaluate LSCC [14], FastWClq [20], and WLMC [21], our main reference algorithms. All instances are originally unweighted. To obtain the corresponding weighted instances, we follow the literature [14, 12] and associate with each vertex i a weight given by $(i \mod 200) + 1$.

Since the weights of the instances described above are artificially generated, we also devised 4 instances[6] based on an interesting application for the MWCP: portfolio selection

[1]The source code of GPULS can be downloaded at https://sites.google.com/site/nogueirabruno/software
[2]dmclique, http://lcs.ios.ac.cn/~caisw/Resource/DIMACS%20machine%20benchmark.tar.gz
[3]http://www.networkrepository.com
[4]http://www.cs.hbg.psu.edu/txn131/clique.html
[5]http://iridia.ulb.ac.be/~fmascia/maximum_clique/BHOSLIB-benchmark
[6]The market graph instances can be downloaded at https://sites.google.com/site/nogueirabruno/software

[6]. In this application, the vertices of the MWCP instance (called market graph) represent the assets to be selected, and the weight associated with each vertex is the corresponding asset return over the time period considered. An edge exists between a pair of vertices if the return correlation coefficient of the assets does not exceed a prespecified threshold, where the threshold value can be adjusted in order to construct different instances. We considered two groups of assets, and generated two instances for each group by using the following threshold values: 0.0 and 0.05. The first group of assets consists of 8103 USA stocks, mutual funds and ETFs. The second group consists of 10616 worldwide stocks. The weight associated with each asset was based on the adjusted closing price of the asset on the year of 2015.

For each instance (151 in total), Table 2 gives the instance name (column '*Instance*'), the number of vertices (column '$|V|$'), the number of edges (column '$|E|$'), and the edge density (column '*density*').

Table 2: MWCP instances: $|V|$ refers to number of vertices, $|E|$ to number of edges, and *density* to the edge density.

Instance	$\|V\|$	$\|E\|$	*density*	Instance	$\|V\|$	$\|E\|$	*density*
Network Data Repository instances							
aff-digg	872622	22501700	0.000059	soc-digg	770799	5907132	0.000020
aff-flickr-user-groups	395979	8537703	0.000109	soc-dogster	426820	8543549	0.000094
aff-orkut-user2groups	8730857	327036486	0.000009	soc-douban	154908	327162	0.000027
bio-dmela	7393	25569	0.000936	soc-epinions	26588	100120	0.000283
bio-human-gene1	22283	12323680	0.049641	soc-flickr	513969	3190452	0.000024
bio-human-gene2	14340	9027024	0.087802	soc-flickr-und	1715255	15555041	0.000011
bio-mouse-gene	45101	14461095	0.014219	soc-flixster	2523386	7918801	0.000002
bio-yeast	1458	1948	0.001834	soc-FourSquare	639014	3214986	0.000016
bn-human-BNU_..._1[7]	1398408	42296922	0.000043	soc-gowalla	196591	950327	0.000049
bn-human-BNU_..._2[8]	1717207	22855526	0.000016	soc-lastfm	1191805	4519330	0.000006
ca-AstroPh	17903	196972	0.001229	soc-livejournal	4033137	27933062	0.000003
ca-citeseer	227320	814134	0.000032	soc-livejournal-user-groups	7489073	112305407	0.000004
ca-coauthors-dblp	540486	15245729	0.000104	soc-LiveMocha	104103	2193083	0.000405
ca-CondMat	21363	91286	0.000400	soc-ljournal-2008	5363186	49514271	0.000003
ca-CSphd	1882	1740	0.000983	soc-orkut	2997166	106349209	0.000024
ca-dblp-2010	226413	716460	0.000028	soc-orkut-dir	3072441	117185083	0.000025
ca-dblp-2012	317080	1049866	0.000021	soc-pokec	1632803	22301964	0.000017
ca-Erdos992	6100	7515	0.000404	soc-sinaweibo	58655849	261321033	0.000000
ca-GrQc	4158	13422	0.001553	soc-slashdot	70068	358647	0.000146
ca-HepPh	11204	117619	0.001874	soc-twitter-follows	404719	713319	0.000009
ca-hollywood-2009	1069126	56306653	0.000099	soc-twitter-higgs	456631	12508442	0.000120
ca-MathSciNet	332689	820644	0.000015	soc-youtube	495957	1936748	0.000016
channel-500x100x100-b050	4802000	3950380	0.000000	soc-youtube-snap	1134890	2987624	0.000005
dbpedia-link	11621692	78621046	0.000001	socfb-A-anon	3097165	23667394	0.000005
delaunay_n22	4194304	5916182	0.000001	socfb-B-anon	2937612	20959854	0.000005
delaunay_n23	8388608	5999193	0.000000	socfb-Berkeley13	22900	852419	0.003251
delaunay_n24	16777216	2183550	0.000000	socfb-CMU	6621	249959	0.011406
friendster	8658744	45671471	0.000001	socfb-Duke14	9885	506437	0.010367
hugebubbles-00020	21198119	2161002	0.000000	socfb-Indiana	29732	1305757	0.002954
hugetrace-00010	12057441	5411527	0.000000	socfb-MIT	6402	251230	0.012261
hugetrace-00020	16002413	5480602	0.000000	socfb-OR	63392	816886	0.000407
ia-email-EU	32430	54397	0.000103	socfb-Penn94	41536	1362220	0.001579
ia-email-univ	1133	5451	0.008500	socfb-Stanford3	11586	568309	0.008468

Continued on next page

[7]The complete name of the instance is bn-human-BNU_1_0025865_session_1-bg

[8]The complete name of the instance is bn-human-BNU_1_0025865_session_2-bg

Table 2 – continued from previous page

Instance	$\|V\|$	$\|E\|$	*density*	Instance	$\|V\|$	$\|E\|$	*density*
ia-enron-large	33696	180811	0.000319	socfb-Texas84	36364	1590651	0.002406
ia-fb-messages	1266	6451	0.008056	socfb-uci-uni	58790782	92208195	0.000000
ia-reality	6809	7680	0.000331	socfb-UCLA	20453	747604	0.003574
ia-wiki-Talk	92117	360767	0.000085	socfb-UConn	17206	604867	0.004087
inf-europe_osm	50912018	54054660	0.000000	socfb-UCSB37	14917	482215	0.004334
inf-germany_osm	11548845	12369181	0.000000	socfb-UF	35111	1465654	0.002378
inf-power	4941	6594	0.000540	socfb-UIllinois	30795	1264421	0.002667
inf-roadNet-CA	1957027	2760388	0.000001	socfb-Wisconsin87	23831	835946	0.002944
inf-roadNet-PA	1087562	1541514	0.000003	tech-as-caida2007	26475	53381	0.000152
inf-road_usa	23947347	28854312	0.000000	tech-as-skitter	1694616	11094209	0.000008
rec-amazon	91813	125704	0.000030	tech-internet-as	40164	85123	0.000106
rec-dating	168792	17351416	0.001218	tech-ip	2250498	21643497	0.000009
rec-epinions	755761	13396042	0.000047	tech-p2p-gnutella	62561	147878	0.000076
rec-libimseti-dir	220970	17233144	0.000706	tech-RL-caida	190914	607610	0.000033
rec-movielens	71567	9991339	0.003902	tech-routers-rf	2113	6632	0.002972
rgg_n_2_24_s0	16777216	583009	0.000000	tech-WHOIS	7476	56943	0.002038
rt-retweet-crawl	1112702	2278852	0.000004	twitter_mpi	9862152	99940317	0.000002
san1000	1000	250500	0.501502	web-arabic-2005	163598	1747269	0.000131
scc_twitter-copen	8580	473614	0.012869	web-baidu-baike	2141300	17014946	0.000007
sc-ldoor	952203	20770807	0.000046	web-BerkStan	12305	19500	0.000258
sc-msdoor	415863	9378650	0.000108	web-edu	3031	6474	0.001410
sc-nasasrb	54870	1311227	0.000871	web-google	1299	2773	0.003289
sc-pkustk11	87804	2565054	0.000665	web-indochina-2004	11358	47606	0.000738
sc-pkustk13	94893	3260967	0.000724	web-it-2004	509338	7178413	0.000055
sc-pwtk	217891	5653221	0.000238	web-sk-2005	121422	334419	0.000045
sc-rel9	5921786	23667162	0.000001	web-spam	4767	37375	0.003290
sc-shipsec1	140385	1707759	0.000173	web-uk-2005	129632	11744049	0.001398
sc-shipsec5	179104	2200076	0.000137	web-webbase-2001	16062	25593	0.000198
soc-BlogCatalog	88784	2093195	0.000531	web-wikipedia2009	1864433	4507315	0.000003
soc-brightkite	56739	212945	0.000132	web-wikipedia-growth	1870709	36532531	0.000021
soc-buzznet	101163	2763066	0.000540	web-wikipedia_link_it	2936413	86754664	0.000020
soc-delicious	536108	1365961	0.000010	wikipedia_link_en	27154756	31024475	0.000000
DIMACS & BHOSLIB instances							
C1000-9	1000	450079	0.901059	MANN-a45	1035	533115	0.996300
C2000-5	2000	999836	0.500168	MANN-a81	3321	5506380	0.998825
C2000-9	2000	1799532	0.900216	p-hat1000-1	1000	122253	0.244751
C4000-5	4000	4000268	0.500159	p-hat1000-2	1000	244799	0.490088
DSJC1000-5	1000	249826	0.500152	p-hat1000-3	1000	371746	0.744236
frb100-40	4000	7425226	0.928385	p-hat1500-1	1500	284923	0.253434
hamming10-2	1024	518656	0.990225	p-hat1500-2	1500	568960	0.506080
hamming10-4	1024	434176	0.828935	p-hat1500-3	1500	847244	0.753608
keller6	3361	4619898	0.818191				
Market Graph instances							
mg_usa_assets_00	8103	11781256	0.358908	mg_worldwide_stocks_00	10616	18644702	0.330906
mg_usa_assets_05	8103	19312628	0.588347	mg_worldwide_stocks_05	10616	31256688	0.554743

4.2. Comparison with local search based heuristics

We first compare the performance of GPULS with the following state-of-the-art MWCP local search based algorithms: LSCC, MN/TS, and ReTS-I (recall the neighborhood structures of these algorithms in Table 1). We considered the BMS (Best from Multiple Selection) [14] versions of LSCC and MN/TS, which are specifically targeted to massive graphs. The source code of LSCC and MN/TS were made available by their authors, but only a binary of ReTS-I could be obtained. In preliminary experiments, the binary of ReTS-I algorithm was not able to solve most of our instances, which we think was mainly due to its memory-expensive data structures. Thus we implemented a version of this algorithm using our own data structures and used it for the experiments of this section. LSCC, MN/TS, and our version of ReTS-I were compiled with g++ 4.7 and '-O3' flag. Besides, we considered their default parameter settings. Since these algorithms do not consider graph reduction procedures, the comparison is based on the GPULS(CPU)-R version of our heuristic.

