A GPU-based Iterated Tabu Search for Solving the Quadratic 3-dimensional Assignment Problem

Thé Van Luong, Nouredine Melab, El-Ghazali Talbi

A GPU-based Iterated Tabu Search for Solving the Quadratic 3-dimensional Assignment Problem

Thé Van Luong*, Lakhdar Loukil †, Nouredine Melab* and El-Ghazali Talbi*
* INRIA Lille Nord Europe - CNRS/LIFL Laboratory
40, avenue Halley, Bt. A, Park Plaza
59650 Villeneuve d'Asq, France
Email: The-Van.Luong@inria.fr, Nouredine.Melab@lifl.fr, Email: El-Ghazali.Talbi@lifl.fr
†Computer Science Department, Faculty of Sciences
Oran University
BP. 1524 El M'Naouer, Oran, Algeria
Email: Loukil.Lakhdar@univ-oran.dz

***Abstract*—The quadratic 3-dimensional assignment problem (Q3AP) is an extension of the well-known NP-hard quadratic assignment problem. It has been proved to be one of the most difficult combinatorial optimization problems. Local search (LS) algorithms are a class of heuristics which have been successfully applied to solve such hard optimization problem. These methods handle with a single solution iteratively improved by exploring its neighborhood in the solution space. In this paper, we propose an iterated tabu search for solving the Q3AP. The design of this algorithm is essentially based on a new large neighborhood structure. Indeed, in LS heuristics, designing operators to explore large promising regions of the search space may improve the quality of the obtained solutions. However, designing such neighborhood is at the expense of a highly computationally process. Therefore, the use of graphics processing units (GPUs) provides an efficient complementary way to speed up the search. The proposed GPU-based iterated tabu search has been experimented on 5 different Q3AP instances. The obtained results are convincing both in terms of efficiency, quality and robustness of the provided solutions at run time.**

***Index Terms*—Quadratic 3-dimensional assignment problem (Q3AP), GPU-based local search, metaheuristics on GPU, iterated tabu search on graphics processing units.**

I. Introduction

The Quadratic 3-dimensional Assignment Problem is a combinatorial optimization problem introduced by Pierskalla [1] in 1967. The Q3AP consists in finding an optimal symbol-mapping over two vectors so as to minimize an objective function. An application example of the Q3AP is the Hybrid Automatic Repeat reQuest (Hybrid-ARQ) error-control mechanism used in wireless communication systems to detect altered bits in transmitted packets [2].

To the best of our knowledge, there is not much work devoted to solving the Q3AP. The most important work found in the literature is that achieved by Peter Hahn's team at the University of Pennsylvania in collaboration with researchers from the University of California, Davis [3], [4]. In this work, LS algorithms have been successfully applied for the resolution of Q3AP instances. Indeed, these methods are single solution-based metaheuristics which have been successfully applied for solving many real and complex problems [5]. LS methods could be viewed as "walks through neighborhoods" meaning search trajectories through the solutions domains of the problems at hand. The walks are performed by iterative procedures that allow to move from a solution to another one in the solution space.

The definition of the neighborhood plays a crucial role in the performance of a LS method. Since the neighborhood structure strongly depends on the target optimization problem, we focus on designing a new large neighborhood adapted to the Q3AP all along of this paper. Indeed, theoretical and experimental studies have shown that the increase of the neighborhood size may improve the quality of provided solutions of LS algorithms [6]. Nevertheless, as it is generally CPU time-consuming it is not often fully exploited in practice. Indeed, experiments with large neighborhood algorithms are often stopped without convergence being reached. That is the reason why, in designing LS methods, there is often a compromise between the size of the neighborhood to use and the computational complexity to explore it. As a consequence, in LS algorithms, there is often a reduction of the size of the explored neighborhood at the expense of the effectiveness. To deal with such issues, only the use of parallelism allows to design algorithms based on large neighborhoods.

Nowadays, GPU computing is recognized as a powerful way to achieve high-performance on long-running scientific applications [7]. GPU is a dedicated graphics rendering device for computers that provide tremendous parallel execution capabilities and fast memory access. Nevertheless, the use of GPU-based parallel computing for metaheuristics is not straightforward. Indeed, several scientific challenges mainly related to the hierarchical memory management have to be faced. As a consequence, designing LS algorithms based on large neighborhood structures for solving real-world optimization problems are good challenges for GPU computing.

In the present work, we propose a hierarchical parallel iterated tabu search algorithm on GPU for solving Q3AP problems. Tabu search (TS) method is a deterministic local search metaheuristic used to solve combinatorial optimization problems. Its search intensification capability and its ease to

implementation make it largely used for solving a number of challenging optimization problems such as the quadratic assignment problem. The motivation behind the use of a tabu search combined with a large neighborhood in solving the Q3AP is to explore larger search space by exploring various search space sub-areas. Therefore, we investigate to measure the impact on how the increase of the size of the neighborhood can improve the quality of the obtained solutions.

The remainder of the paper is organized as follows: Section II gives the mathematical formulation of the Q3AP. Our proposed neighborhood for LS algorithms and the commonly used one are presented in Section III. In Section IV, for a better understanding of the difficulties of using the GPU architecture, GPU computing for metaheuristics is described. Section V presents high-level concepts for the design and the implementation of a tabu search on GPU. Section VI reports the performance results obtained for the implemented iterated tabu search for the Q3AP. Finally, a discussion and some conclusions of this work are drawn in Section VII.

II. Q3AP Formulation

The quadratic 3-dimensional assignment problem (Q3AP) is an extension of the quadratic assignment problem and of the axial 3-dimensional assignment problem (3AP). Q3AP was introduced by William