We performed 40 independent runs of each heuristic for each benchmark instance. As stop criteria for the runs, we defined a time limit (cutoff time) of 100 seconds. Table 3 presents the comparison results. The methods are compared using the following criteria: the best (average in parenthesis) solution obtained (column '*Best*'), and the average CPU time to find the best solution (column '*t (s)*'). These are the most common criteria used in the literature to compare MWCP heuristics [12, 13, 19]. If in a run the heuristic fails to attain the best solution, its CPU time to find the best is considered to be the cutoff time. The bottom of Table 3 shows a summary that includes: average of the average CPU time values to find the best solution, number of instances in which the method found the best weight (in parenthesis the number of instances in which the method determined an average clique weight that is not inferior to those determined by the other methods), and the average of the average relative gap values. The relative gap is calculated as $gap = 100(Weight(C^*) - Weight(C'))/Weight(C^*)$, where C^* is the best solution for the instance and C' is the best solution found by the method.

Table 3: Comparative results of GPULS(CPU)-R, LSCC, MN/TS and ReTS-I.

Instance	LSCC		MN/TS		ReTS-I		GPULS(CPU)-R	
	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)
aff-digg	3836 (3766.18)	85.27	3836 (3800)	76.34	3836 (3836)	6.43	3836 (3833)	21.96
aff-flickr-user-groups	1720 (1720)	11.45	1720 (1720)	0.70	1720 (1720)	0.57	1720 (1720)	0.22
aff-orkut-user2groups	971 (966.925)	63.72	971 (833.15)	79.33	971 (964.475)	38.13	971 (948.375)	60.59
bio-dmela	805 (805)	0.00	805 (805)	0.00	805 (805)	0.00	805 (805)	0.00
bio-human-gene1	134602 (134326)	100.00	133942 (133211)	100.00	134605 (134364)	100.00	**134611** (134269)	98.91
bio-human-gene2	135262 (135068)	100.00	134781 (134192)	100.00	**135286** (135154)	98.34	135253 (135136)	100.00
bio-mouse-gene	59952 (59884.7)	98.22	59619 (58712.3)	100.00	59952 (59854.9)	87.29	59928 (59791.2)	100.00
bio-yeast	629 (629)	0.00	629 (629)	2.25	629 (629)	0.06	629 (629)	0.13
bn-human-BNU_..._1	22038 (13648.8)	100.00	23307 (13405.3)	100.00	21733 (10331.2)	100.00	**25376** (17416.8)	98.90
bn-human-BNU_..._2	**14141** (6399.68)	99.00	11044 (6131.02)	100.00	10913 (3059.78)	100.00	12305 (6811.12)	100.00
C1000-9	9254 (9214.8)	78.41	9254 (9254)	32.80	9254 (9254)	2.28	9254 (9254)	4.28
C2000-5	2466 (2466)	1.49	2466 (2466)	0.96	2466 (2466)	0.40	2466 (2466)	0.66
C2000-9	10999 (10839.5)	99.33	10964 (10910.5)	100.00	10999 (10981.5)	74.91	10999 (10915.3)	99.39
C4000-5	2792 (2790.43)	44.95	2792 (2788.45)	53.50	2792 (2792)	25.12	2792 (2791.62)	40.70
ca-AstroPh	5338 (5338)	11.01	5338 (5336.65)	25.70	5338 (5338)	18.49	5338 (5338)	5.53
ca-citeseer	8838 (8095.23)	69.16	8838 (7451.2)	78.08	8838 (8521.23)	45.62	8838 (8724.42)	34.39
ca-coauthors-dblp	37884 (23954.2)	98.08	34750 (16251.3)	100.00	37884 (23688)	99.93	37884 (27079.9)	95.03
ca-CondMat	2887 (2887)	1.14	2887 (2831.93)	37.46	2887 (2887)	1.49	2887 (2887)	6.63
ca-CSphd	489 (489)	0.00	489 (489)	9.02	489 (489)	0.01	489 (489)	0.53
ca-dblp-2010	7575 (7035.88)	98.38	7456 (6340.15)	100.00	7456 (6624.38)	100.00	7575 (7204.95)	96.06
ca-dblp-2012	14108 (8985.38)	78.61	14108 (5601.62)	91.69	14108 (10342.4)	70.41	14108 (10626.1)	68.88
ca-Erdos992	958 (958)	0.00	958 (958)	0.25	958 (958)	0.01	958 (958)	0.03
ca-GrQc	4279 (4279)	0.04	4279 (4279)	0.25	4279 (4279)	0.01	4279 (4279)	0.01
ca-HepPh	24533 (24533)	0.03	24533 (24533)	0.23	24533 (24533)	0.02	24533 (24533)	0.03
ca-hollywood-2009	222720 (103937)	92.41	222720 (18079.7)	100.00	222720 (107596)	85.98	222720 (113564)	84.79
ca-MathSciNet	2611 (2141.97)	100.00	**2792** (1967.38)	98.60	2611 (2332.53)	100.00	2611 (2238.03)	100.00
channel-500x100x100-b050	796 (796)	1.24	796 (796)	1.37	796 (259.6)	92.25	796 (796)	4.16
dbpedia-link	3428 (1695.67)	100.00	3428 (1727.47)	100.00	3513 (2079.3)	99.01	3513 (2393.43)	99.30

Continued on next page

Table 3 – continued from previous page

Instance	LSCC		MN/TS		ReTS-I		GPULS(CPU)-R	
	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)
delaunay_n22	**793** (772.125)	93.81	790 (638)	100.00	642 (433.925)	100.00	761 (629.1)	100.00
delaunay_n23	**794** (768.275)	96.38	786 (632.375)	100.00	784 (315.7)	100.00	759 (612.425)	100.00
delaunay_n24	788 (647.15)	100.00	**790** (642.9)	98.69	596 (229.225)	100.00	710 (465.675)	100.00
DSJC1000-5	2186 (2186)	6.72	2186 (2186)	0.13	2186 (2186)	0.05	2186 (2186)	0.03
frb100-40	10424 (10321)	100.00	10427 (10253)	100.00	10398 (10255)	100.00	**10443** (10339.3)	98.97
friendster	1874 (1323.85)	100.00	1864 (1077.05)	100.00	**2371** (1578.28)	97.57	2316 (1722.45)	100.00
hamming10-2	50512 (50512)	0.27	50512 (50512)	3.45	50512 (50512)	0.03	50512 (50512)	0.08
hamming10-4	5129 (5129)	12.72	5129 (5129)	9.10	5129 (5129)	3.17	5129 (5129)	2.63
hugebubbles-00020	400 (399.075)	97.29	400 (399.05)	97.68	395 (212.15)	100.00	395 (269.025)	100.00
hugetrace-00010	400 (399.05)	97.78	400 (396.475)	98.91	399 (227.675)	100.00	399 (367.25)	100.00
hugetrace-00020	400 (399.125)	93.61	400 (398.675)	98.80	399 (229.75)	100.00	399 (365.925)	100.00
ia-email-EU	1350 (1350)	0.12	1350 (1350)	0.15	1350 (1350)	0.02	1350 (1350)	0.03
ia-email-univ	1473 (1473)	0.00	1473 (1473)	0.06	1473 (1473)	0.00	1473 (1473)	0.00
ia-enron-large	2490 (2490)	3.48	2490 (2490)	0.86	2490 (2490)	0.08	2490 (2490)	0.06
ia-fb-messages	791 (791)	0.00	791 (791)	0.00	791 (791)	0.00	791 (791)	0.00
ia-reality	374 (374)	0.04	374 (368.875)	44.95	374 (374)	0.02	374 (374)	2.17
ia-wiki-Talk	1884 (1884)	0.29	1884 (1884)	0.10	1884 (1884)	0.04	1884 (1884)	0.08
inf-europe_osm	**592** (426.325)	99.41	591 (418.375)	100.00	399 (397.2)	97.51	430 (399.775)	95.03
inf-germany_osm	597 (449.925)	97.78	597 (410.525)	98.83	577 (403.1)	100.00	577 (405.25)	100.00
inf-power	888 (888)	0.02	888 (888)	13.16	888 (888)	0.06	888 (888)	0.45
inf-roadNet-CA	**668** (600.525)	98.92	597 (580.675)	100.00	588 (579.5)	100.00	597 (582.725)	100.00
inf-roadNet-PA	599 (597.375)	89.94	599 (587.85)	98.29	590 (587.1)	100.00	599 (589.925)	97.55
inf-road_usa	**612** (573.025)	99.31	597 (507.9)	100.00	521 (467.525)	99.89	532 (422.775)	100.00
keller6	7861 (7629.88)	100.00	7861 (7628.18)	100.00	7895 (7625.45)	100.00	**7968** (7797.77)	98.17
MANN-a45	**34249** (34219.2)	99.39	34171 (34154.8)	100.00	34184 (34177.2)	100.00	34197 (34188.4)	100.00
MANN-a81	111077 (111030)	100.00	111069 (111011)	100.00	111115 (111096)	100.00	**111150** (111127)	97.95
p-hat1000-1	1514 (1514)	0.91	1514 (1514)	0.02	1514 (1514)	0.02	1514 (1514)	0.08
p-hat1000-2	5777 (5777)	0.15	5777 (5777)	0.08	5777 (5777)	0.00	5777 (5777)	0.00
p-hat1000-3	8111 (8111)	3.25	8111 (8111)	2.43	8111 (8111)	0.01	8111 (8111)	0.08
p-hat1500-1	1619 (1619)	0.01	1619 (1619)	0.05	1619 (1619)	0.03	1619 (1619)	0.08
p-hat1500-2	7360 (7360)	1.33	7360 (7360)	0.86	7360 (7360)	0.14	7360 (7360)	0.04
p-hat1500-3	10321 (10303)	74.76	10321 (10290)	90.21	10321 (10321)	0.33	10321 (10321)	6.78
rec-amazon	942 (942)	2.18	942 (942)	2.18	942 (942)	1.75	942 (899.925)	84.35
rec-dating	1699 (1699)	1.59	1699 (1699)	0.75	1699 (1699)	0.46	1699 (1699)	0.21
rec-epinions	1054 (1054)	5.22	1054 (1035.9)	64.65	1054 (1054)	6.68	1054 (1054)	7.55
rec-libimseti-dir	1938 (1909.53)	66.28	1938 (1938)	6.46	1938 (1938)	0.28	1938 (1938)	0.62
rec-movielens	3777 (3777)	3.39	3777 (3777)	1.03	3777 (3777)	0.25	3777 (3777)	0.31
rgg_n_2_24_s0	1629 (1165.47)	100.00	**1699** (1179.97)	97.84	200 (200)	100.00	1198 (1047.7)	100.00
rt-retweet-crawl	1367 (1332.6)	65.95	1367 (994.775)	94.18	1367 (1367)	7.72	1367 (1215.12)	69.21
san1000	1716 (1715.25)	37.22	1716 (1709.5)	75.55	1716 (1716)	2.58	1716 (1716)	21.02
scc_twitter-copen	58699 (58699)	4.28	58699 (58699)	8.57	58699 (58699)	3.76	58699 (58699)	4.05
sc-ldoor	4074 (3659)	98.65	3976 (3378.75)	100.00	4067 (3779.3)	100.00	4074 (3868.78)	98.37
sc-msdoor	4074 (3885.43)	100.00	4046 (3765.9)	100.00	**4088** (3933.72)	98.23	4067 (3925.8)	100.00
sc-nasasrb	4548 (4531.8)	73.10	4548 (4254.5)	98.42	4548 (4545)	41.47	4548 (4469.8)	85.35
sc-pkustk11	5298 (4764.2)	94.80	5298 (4440.25)	99.29	5298 (5037.95)	83.43	5298 (4828.57)	85.94
sc-pkustk13	5928 (5695.73)	100.00	5853 (5307)	100.00	6306 (5967.68)	88.13	6306 (5775.52)	96.28
sc-pwtk	4620 (4544.7)	92.64	4476 (4215.5)	100.00	4620 (4506.3)	90.54	4620 (4361.7)	98.15
sc-rel9	572 (410)	96.27	572 (403.65)	98.49	572 (429.1)	90.82	572 (410.3)	93.08
sc-shipsec1	3540 (3075.25)	94.99	3255 (2857.5)	100.00	3540 (3185.55)	84.94	3540 (3174.05)	90.28
sc-shipsec5	4500 (3962.55)	100.00	4440 (3719.22)	100.00	4524 (4365.15)	86.84	4524 (4349.7)	92.85
soc-BlogCatalog	4803 (4803)	10.18	4803 (4803)	21.99	4803 (4803)	0.35	4803 (4803)	0.66
soc-brightkite	3672 (3645.15)	99.21	3672 (3598.2)	96.09	3672 (3412.5)	62.35	3672 (3655.53)	76.40
soc-buzznet	2981 (2979.5)	50.33	2981 (2980.25)	44.63	2981 (2981)	8.07	2981 (2981)	0.74
soc-delicious	1547 (1494.62)	89.95	1547 (1485.38)	88.17	1540 (1488.2)	100.00	1547 (1540.6)	83.28
soc-digg	4675 (4128.32)	100.00	5286 (4449.75)	100.00	5303 (4537.57)	99.92	5303 (4791.48)	96.42
soc-dogster	4418 (4006.78)	95.89	4356 (3900.75)	100.00	4257 (3929.4)	100.00	4418 (4289.73)	98.50
soc-douban	1682 (1682)	1.27	1682 (1631.05)	47.31	1682 (1547.9)	57.54	1682 (1682)	21.30
soc-epinions	1657 (1657)	19.41	1657 (1654.4)	20.88	1657 (1646.6)	45.62	1657 (1657)	1.85
soc-flickr	7083 (4940.93)	85.79	7083 (6785.3)	98.30	7083 (7061.4)	47.80	7083 (7073.1)	59.71

Continued on next page

Table 3 – continued from previous page

Instance	LSCC		MN/TS		ReTS-I		GPULS(CPU)-R	
	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)
soc-flickr-und	10127 (5176.85)	97.53	9948 (5848.4)	100.00	10127 (5862.68)	94.00	10126 (8567.98)	100.00
soc-flixster	3805 (2094.97)	98.35	3805 (2221.32)	96.04	3805 (2701.88)	93.42	3805 (3684.9)	59.98
soc-FourSquare	3064 (2974.65)	96.37	3064 (2990.18)	95.22	3064 (3055)	51.55	3064 (3055.15)	46.37
soc-gowalla	2335 (2194.4)	95.66	2335 (2215.32)	88.14	2335 (2194.35)	96.13	2335 (2246.9)	77.72
soc-lastfm	1773 (1670.5)	86.68	1773 (1677.75)	87.14	1773 (1773)	18.02	1773 (1772.55)	25.35
soc-livejournal	5975 (1649.45)	100.00	2796 (1586.88)	100.00	3299 (2050.9)	100.00	**19368** (3153.62)	99.73
soc-livejournal-user-groups	1054 (1024.65)	78.75	1054 (928.65)	97.56	1054 (1017.9)	67.54	1054 (1028.08)	71.34
soc-LiveMocha	1784 (1784)	4.12	1784 (1784)	0.66	1784 (1784)	0.06	1784 (1784)	0.06
soc-ljournal-2008	9862 (3068.07)	100.00	9626 (2671.78)	100.00	16000 (3586.22)	100.00	**21013** (5442.98)	98.61
soc-orkut	4243 (2640.68)	100.00	3572 (2624.97)	100.00	**4969** (2947.72)	98.18	4905 (3553.32)	100.00
soc-orkut-dir	4415 (2683.78)	100.00	5517 (2971)	100.00	**5897** (3152.62)	98.22	5453 (3855.5)	100.00
soc-pokec	2341 (1455.12)	100.00	3191 (1539.3)	98.68	3191 (1960.47)	96.67	3191 (2277.7)	94.75
soc-sinaweibo	4759 (1899.75)	97.81	4759 (1984.22)	98.04	4667 (2593.22)	100.00	4759 (2543.75)	98.24
soc-slashdot	2811 (2811)	8.43	2811 (2811)	0.15	2811 (2811)	0.11	2811 (2811)	0.07
soc-twitter-follows	808 (808)	7.15	808 (638.725)	95.85	808 (808)	13.16	808 (719.275)	85.13
soc-twitter-higgs	4727 (4187.82)	100.00	4727 (4452.6)	100.00	4727 (4595.27)	100.00	**8039** (4974.32)	96.98
soc-youtube	1961 (1952.65)	19.96	1961 (1952.97)	20.18	1961 (1961)	18.82	1961 (1961)	3.49
soc-youtube-snap	1787 (1579.7)	84.80	1787 (1633.28)	74.28	1787 (1687.45)	64.93	1787 (1781.65)	16.78
socfb-A-anon	2269 (1560.55)	100.00	2260 (1451.22)	100.00	2358 (1889.12)	100.00	**2872** (2251.2)	98.01
socfb-B-anon	2470 (1602.92)	100.00	2513 (1445.9)	100.00	2513 (1789.53)	100.00	**2537** (2049.1)	95.33
socfb-Berkeley13	4906 (4906)	24.69	4906 (4855.85)	40.86	4906 (4906)	15.38	4906 (4906)	6.07
socfb-CMU	4141 (4141)	2.17	4141 (4141)	2.25	4141 (4141)	1.36	4141 (4141)	0.15
socfb-Duke14	3694 (3694)	2.99	3694 (3694)	9.61	3694 (3694)	1.51	3694 (3694)	0.31
socfb-Indiana	5412 (5343)	44.05	5412 (5377.7)	44.97	5412 (5387.8)	34.03	5412 (5412)	8.03
socfb-MIT	3658 (3658)	1.79	3658 (3658)	2.85	3658 (3658)	2.76	3658 (3658)	0.27
socfb-OR	3523 (3417.07)	74.11	3523 (3467.45)	65.98	3523 (3504.45)	53.92	3523 (3523)	13.94
socfb-Penn94	4738 (4569)	46.17	4738 (4474.57)	78.12	4738 (4386.65)	74.62	4738 (4738)	10.77
socfb-Stanford3	5769 (5769)	9.08	5769 (5769)	12.46	5769 (5769)	8.98	5769 (5769)	2.59
socfb-Texas84	5546 (5533.68)	36.62	5546 (5423.23)	93.15	5546 (5546)	19.84	5546 (5543.2)	56.13
socfb-uci-uni	838 (603)	100.00	594 (487.3)	100.00	774 (598.575)	100.00	**1045** (542.475)	98.85
socfb-UCLA	5595 (5595)	20.01	5595 (5587.25)	30.67	5595 (5579.5)	33.84	5595 (5595)	4.87
socfb-UConn	5733 (5733)	11.38	5733 (5733)	5.52	5733 (5733)	2.23	5733 (5733)	0.75
socfb-UCSB37	5669 (5669)	11.42	5669 (5669)	16.30	5669 (5669)	7.52	5669 (5669)	2.87
socfb-UF	6043 (6035.1)	34.95	6043 (5968.25)	89.15	6043 (6035.1)	43.17	6043 (6042.4)	28.27
socfb-UIllinois	5730 (5695.88)	54.73	5730 (5662.88)	94.86	5730 (5675.18)	66.90	5730 (5728.6)	63.28
socfb-Wisconsin87	4239 (4239)	27.15	4239 (4239)	19.83	4239 (4239)	10.95	4239 (4239)	2.35
tech-as-caida2007	1869 (1869)	0.06	1869 (1869)	0.04	1869 (1869)	0.00	1869 (1869)	0.00
tech-as-skitter	5703 (2941.55)	97.52	5524 (2653.22)	100.00	5703 (4029.4)	99.51	5703 (5031.7)	98.60
tech-internet-as	1692 (1692)	0.54	1692 (1692)	0.07	1692 (1692)	0.02	1692 (1692)	0.05
tech-ip	668 (593.075)	98.51	597 (526.4)	100.00	668 (607.525)	95.08	668 (668)	1.71
tech-p2p-gnutella	703 (703)	0.59	703 (619.575)	82.21	703 (590.45)	95.44	703 (686.7)	44.16
tech-RL-caida	1861 (1861)	10.01	1861 (1860.42)	13.68	1861 (1859.85)	21.46	1861 (1861)	2.39
tech-routers-rf	1460 (1460)	0.08	1460 (1460)	0.18	1460 (1460)	0.18	1460 (1460)	0.03
tech-WHOIS	6154 (6154)	0.27	6154 (6154)	9.90	6154 (6154)	0.11	6154 (6154)	1.14
twitter_mpi	10893 (6788.62)	97.74	10794 (8649)	100.00	10893 (8481.35)	98.31	10858 (10836.9)	100.00
web-arabic-2005	10558 (10558)	16.87	10558 (10075.1)	88.53	10558 (10220.2)	87.72	10558 (10427.6)	88.69
web-baidu-baike	1823 (1558.5)	97.50	1823 (1630.55)	97.50	1707 (1688.78)	100.00	1823 (1715.1)	92.51
web-BerkStan	3249 (3249)	0.07	3249 (3249)	0.77	3249 (3249)	0.06	3249 (3249)	1.56
web-edu	2077 (2077)	1.81	2077 (2077)	1.94	2077 (1988.8)	45.83	2077 (2077)	0.38
web-google	1749 (1749)	0.07	1749 (1749)	0.06	1749 (1749)	0.01	1749 (1749)	0.10
web-indochina-2004	6997 (6997)	0.03	6997 (6997)	7.09	6997 (6997)	4.93	6997 (6997)	0.56
web-it-2004	45477 (43169.8)	86.73	42412 (21075.6)	100.00	41378 (11257.2)	100.00	45477 (39712.4)	97.64
web-sk-2005	11925 (10256.5)	71.04	11925 (7271.15)	97.84	9245 (4776.3)	100.00	11925 (10919.9)	79.93
web-spam	2503 (2503)	6.75	2503 (2503)	6.69	2503 (2503)	6.23	2503 (2503)	0.77
web-uk-2005	54850 (54782.1)	22.01	54850 (50660.4)	82.18	46050 (34057.6)	100.00	54850 (54494.2)	33.18
web-webbase-2001	3574 (3251.43)	57.19	3574 (3222.1)	61.82	2401 (2401)	100.00	3574 (3574)	9.14
web-wikipedia2009	3066 (1275.12)	100.00	3066 (1092.12)	100.00	2806 (1345.3)	100.00	**3455** (1725.83)	99.13
web-wikipedia-growth	2618 (1943.97)	100.00	3061 (2089.93)	100.00	2652 (2181)	100.00	**4741** (2559.03)	97.59
web-wikipedia_link_it	77799 (12870.5)	97.63	77799 (9911)	97.63	63022 (13133.9)	100.00	77799 (32026.8)	97.57

Continued on next page

Table 3 – continued from previous page

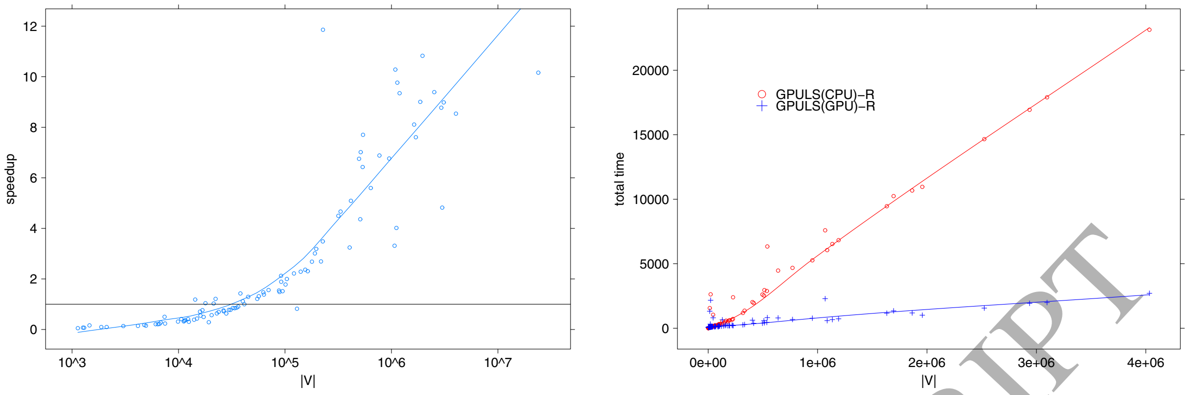
Instance	LSCC		MN/TS		ReTS-I		GPULS(CPU)-R	
	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)
wikipedia_link_en	1002 (772.4)	100.00	1002 (716.875)	100.00	1115 (929.85)	100.00	**1584** (770.525)	99.62
Avg. t (s)	54.61		60.66		52.37		**48.99**	
# Best (# Best Avg.)	120 (76)		104 (49)		107 (74)		**125 (106)**	
Avg. Gap	2.95		3.36		4.37		**0.92**	

Table 3 indicates GPULS(CPU)-R found the best solution values in 125 out of the 147 instances, whereas LSCC, MN/TS and ReTS-I found the best solutions in 120, 104 and 107 instances, respectively. It is worth noting that for the instance `soc-livejournal`, GPULS(CPU)-R was able to find a solution that is greater by a factor of 3.24, 6.92 and 5.87 compared with the best ones found LSCC, MN/TS and ReTS-I, respectively. Moreover, in 106 instances GPULS(CPU)-R determined average clique weights that are not inferior to those found by the other algorithms (the same measure is 76, 49, and 74 for LSCC, MN/TS and ReTS-I, respectively). Our method was better not only with respect to the solution quality, but it also shows a superior performance in terms of execution time: GPULS(CPU)-R was 1.11, 1.23, 1.06 times faster than LSCC, MN/TS and ReTS-I, respectively. These results demonstrate that our neighborhood structure is better than the current ones, and it also has the additional benefit that it can be explored using a massivelly parallel architecture.

4.3. GPU acceleration results

In this subsection, we analyse the speedup of our heuristic running on a hybrid CPU-GPU environment. For this experiment, we performed 40 independent runs of GPULS(GPU)-R and GPULS(CPU)-R on each benchmark instance, and set as stop criteria for both algorithms a maximum number of 2×10^6 local search iterations. The speedup was measured as the total time of GPULS(CPU)-R divided by the total time of GPULS(GPU)-R. Due to the 3 GB memory limitation of our GPU board, we were not able to run GPULS(GPU)-R on the instances `aff-orkut-user2groups`, `socfb-uci-uni`, and `soc-sinaweibo` (the larger ones).

For each instance, Figure 4a plots the number of vertices on a log scale vs. the speedup obtained. This figure indicates that the speedup varied from 0.05 to 11.85, and that the speedup increases with the problem size. In particular, for the instance `ca-coauthors`, GPULS(GPU)-R reduced the average execution time from 40 minutes to 4 minutes. Except for the instance `web-uk-2005`, the speedup was below one for instances with less than 42000 vertices. This is mainly due to the overhead incurred in calling the GPU kernels, as well as transferring data between the GPU and CPU memories. Figure 4b shows the execution time of GPULS(CPU)-R and GPULS(GPU)-R as a function of the number of vertices. It can be seen in this figure that the slope of GPULS(GPU)-R line is much smaller than the slope of GPULS(CPU)-R line. These results demonstrate the scalability of our heuristic, and its potential for efficiently solving massive MWCP instances.



(a) Number of vertices on a log scale vs. speedup.

(b) Number of vertices vs. total time.

Figure 4: GPU acceleration.

4.4. Comparison with the FastWClq heuristic

We now compare our approach with the reduction based heuristic FastWClq [20]. The source code of FastWClq was provided by their authors and compiled with g++ 4.7 using the '-O3' flag. For this comparison, we consider the following versions of our heuristic: GPULS(CPU)-R, GPULS(CPU), and GPULS. For this and the next experiments, we set the parameter v_{max} of GPULS to 100000. This value was based on the results of the previous subsection, which indicates that a speedup greater than two is very likely to be obtained if the number of vertices is greater than 100000. The same protocol used in Subsection 4.2 (i.e., number of runs, cutoff time) was adopted to perform the comparison. Table 4 presents the results, where an asterisk means that the method has been able to prove the optimality of a given result. Henceforth, on the results of the instances in which GPULS was not able to run due to the GPU memory limitation (`aff-orkut-user2groups`, `socfb-uci-uni`, and `soc-sinaweibo`), we use GPULS(CPU) results.

Table 4: Comparative results of FastWClq, GPULS(CPU)-R, GPULS(CPU), GPULS.

	FastWClq		GPULS(CPU)-R		GPULS(CPU)		GPULS	
Instance	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)
aff-digg	1803 (1064.38)	100.00	3836 (3833)	21.96	3836 (3836)	33.16	3836 (3836)	37.96
aff-flickr-user-groups	1535 (1293.25)	100.00	1720 (1720)	0.22	1720 (1720)	0.53	1720 (1720)	0.29
aff-orkut-user2groups	693 (592.475)	100.00	971 (948.375)	60.59	971 (963.7)	58.38	971 (963.7)	58.38
bio-dmela	805 (805)	0.01	805 (805)	0.00	805 (805)	0.00	805 (805)	0.00
bio-human-gene1	133718 (133232)	100.00	134611 (134269)	100.00	134674 (134533)	97.75	134674 (134533)	97.75
bio-human-gene2	134845 (134408)	100.00	135253 (135136)	100.00	135310 (135247)	99.20	135310 (135247)	99.20
bio-mouse-gene	59602 (59280.4)	100.00	59928 (59791.2)	100.00	59943 (59863.3)	98.01	59943 (59863.3)	98.01
bio-yeast	629* (629)	0.00	629 (629)	0.13	629* (629)	0.00	629* (629)	0.00
bn-human-BNU_..._1	26521 (25325.6)	100.00	25376 (17416.8)	100.00	**27545** (22517.7)	99.01	20213 (18927.3)	100.00
bn-human-BNU_..._2	19189 (19189)	48.68	12305 (6811.12)	100.00	19189 (13540.6)	99.25	19189 (16348.2)	80.63
C1000-9	7947 (7505.57)	100.00	9254 (9254)	4.28	9254 (9254)	4.84	9254 (9254)	4.84
C2000-5	2420 (2086.93)	100.00	2466 (2466)	0.66	2466 (2466)	0.75	2466 (2466)	0.75
C2000-9	8487 (8139.75)	100.00	10999 (10915.3)	99.39	10999 (10908.9)	99.53	10999 (10908.9)	99.53
C4000-5	2290 (2112.1)	100.00	2792 (2791.62)	40.70	2792 (2791.62)	42.36	2792 (2791.62)	42.36
ca-AstroPh	5338* (5338)	0.03	5338 (5338)	5.53	5338* (5338)	0.11	5338* (5338)	0.11

Continued on next page

Table 4 – continued from previous page

Instance	FastWClq		GPULS(CPU)-R		GPULS(CPU)		GPULS	
	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)
ca-citeseer	8838* (8838)	0.10	8838 (8724.42)	34.39	8838* (8838)	0.38	8838* (8838)	0.22
ca-coauthors-dblp	37884* (37884)	1.39	37884 (27079.9)	95.03	37884* (37884)	2.64	37884* (37884)	1.66
ca-CondMat	2887* (2887)	0.01	2887 (2887)	6.63	2887* (2887)	0.02	2887* (2887)	0.02
ca-CSphd	489* (489)	0.00	489 (489)	0.53	489* (489)	0.00	489* (489)	0.00
ca-dblp-2010	7575* (7575)	0.08	7575 (7204.95)	96.06	7575* (7575)	0.30	7575* (7575)	0.14
ca-dblp-2012	14108* (14108)	0.22	14108 (10626.1)	68.88	14108* (14108)	0.74	14108* (14108)	0.45
ca-Erdos992	958* (958)	0.00	958 (958)	0.03	958* (958)	0.00	958* (958)	0.00
ca-GrQc	4279* (4279)	0.00	4279 (4279)	0.01	4279* (4279)	0.00	4279* (4279)	0.00
ca-HepPh	24533* (24533)	0.01	24533 (24533)	0.03	24533* (24533)	0.02	24533* (24533)	0.02
ca-hollywood-2009	222720* (222720)	25.58	222720 (113564)	84.79	222720* (222720)	61.38	222720* (222720)	67.92
ca-MathSciNet	2792* (2792)	0.23	2611 (2238.03)	100.00	2792* (2792)	0.81	2792* (2792)	0.39
channel-500x100x100-b050	796 (796)	2.11	796 (796)	4.16	796 (796)	6.54	796 (796)	2.05
dbpedia-link	1925 (1177.58)	100.00	3513 (2393.43)	99.30	3513 (2513.85)	97.98	3513 (2634.48)	99.98
delaunay_n22	794* (794)	3.51	761 (629.1)	100.00	794* (791.175)	37.55	794* (794)	11.39
delaunay_n23	794* (794)	5.83	759 (612.425)	100.00	794* (781.025)	55.51	794* (794)	21.27
delaunay_n24	597* (597)	100.00	710 (465.675)	100.00	793* (793)	23.97	793* (793)	97.66
DSJC1000-5	2186 (2113.32)	99.23	2186 (2186)	0.03	2186 (2186)	0.05	2186 (2186)	0.05
frb100-40	8427 (8088.62)	100.00	10443 (10339.3)	98.97	10443 (10332.9)	99.12	10443 (10332.9)	99.12
friendster	**5511*** (5511)	55.55	2316 (1722.45)	100.00	2885 (1917.47)	100.00	2885 (2281.47)	100.00
hamming10-2	50512 (49695.4)	94.85	50512 (50512)	0.08	50512 (50512)	0.13	50512 (50512)	0.13
hamming10-4	4498 (4308.68)	100.00	5129 (5129)	2.63	5129 (5129)	2.64	5129 (5129)	2.64
hugebubbles-00020	400* (400)	9.51	395 (269.025)	100.00	400 (400)	24.19	400 (400)	6.23
hugetrace-00010	**400** (400)	6.70	399 (367.25)	100.00	399 (399)	100.00	399 (399)	100.00
hugetrace-00020	400 (400)	8.19	399 (365.925)	100.00	399 (399)	100.00	400 (399.1)	95.40
ia-email-EU	1350 (1350)	0.02	1350 (1350)	0.03	1350 (1350)	0.02	1350 (1350)	0.02
ia-email-univ	1473* (1473)	0.00	1473 (1473)	0.00	1473* (1473)	0.00	1473* (1473)	0.00
ia-enron-large	2490 (2490)	0.08	2490 (2490)	0.06	2490 (2490)	0.05	2490 (2490)	0.05
ia-fb-messages	791 (791)	0.00	791 (791)	0.00	791 (791)	0.00	791 (791)	0.00
ia-reality	374* (374)	0.00	374 (374)	2.17	374* (374)	0.00	374* (374)	0.00
ia-wiki-Talk	1884 (1884)	0.51	1884 (1884)	0.08	1884 (1884)	0.06	1884 (1884)	0.06
inf-europe_osm	646* (646)	45.89	430 (399.775)	100.00	646* (505.925)	87.74	646* (505.925)	87.74
inf-germany_osm	597* (597)	10.55	577 (405.25)	100.00	597* (509)	78.84	597* (592.55)	40.83
inf-power	888* (888)	0.00	888 (888)	0.45	888* (888)	0.00	888* (888)	0.00
inf-roadNet-CA	752* (752)	0.41	597 (582.725)	100.00	752* (752)	6.70	752* (752)	2.57
inf-roadNet-PA	669 (669)	0.21	599 (589.925)	100.00	669 (669)	2.35	669 (669)	0.88
inf-road_usa	766* (766)	4.32	532 (422.775)	100.00	766* (766)	34.56	766* (766)	11.09
keller6	5212 (4828.45)	100.00	7968 (7797.77)	98.17	7968 (7781.65)	98.81	7968 (7781.65)	98.81
MANN-a45	33846 (33812.4)	100.00	34197 (34188.4)	97.08	34197 (34187.9)	97.77	34197 (34187.9)	97.77
MANN-a81	110032 (109911)	100.00	111150 (111127)	97.95	111150 (111127)	98.01	111150 (111127)	98.01
p-hat1000-1	1514 (1510.88)	43.19	1514 (1514)	0.08	1514 (1514)	0.09	1514 (1514)	0.09
p-hat1000-2	5696 (5579)	100.00	5777 (5777)	0.00	5777 (5777)	0.01	5777 (5777)	0.01
p-hat1000-3	7798 (7378.2)	100.00	8111 (8111)	0.08	8111 (8111)	0.10	8111 (8111)	0.10
p-hat1500-1	1619 (1573.47)	98.91	1619 (1619)	0.08	1619 (1619)	0.11	1619 (1619)	0.11
p-hat1500-2	6973 (6458.57)	100.00	7360 (7360)	0.04	7360 (7360)	0.07	7360 (7360)	0.07
p-hat1500-3	9422 (8802.08)	100.00	10321 (10321)	6.78	10321 (10321)	7.42	10321 (10321)	7.42
rec-amazon	942* (942)	0.03	942 (899.925)	84.35	942* (942)	0.04	942* (942)	0.04
rec-dating	1320 (1166.03)	100.00	1699 (1699)	0.21	1699 (1699)	0.21	1699 (1699)	0.39
rec-epinions	978 (828.975)	100.00	1054 (1054)	7.55	1054 (1054)	33.79	1054 (1054)	28.78
rec-libimseti-dir	1741 (1416.58)	100.00	1938 (1938)	0.62	1938 (1938)	2.52	1938 (1938)	2.28
rec-movielens	3267 (2788.93)	100.00	3777 (3777)	0.31	3777 (3777)	7.45	3777 (3777)	7.45
rgg_n_2_24_s0	1964 (1964)	9.80	1198 (1047.7)	100.00	1964 (1964)	13.34	1964 (1964)	11.13
rt-retweet-crawl	1367 (1367)	0.66	1367 (1215.12)	69.21	1367 (1327.22)	54.20	1367 (1312.17)	61.24
san1000	1688 (1592.83)	100.00	1716 (1716)	21.02	1716 (1716)	21.11	1716 (1716)	21.11
scc_twitter-copen	58699 (58699)	1.98	58699 (58699)	4.05	58699 (58699)	0.39	58699 (58699)	0.39
sc-ldoor	4081 (4081)	0.91	4074 (3868.78)	100.00	4081 (4058.6)	99.47	4081 (4074.7)	88.60
sc-msdoor	4088 (4088)	0.54	4067 (3925.8)	100.00	4088 (4069.8)	86.82	4088 (4073.12)	87.54
sc-nasasrb	4548 (4548)	0.11	4548 (4469.8)	85.35	4548 (4548)	0.81	4548 (4548)	0.81
sc-pkustk11	5298 (5298)	0.26	5298 (4828.57)	85.94	5298 (5298)	11.50	5298 (5298)	11.50
sc-pkustk13	6306* (6306)	0.29	6306 (5775.52)	96.28	6306* (6306)	1.64	6306* (6306)	1.64

Continued on next page

Table 4 – continued from previous page

Instance	FastWClq		GPULS(CPU)-R		GPULS(CPU)		GPULS	
	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)
sc-pwtk	4620 (4620)	0.24	4620 (4361.7)	98.15	4620 (4620)	8.60	4620 (4620)	4.75
sc-rel9	572 (572)	31.44	572 (410.3)	93.08	498 (404.875)	97.82	572 (435.425)	91.13
sc-shipsec1	3540* (3540)	0.16	3540 (3174.05)	90.28	3540* (3540)	0.72	3540* (3540)	0.76
sc-shipsec5	4524* (4524)	0.15	4524 (4349.7)	92.85	4524* (4524)	0.88	4524* (4524)	0.44
soc-BlogCatalog	4795 (4760.35)	100.00	4803 (4803)	0.66	4803 (4803)	1.49	4803 (4803)	1.49
soc-brightkite	3672 (3672)	0.06	3672 (3655.53)	76.40	3672 (3672)	0.12	3672 (3672)	0.12
soc-buzznet	2981 (2979.22)	44.59	2981 (2981)	0.74	2981 (2981)	9.44	2981 (2981)	10.28
soc-delicious	1547 (1547)	0.53	1547 (1540.6)	83.28	1547 (1547)	8.99	1547 (1547)	10.20
soc-digg	5303 (5295.68)	84.15	5303 (4791.48)	96.42	5303 (5301.43)	48.23	5303 (5302.57)	33.97
soc-dogster	4222 (3657.7)	100.00	4418 (4289.73)	98.50	4418 (4361.48)	97.87	4418 (4343.88)	98.17
soc-douban	1682* (1682)	0.10	1682 (1682)	21.30	1682* (1682)	0.46	1682* (1682)	0.46
soc-epinions	1657 (1657)	0.03	1657 (1657)	1.85	1657 (1657)	0.15	1657 (1657)	0.15
soc-flickr	7050 (6907.95)	100.00	7083 (7073.1)	59.71	7083 (7083)	2.79	7083 (7083)	1.81
soc-flickr-und	9552 (8798.85)	100.00	10126 (8567.98)	100.00	10127 (10121.7)	67.21	10127 (10086.6)	88.58
soc-flixster	3805 (3805)	4.43	3805 (3684.9)	59.98	3805 (3805)	22.58	3805 (3805)	16.10
soc-FourSquare	3064 (3059.97)	71.05	3064 (3055.15)	46.37	3064 (3061.15)	86.71	3064 (3064)	79.49
soc-gowalla	2335 (2335)	0.44	2335 (2246.9)	77.72	2335 (2335)	9.50	2335 (2335)	11.01
soc-lastfm	1773 (1773)	7.97	1773 (1772.55)	25.35	1773 (1773)	7.74	1773 (1773)	7.04
soc-livejournal	21368* (21368)	7.56	19368 (3153.62)	100.00	21368* (21368)	54.92	21368* (21368)	23.39
soc-livejournal-user-groups	642 (439.975)	100.00	1054 (1028.08)	71.34	1054 (916.6)	95.14	1054 (1029.25)	93.66
soc-LiveMocha	1784 (1778.15)	51.99	1784 (1784)	0.06	1784 (1784)	0.15	1784 (1784)	0.16
soc-ljournal-2008	40432 (40432)	43.84	21013 (5442.98)	100.00	40432 (40432)	31.12	40432 (40432)	43.60
soc-orkut	5452 (5451.8)	46.87	4905 (3553.32)	100.00	4969 (3608)	100.00	5452 (3921.62)	98.54
soc-orkut-dir	6147 (5282.52)	99.21	5453 (3855.5)	100.00	6041 (4042.07)	100.00	6147 (4065.57)	99.97
soc-pokec	3191 (3191)	10.03	3191 (2277.7)	94.75	3191 (2125.35)	94.86	3191 (2474.25)	90.90
soc-sinaweibo	621 (292.9)	100.00	**4759** (2543.75)	98.24	4638 (2194.2)	100.00	4638 (2194.2)	100.00
soc-slashdot	2811 (2811)	0.51	2811 (2811)	0.07	2811 (2811)	0.10	2811 (2811)	0.10
soc-twitter-follows	808 (808)	0.53	808 (719.275)	85.13	808 (795.05)	40.90	808 (808)	26.47
soc-twitter-higgs	8039 (7075.7)	98.06	8039 (4974.32)	96.98	8039 (6708.82)	78.56	8039 (5305.52)	93.20
soc-youtube	1961 (1961)	2.76	1961 (1961)	3.49	1961 (1961)	1.80	1961 (1961)	1.02
soc-youtube-snap	1787 (1787)	4.53	1787 (1781.65)	16.78	1787 (1787)	3.99	1787 (1787)	2.61
socfb-A-anon	2872 (2872)	29.51	2872 (2251.2)	98.01	2777 (2212.28)	100.00	2872 (2571.32)	96.22
socfb-B-anon	2662 (2662)	50.15	2537 (2049.1)	100.00	2537 (2152.5)	100.00	2662 (2512.75)	89.98
socfb-Berkeley13	4906 (4906)	0.59	4906 (4906)	6.07	4906 (4906)	1.49	4906 (4906)	1.49
socfb-CMU	4141 (4141)	0.11	4141 (4141)	0.15	4141 (4141)	0.30	4141 (4141)	0.30
socfb-Duke14	3694 (3694)	0.73	3694 (3694)	0.31	3694 (3694)	0.40	3694 (3694)	0.40
socfb-Indiana	5412 (5412)	0.77	5412 (5412)	8.03	5412 (5412)	3.05	5412 (5412)	3.05
socfb-MIT	3658 (3658)	0.31	3658 (3658)	0.27	3658 (3658)	0.27	3658 (3658)	0.27
socfb-OR	3523 (3523)	0.45	3523 (3523)	13.94	3523 (3523)	1.18	3523 (3523)	1.18
socfb-Penn94	4738 (4738)	0.85	4738 (4738)	10.77	4738 (4738)	7.07	4738 (4738)	7.07
socfb-Stanford3	5769 (5769)	0.35	5769 (5769)	2.59	5769 (5769)	1.66	5769 (5769)	1.66
socfb-Texas84	5546 (5546)	2.66	5546 (5543.2)	56.13	5546 (5545.8)	31.50	5546 (5545.8)	31.50
socfb-uci-uni	1045 (1045)	30.79	1045 (542.475)	98.85	802 (617.675)	100.00	802 (617.675)	100.00
socfb-UCLA	5595 (5595)	0.26	5595 (5595)	4.87	5595 (5595)	1.83	5595 (5595)	1.83
socfb-UConn	5733 (5733)	0.24	5733 (5733)	0.75	5733 (5733)	0.47	5733 (5733)	0.47
socfb-UCSB37	5669 (5669)	0.21	5669 (5669)	2.87	5669 (5669)	0.62	5669 (5669)	0.62
socfb-UF	6043 (6043)	1.40	6043 (6042.4)	28.27	6043 (6043)	5.62	6043 (6043)	5.62
socfb-UIllinois	5730 (5730)	3.38	5730 (5728.6)	63.28	5730 (5730)	22.91	5730 (5730)	22.91
socfb-Wisconsin87	4239 (4239)	1.12	4239 (4239)	2.35	4239 (4239)	1.26	4239 (4239)	1.26
tech-as-caida2007	1869 (1869)	0.03	1869 (1869)	0.00	1869 (1869)	0.00	1869 (1869)	0.00
tech-as-skitter	5703 (5703)	11.11	5703 (5031.7)	98.60	5703 (5697.73)	44.76	5703 (5695.43)	46.64
tech-internet-as	1692 (1692)	0.05	1692 (1692)	0.05	1692 (1692)	0.03	1692 (1692)	0.03
tech-ip	423 (225.975)	97.52	668 (668)	1.71	668 (655.325)	14.09	668 (634.45)	25.37
tech-p2p-gnutella	703* (703)	0.07	703 (686.7)	44.16	703* (703)	2.20	703* (703)	2.20
tech-RL-caida	1861 (1861)	0.22	1861 (1861)	2.39	1861 (1861)	0.48	1861 (1861)	0.39
tech-routers-rf	1460* (1460)	0.00	1460 (1460)	0.03	1460* (1460)	0.00	1460* (1460)	0.00
tech-WHOIS	6154 (6154)	0.08	6154 (6154)	1.14	6154 (6154)	0.02	6154 (6154)	0.02
twitter_mpi	6343 (550.45)	100.00	10858 (10836.9)	100.00	10854 (6753.05)	100.00	**12112** (10871.5)	99.65
web-arabic-2005	10558* (10558)	0.06	10558 (10427.6)	88.69	10558* (10558)	0.23	10558* (10558)	0.15

Continued on next page

Table 4 – continued from previous page

Instance	FastWClq		GPULS(CPU)-R		GPULS(CPU)		GPULS	
	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)	Best (Avg.)	t (s)
web-baidu-baike	2340 (959.9)	100.00	1823 (1715.1)	100.00	2651 (1764.88)	100.00	**3814** (1907.92)	99.60
web-BerkStan	3249* (3249)	0.00	3249 (3249)	1.56	3249* (3249)	0.01	3249* (3249)	0.01
web-edu	2077* (2077)	0.00	2077 (2077)	0.38	2077* (2077)	0.00	2077* (2077)	0.00
web-google	1749* (1749)	0.00	1749 (1749)	0.10	1749* (1749)	0.00	1749* (1749)	0.00
web-indochina-2004	6997* (6997)	0.00	6997 (6997)	0.56	6997* (6997)	0.01	6997* (6997)	0.01
web-it-2004	45477* (45477)	0.39	45477 (39712.4)	97.64	45477* (45477)	1.38	45477* (45477)	0.76
web-sk-2005	11925* (11925)	0.03	11925 (10919.9)	79.93	11925* (11925)	0.09	11925* (11925)	0.06
web-spam	2503 (2503)	0.01	2503 (2503)	0.77	2503 (2503)	0.04	2503 (2503)	0.04
web-uk-2005	54850* (54850)	0.71	54850 (54494.2)	33.18	54850* (54850)	0.38	54850* (54850)	0.32
web-webbase-2001	3574* (3574)	0.00	3574 (3574)	9.14	3574* (3574)	0.01	3574* (3574)	0.01
web-wikipedia2009	3891 (3891)	1.24	3455 (1725.83)	100.00	3891 (3891)	14.54	3891 (3891)	10.68
web-wikipedia-growth	1916 (1252.9)	100.00	4741 (2559.03)	97.59	4741 (2839.55)	99.14	3606 (2155.05)	100.00
web-wikipedia_link_it	77607 (11710.1)	100.00	77799 (32026.8)	97.57	77799 (21177)	99.46	77799 (21177)	99.46
wikipedia_link_en	4525 (4381.68)	100.00	1584 (770.525)	100.00	1584 (861.825)	100.00	**4624** (1324.08)	99.94
Avg. t (s)	35.56		49.15		30.08		**29.09**	
# Best (# Best Avg.)	108 (102)		116 (73)		134 (110)		**141 (116)**	
Avg. Gap	4.68		3.33		1.44		**0.85**	

The results of Table 4 reveals that GPULS(CPU) outperforms GPULS(CPU)-R in terms of both solution quality and execution time, which indicates the effectiveness of the reduction procedures. The results also show that GPULS(CPU) is superior in terms of solution quality to FastWClq: GPULS(CPU) missed the best solution in 13 out of the 147 instances, whereas FastWClq missed the best in 39 instances. In 123 instances the average clique weight found by GPULS(CPU) is greater or equal than that found by FastWClq, whereas the same measure is 102 for FastWClq. Moreover, GPULS(CPU) was also 1.18 times faster than FastWClq. These results demonstrate that, even without using the GPU, our heuristic outperforms the state-of-the-art heuristics for the MWCP.

Table 4 also shows that the main version of our heuristic, namely GPULS, outperforms both GPULS(CPU) and FastWClq in terms of solution quality and execution time. GPULS found 141 of the best solutions. Although GPULS makes use of the GPU, its speedup over GPULS(CPU) is much lower than the one observed in Subsection 4.3. This is explained by the fact that the reduction procedures remove a large amount vertices, and, as mentioned in Subsection 4.3, the speedup is correlated with the problem size.

4.5. Comparison with the WLMC exact algorithm

Table 5 compares the performance of GPULS with the WLMC exact algorithm. For each instance, besides the information regarding the best solution found and the CPU time (columns 'Best' and 't (s)'), we also include the time the method takes to read the instance (column 'Read (s)') and the total execution time (column 'Total t (s)'). The total execution time refers to the reading time plus the CPU time to find the best solution. We included the reading time information in this comparison because WLMC initiates several data structures while reading an instance. Column 'Prove (s)' indicates the time WLMC takes to prove the optimal solution. Since WLMC is a nonstochastic algorithm, we ran it only once and considered a cutoff time of 5 hours. When WLMC fails to find the optimal solution, we assume that its CPU time is the cutoff time. GPULS was run 10 times with a cutoff time of 1000 seconds. The main comparison criterion in this experiment is the quality of the solutions found. Due to the differences in the number of runs and cutoff time, running times are provided only for reference purposes.

Table 5: Comparative results of WLMC and GPULS.

	WLMC					GPULS			
Instance	Best	Read (s)	t (s)	Total t (s)	Prove (s)	Best (Avg.)	Read (s)	t (s)	Total t (s)
aff-digg	3836*	58.94	248.08	307.02	679.38	3836 (3836)	4.83	38.04	42.87
aff-flickr-user-groups	1720*	2.95	0.50	3.45	7.32	1720 (1720)	1.95	0.30	2.25
aff-orkut-user2groups	971*	150.71	82.20	232.91	446.10	971 (971)	95.71	63.74	159.45
bio-dmela	805*	0.01	0.01	0.02	0.02	805 (805)	0.00	0.00	0.00
bio-human-gene1	134713*	8.79	1415.38	1424.17	2139.61	134713 (134674)	2.14	847.04	849.18
bio-human-gene2	**135310***	5.88	1031.60	1037.48	1226.49	135301 (135291)	1.55	1000	1001.55
bio-mouse-gene	**59952***	8.47	3229.08	3237.55	3286.03	59943 (59931.6)	2.59	1000	1002.59
bio-yeast	629*	0.02	0.00	0.02	0.02	629* (629)	0.00	0.00	0.00
bn-human-BNU_..._1	**20598***	25.13	659.16	684.29	767.64	20034 (19852.8)	8.10	1000	1008.10
bn-human-BNU_..._2	19189*	12.42	70.30	82.72	84.66	19189 (19189)	4.40	368.44	372.84
C1000-9	8712	0.17	18000.08	18000.25	–	**9254** (9254)	0.07	5.16	5.23
C2000-5	2466	0.37	2765.94	2766.31	–	2466 (2466)	0.16	0.51	0.67
C2000-9	9438	0.80	18000.40	18001.20	–	**10999** (10983.1)	0.28	734.17	734.45
C4000-5	2730	1.87	18000.86	18002.73	–	**2792** (2792)	0.62	27.01	27.63
ca-AstroPh	5338*	0.07	0.04	0.11	0.11	5338* (5338)	0.04	0.08	0.12
ca-citeseer	8838*	0.24	0.02	0.26	0.26	8838* (8838)	0.19	0.14	0.33
ca-coauthors-dblp	37884*	4.69	0.16	4.85	4.85	37884* (37884)	2.97	1.28	4.25
ca-CondMat	2887*	0.04	0.03	0.07	0.07	2887* (2887)	0.02	0.02	0.04
ca-CSphd	489*	0.01	0.00	0.01	0.01	489* (489)	0.00	0.00	0.00
ca-dblp-2010	7575*	0.22	0.02	0.24	0.24	7575* (7575)	0.16	0.08	0.24
ca-dblp-2012	14108*	0.34	0.03	0.37	0.37	14108* (14108)	0.29	0.22	0.51
ca-Erdos992	958*	0.01	0.00	0.01	0.01	958* (958)	0.00	0.00	0.00
ca-GrQc	4279*	0.01	0.00	0.01	0.01	4279* (4279)	0.00	0.01	0.01
ca-HepPh	24533*	0.05	0.01	0.06	0.06	24533* (24533)	0.02	0.01	0.03
ca-hollywood-2009	222720*	23.64	0.87	24.51	24.52	222720* (222720)	11.29	28.85	40.14
ca-MathSciNet	2792*	0.28	0.04	0.32	0.31	2792* (2792)	0.24	0.19	0.43
channel-500x100x100-b050	796*	1.41	0.23	1.64	1.76	796 (796)	0.84	0.97	1.81
dbpedia-link	5062*	40.81	19.65	60.46	60.48	5062 (3768.5)	22.66	913.19	935.85
delaunay_n22	794*	1.84	0.81	2.65	2.65	794* (794)	1.45	6.44	7.89
delaunay_n23	794*	1.87	0.95	2.82	2.82	794* (794)	1.51	10.85	12.36
delaunay_n24	793*	0.82	0.36	1.18	1.19	793* (793)	0.65	12.29	12.94
DSJC1000-5	2186*	0.08	0.03	0.11	184.73	2186 (2186)	0.04	0.05	0.09
frb100-40	6792	8.87	18001.66	18010.53	–	**10501** (10436.4)	1.19	930.75	931.94
friendster	5511*	23.48	4.67	28.15	28.18	5511* (3410.2)	13.55	934.83	948.38
hamming10-2	50512*	0.19	1201.42	1201.61	1201.68	50512 (50512)	0.08	0.10	0.18
hamming10-4	5022	0.17	18000.08	18000.25	–	**5129** (5129)	0.07	1.75	1.82
hugebubbles-00020	400*	1.02	0.54	1.56	1.57	400 (400)	0.84	3.02	3.86
hugetrace-00010	400*	2.00	0.98	2.98	3.03	400 (399.9)	1.68	537.89	539.57
hugetrace-00020	400*	2.26	1.20	3.46	3.80	400 (399.9)	1.80	348.30	350.10
ia-email-EU	1350*	0.03	0.01	0.04	0.04	1350 (1350)	0.01	0.01	0.02
ia-email-univ	1473*	0.00	0.00	0.00	0.00	1473* (1473)	0.00	0.00	0.00
ia-enron-large	2490*	0.06	0.04	0.10	0.12	2490 (2490)	0.04	0.04	0.08
ia-fb-messages	791*	0.00	0.00	0.00	0.00	791 (791)	0.00	0.00	0.00
ia-reality	374*	0.01	0.00	0.01	0.01	374 (374)	0.00	0.01	0.01
ia-wiki-Talk	1884*	0.13	0.10	0.23	0.27	1884 (1884)	0.08	0.04	0.12
inf-europe_osm	646*	18.79	7.69	26.48	26.55	646* (646)	15.07	46.03	61.10
inf-germany_osm	597*	4.12	1.61	5.73	5.90	597* (597)	3.38	22.29	25.67
inf-power	888*	0.01	0.00	0.01	0.01	888* (888)	0.00	0.00	0.00
inf-roadNet-CA	752*	0.86	0.50	1.36	1.39	752* (752)	0.68	1.07	1.75
inf-roadNet-PA	669*	0.46	0.22	0.68	0.69	669 (669)	0.38	0.37	0.75
inf-road_usa	766*	9.54	6.15	15.69	15.90	766* (766)	8.18	15.41	23.59
keller6	5079	2.52	18000.65	18003.17	–	**8038** (7941.1)	0.75	972.30	973.05
MANN-a45	**34265***	0.20	123.50	123.70	355.39	34203 (34195.9)	0.08	1000	1000.08

Continued on next page

Table 5 – continued from previous page

Instance	WLMC					GPULS			
	Best	Read (s)	t (s)	Total t (s)	Prove (s)	Best (Avg.)	Read (s)	t (s)	Total t (s)
MANN-a81	**111343**	3.28	14455.31	14458.59	–	111148 (111138)	0.87	1000	1000.87
p-hat1000-1	1514*	0.05	0.13	0.18	0.54	1514 (1514)	0.02	0.09	0.11
p-hat1000-2	5777*	0.09	1.99	2.08	70.91	5777 (5777)	0.04	0.01	0.05
p-hat1000-3	8111	0.13	3636.67	3636.80	–	8111 (8111)	0.05	0.06	0.11
p-hat1500-1	1619*	0.09	0.71	0.80	4.47	1619 (1619)	0.04	0.09	0.13
p-hat1500-2	7360*	0.20	14.48	14.68	6204.81	7360 (7360)	0.09	0.06	0.15
p-hat1500-3	10321	0.32	653.89	654.21	–	10321 (10321)	0.12	8.53	8.65
rec-amazon	942*	0.04	0.10	0.14	0.14	942* (942)	0.03	0.04	0.07
rec-dating	1699*	8.43	1.37	9.80	21.13	1699 (1699)	3.90	0.28	4.18
rec-epinions	1054*	28.75	1.62	30.37	32.47	1054 (1054)	3.52	12.58	16.10
rec-libimseti-dir	1938*	8.85	2.09	10.94	19.63	1938 (1938)	3.74	0.85	4.59
rec-movielens	3777*	3.52	3.76	7.28	29.16	3777 (3777)	1.82	2.27	4.09
rgg_n_2_24_s0	1964*	0.30	0.14	0.44	0.45	1964 (1964)	0.27	5.23	5.50
rt-retweet-crawl	1367*	0.92	0.14	1.06	1.06	1367 (1367)	0.71	86.74	87.45
san1000	1716*	0.09	0.55	0.64	1.17	1716 (1716)	0.04	28.16	28.20
scc_twitter-copen	58699*	0.22	6.77	6.99	6.98	58699 (58699)	0.08	0.13	0.21
sc-ldoor	4081*	6.07	0.72	6.79	6.91	4081 (4081)	3.98	145.87	149.85
sc-msdoor	4088*	2.75	0.27	3.02	3.08	4088 (4088)	1.78	46.12	47.90
sc-nasasrb	4548*	0.36	0.44	0.80	0.80	4548 (4548)	0.24	0.56	0.80
sc-pkustk11	5298*	0.73	0.66	1.39	1.40	5298 (5298)	0.46	12.22	12.68
sc-pkustk13	6306*	0.94	1.47	2.41	2.41	6306* (6306)	0.58	0.87	1.45
sc-pwtk	4620*	1.64	0.17	1.81	1.81	4620 (4620)	1.01	2.94	3.95
sc-rel9	572*	9.35	1.37	10.72	10.72	572 (443.3)	7.32	804.68	812.00
sc-shipsec1	3540*	0.50	0.07	0.57	0.57	3540 (3540)	0.33	0.56	0.89
sc-shipsec5	4524*	0.62	0.05	0.67	0.67	4524* (4524)	0.42	0.35	0.77
soc-BlogCatalog	4803*	0.68	0.86	1.54	3.48	4803 (4803)	0.40	0.65	1.05
soc-brightkite	3672*	0.07	0.11	0.18	0.18	3672 (3672)	0.04	0.09	0.13
soc-buzznet	2981*	1.88	1.21	3.09	3.48	2981 (2981)	0.52	12.86	13.38
soc-delicious	1547*	0.45	0.07	0.52	0.53	1547 (1547)	0.34	4.31	4.65
soc-digg	5303*	2.60	3.88	6.48	6.96	5303 (5303)	1.39	14.72	16.11
soc-dogster	4418*	2.92	1.01	3.93	6.20	4418 (4416.6)	1.95	485.81	487.76
soc-douban	1682*	0.09	0.01	0.10	0.10	1682 (1682)	0.07	0.43	0.50
soc-epinions	1657*	0.04	0.03	0.07	0.08	1657 (1657)	0.02	0.12	0.14
soc-flickr	7083*	1.08	0.37	1.45	5.01	7083 (7083)	0.73	0.90	1.63
soc-flickr-und	10127*	8.62	100.78	109.40	157.51	10127 (10127)	3.52	146.80	150.32
soc-flixster	3805*	3.09	0.37	3.46	3.60	3805 (3805)	2.17	7.01	9.18
soc-FourSquare	3064*	9.30	0.15	9.45	9.60	3064 (3064)	0.78	22.27	23.05
soc-gowalla	2335*	0.30	0.08	0.38	0.38	2335 (2335)	0.22	4.52	4.74
soc-lastfm	1773*	1.76	0.23	1.99	2.23	1773 (1773)	1.26	2.42	3.68
soc-livejournal	21368*	10.94	1.72	12.66	12.66	21368* (21368)	8.09	23.81	31.90
soc-livejournal-user-groups	1054*	38.94	10.00	48.94	88.72	1054 (1054)	32.20	348.73	380.93
soc-LiveMocha	1784*	0.74	0.18	0.92	1.27	1784 (1784)	0.46	0.09	0.55
soc-ljournal-2008	40432*	31.43	5.30	36.73	36.74	40432 (40432)	11.93	24.07	36.00
soc-orkut-dir	6147*	54.33	32.46	86.79	92.51	6147 (6079.6)	32.31	717.47	749.78
soc-orkut	**5452***	46.73	32.72	79.45	80.70	4969 (4610.8)	29.43	1000	1029.43
soc-pokec	3191*	9.73	1.78	11.51	12.62	3191 (3191)	6.44	117.66	124.10
soc-sinaweibo	4759*	238.35	26.37	264.72	279.22	4759 (4145.4)	119.41	911.31	1030.72
soc-slashdot	2811*	0.10	0.13	0.23	0.25	2811 (2811)	0.07	0.08	0.15
soc-twitter-follows	808*	0.25	0.04	0.29	0.30	808 (808)	0.18	7.48	7.66
soc-twitter-higgs	8039*	5.15	2.71	7.86	7.90	8039 (8039)	3.18	157.68	160.86
soc-youtube	1961*	0.64	0.10	0.74	0.85	1961 (1961)	0.50	0.75	1.25
soc-youtube-snap	1787*	0.99	0.20	1.19	1.29	1787 (1787)	0.79	1.39	2.18
socfb-A-anon	2872*	9.68	1.94	11.62	13.62	2872 (2872)	6.61	182.16	188.77
socfb-B-anon	2662*	8.70	1.86	10.56	12.40	2662 (2662)	5.85	250.91	256.76
socfb-Berkeley13	4906*	0.26	0.27	0.53	0.71	4906 (4906)	0.16	0.80	0.96
socfb-CMU	4141*	0.08	0.07	0.15	0.20	4141 (4141)	0.04	0.14	0.18
socfb-Duke14	3694*	0.16	0.10	0.26	0.46	3694 (3694)	0.09	0.22	0.31
socfb-Indiana	5412*	0.40	0.57	0.97	1.22	5412 (5412)	0.26	2.59	2.85
socfb-MIT	3658*	0.08	0.04	0.12	0.20	3658 (3658)	0.04	0.21	0.25

Continued on next page

Table 5 – continued from previous page

	WLMC					GPULS			
Instance	Best	Read (s)	t (s)	Total t (s)	Prove (s)	Best (Avg.)	Read (s)	t (s)	Total t (s)
socfb-OR	3523*	0.25	0.46	0.71	0.76	3523 (3523)	0.16	0.60	0.76
socfb-Penn94	4738*	0.43	0.91	1.34	1.34	4738 (4738)	0.26	5.80	6.06
socfb-Stanford3	5769*	0.18	0.15	0.33	0.51	5769 (5769)	0.10	1.42	1.52
socfb-Texas84	5546*	0.51	0.65	1.16	1.52	5546 (5546)	0.30	27.78	28.08
socfb-uci-uni	1045*	44.49	7.31	51.80	51.82	1045 (1019.1)	33.86	634.82	668.68
socfb-UCLA	5595*	0.24	0.23	0.47	0.60	5595 (5595)	0.14	1.23	1.37
socfb-UConn	5733*	0.20	0.16	0.36	0.38	5733 (5733)	0.11	0.26	0.37
socfb-UCSB37	5669*	0.16	0.13	0.29	0.28	5669 (5669)	0.09	0.39	0.48
socfb-UF	6043*	0.46	0.66	1.12	1.34	6043 (6043)	0.28	5.05	5.33
socfb-UIllinois	5730*	0.39	0.57	0.96	1.19	5730 (5730)	0.24	17.76	18.00
socfb-Wisconsin87	4239*	0.26	0.46	0.72	0.72	4239 (4239)	0.16	1.07	1.23
tech-as-caida2007	1869*	0.02	0.01	0.03	0.03	1869 (1869)	0.01	0.00	0.01
tech-as-skitter	5703*	8.62	0.39	9.01	9.22	5703 (5703)	2.68	23.37	26.05
tech-internet-as	1692*	0.03	0.02	0.05	0.05	1692 (1692)	0.02	0.03	0.05
tech-ip	668*	86.58	6.27	92.85	93.25	668 (588.8)	4.58	500.13	504.71
tech-p2p-gnutella	703*	0.06	0.15	0.21	0.20	703 (703)	0.03	2.14	2.17
tech-RL-caida	1861*	0.18	0.04	0.22	0.23	1861 (1861)	0.16	0.21	0.37
tech-routers-rf	1460*	0.01	0.00	0.01	0.01	1460* (1460)	0.00	0.00	0.00
tech-WHOIS	6154*	0.03	0.00	0.03	0.04	6154 (6154)	0.01	0.02	0.03
twitter_mpi	13524*	301.97	1513.78	1815.75	1850.84	13524 (13486.4)	26.84	884.91	911.75
web-arabic-2005	10558*	0.52	0.02	0.54	0.54	10558* (10558)	0.33	0.10	0.43
web-baidu-baike	3814*	7.01	2.99	10.00	10.00	3814 (3415.5)	4.98	749.92	754.90
web-BerkStan	3249*	0.01	0.00	0.01	0.02	3249* (3249)	0.00	0.01	0.01
web-edu	2077*	0.01	0.00	0.01	0.01	2077* (2077)	0.00	0.00	0.00
web-google	1749*	0.02	0.00	0.02	0.02	1749* (1749)	0.00	0.00	0.00
web-indochina-2004	6997*	0.02	0.00	0.02	0.02	6997* (6997)	0.01	0.01	0.02
web-it-2004	45477*	2.14	0.05	2.19	2.19	45477* (45477)	1.35	0.49	1.84
web-sk-2005	11925*	0.10	0.00	0.10	0.11	11925* (11925)	0.07	0.05	0.12
web-spam	2503*	0.03	0.00	0.03	0.03	2503 (2503)	0.01	0.04	0.05
web-uk-2005	54850*	3.53	0.04	3.57	3.57	54850* (54850)	1.98	0.24	2.22
web-webbase-2001	3574*	0.01	0.00	0.01	0.02	3574* (3574)	0.00	0.01	0.01
web-wikipedia2009	3891*	2.01	0.53	2.54	2.56	3891 (3891)	1.54	4.67	6.21
web-wikipedia-growth	4741*	17.84	9.15	26.99	26.99	4741 (4267.1)	10.34	622.40	632.74
web-wikipedia_link_it	89947*	40.28	60.88	101.16	101.36	89947 (73385.3)	21.14	971.25	992.39
wikipedia_link_en	4624*	15.50	3.82	19.32	20.68	4624 (3565.4)	9.38	797.30	806.68
Avg. Total t (s)			959.39			159.00			
# Best			141			141			
Avg. Gap			0.0066			0.0008			

GPULS and WLMC fail to find the best solution on the same number of instances. The following lists present the gap values of both methods on these instances:

- WLMC gaps: `C1000-9` — 5.85693, `C2000-9` — 14.1922, `C4000-5` — 2.22063, `frb100--40` — 35.3204, `hamming10-4` — 2.08618, and `keller6` — 36.8126.
- GPULS gaps: `bn-human-BNU_..._1` — 2.73813, `bio-human-gene2` — 0.00665139, `bio-mouse-gene` — 0.015012, `MANN-a45` — 0.180943, `MANN-a81` — 0.175134, and `soc--orkut` — 8.85913.

As can be seen from these lists and from the bottom of Table 5, WLMC gaps are much wider than GPULS gaps. The average gap of WLMC is 8.25 times larger than that of GPULS. While WLMC gaps are always greater than 2%, in only two instances GPULS gap is greater than 2%. Moreover, all instances in which WLMC fails to find the best solutions are contained in the DIMACS & BHOSLIB benchmark. This fact indicates that WLMC is not well suited for large and dense instances. GPULS, on the other hand, performs generally well on all kinds of instances.

4.6. Portfolio selection application

Table 6 presents the results regarding the Market Graph instances described in Section 4.1. For each instance, the table indicates the best (average in parenthesis) solution found by each method. For this experiment, we ran each heuristic method 10 times using a cutoff time of 100 seconds. WLMC was executed only once with a cutoff time of 5 hours. The results show that GPULS(CPU) attained the best solutions among the methods. WLMC was able to prove two optimal solutions (the sparser instances), and missed the best solution in one instance.

Table 6: Comparative results on Market Graph instances.

Method \ Instance	mg_usa_assets_00	mg_usa_assets_05	mg_worldwide_stocks_00	mg_worldwide_stocks_05
LSCC	328450 (328369.00)	607712 (571551.20)	512619 (509942.65)	998245 (991070.00)
MNTS	328450 (328139.62)	619389 (574997.72)	512619 (509311.30)	995716 (991089.10)
RETS	328450 (328450.00)	630720 (630720.00)	512619 (510904.70)	995716 (994686.10)
GPULS(CPU)	328450 (328450.00)	630720 (630448.50)	512619 (512619.00)	998245 (998245.00)
WLMC	328450*	619058	512619*	998245

5. Concluding remarks

This paper proposed GPULS, a CPU-GPU multi-start local search heuristic for solving the MWCP on massive graphs. Our approach leverages the GPUs massively parallel power and high memory bandwidth to solve the problem. We remark that, up to the authors knowledge, we are the first to investigate the use of GPUs on the MWCP. We also proposed two new neighborhood structures, which are explored using a tabu search and a first-improvement approach.

Computational results have shown that, even when GPULS was executed on a CPU-only architecture, it was faster and found better solutions than the state-of-the-art heuristics. Moreover, the results also indicate that, on massive instances that are not sparse, GPULS outperforms the best exact method for the problem. The speedups compared to the sequential GPULS varied up to 12 times, achieving better speedups as the size of the instances grow.

As future work, the proposed GPULS could be extended to solve related clique problems, for instance, the maximum independent set problem or maximum quasi-clique problem. Future research also includes the development of a hybrid approach combining the advantages of GPULS with an exact method.

Acknowledgments

We would like to thank Prof. Shaowei Cai for providing the source codes of LSCC+BMS, MN/TS+BMS and FastWClq; Prof. Jin-Kao Hao for making available the binary of ReTS-I; and Prof. Hua Jiang for providing the Network Data Repository instances. We would also like to thank the anonymous reviewers for their useful comments and suggestions.

References

[1] R. Karp, Reducibility among combinatorial problems, in: R. Miller, J. Thatcher (Eds.), Complexity of Computer Computations, Plenum Press, 1972, pp. 85–103.

[2] Q. Wu, J.-K. Hao, Solving the winner determination problem via a weighted maximum clique heuristic, Expert Syst Appl 42 (1) (2015) 355–365.

[3] J. Liu, Y. T. Lee, Graph-based method for face identification from a single 2d line drawing, IEEE Trans Pattern Anal Mach Intell 23 (10) (2001) 1106–1119.

[4] M. Pelillo, K. Siddiqi, S. W. Zucker, Matching hierarchical structures using association graphs, IEEE Trans Pattern Anal Mach Intell 21 (11) (1999) 1105–1120.

[5] M. Tepper, G. Sapiro, Ants crawling to discover the community structure in networks, in: Iberoamerican Congress on Pattern Recognition, Springer, 2013, pp. 552–559.

[6] V. Boginski, S. Butenko, O. Shirokikh, S. Trukhanov, J. G. Lafuente, A network-based data mining approach to portfolio selection via weighted clique relaxations, Ann Oper Res 216 (1) (2014) 23–34.

[7] C. Zheng, Q. Zhu, D. Sankoff, Removing noise and ambiguities from comparative maps in rearrangement analysis, IEEE/ACM Trans Comp Bio Bioinf 4 (4) (2007) 515–522.

[8] C. T. Ng, T. C. E. Cheng, A. M. Bandalouski, M. Y. Kovalyov, S. S. Lam, A graph-theoretic approach to interval scheduling on dedicated unrelated parallel machines, J Oper Res Soc 65 (10) (2013) 1571–1579.

[9] U. Feige, Approximating maximum clique by removing subgraphs, SIAM J Discrete Math 18 (2) (2004) 219–225.

[10] D. Zuckerman, Linear degree extractors and the inapproximability of max clique and chromatic number, in: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing, ACM, 2006, pp. 681–690.

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

[12] Q. Wu, J.-K. Hao, F. Glover, Multi-neighborhood tabu search for the maximum weight clique problem, Ann Oper Res 196 (1) (2012) 611–634.

[13] U. Benlic, J.-K. Hao, Breakout local search for maximum clique problems, Comput Oper Res 40 (1) (2013) 192–206.

[14] Y. Wang, S. Cai, M. Yin, Two efficient local search algorithms for maximum weight clique problem, in: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, 2016, pp. 805–811.

[15] Y. Zhou, J.-K. Hao, A. Goëffon, Push: A generalized operator for the maximum vertex weight clique problem, Eur J Oper Res 257 (1) (2017) 41–54.

[16] B. Nogueira, R. G. S. Pinheiro, A. Subramanian, A hybrid iterated local search heuristic for the maximum weight independent set problem, Optim Lett (2017) 1–17.

[17] E. Balas, W. Niehaus, Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems, J Heuristics 4 (2) (1998) 107–122.

[18] I. Bomze, M. Pelillo, V. Stix, Approximating the maximum weight clique using replicator dynamics, IEEE Trans Neural Netw 11 (6) (2000) 1228–1241.

[19] Y. Wang, J.-K. Hao, F. Glover, Z. Lü, Q. Wu, Solving the maximum vertex weight clique problem via binary quadratic programming, J Comb Optim 32 (2) (2016) 531–549.

[20] S. Cai, J. Lin, Fast solving maximum weight clique problem in massive graphs, in: Proceedings of 25th International Joint Conference on Artificial Intelligence (IJCAI), 2016, pp. 568–574.

[21] H. Jiang, C.-M. Li, F. Manya, An exact algorithm for the maximum weight clique problem in large graphs., in: Proceedings of AAAI Conference on Artificial Intelligence, 2017, pp. 830–838.
[22] M. Gendreau, J.-Y. Potvin, Handbook of Metaheuristics, Springer, 2010, Ch. Tabu Search, pp. 41–60.
[23] R. Martí, J. M. Moreno-Vega, A. Duarte, Handbook of Metaheuristics, Springer, 2010, Ch. Advanced multi-start methods, pp. 265–281.
[24] M. Harris, et al., Optimizing parallel reduction in cuda, NVIDIA Developer Technology 2 (4).
[25] D. V. Andrade, M. G. Resende, R. F. Werneck, Fast local search for the maximum independent set problem, J Heuristics 18 (4) (2012) 525–547.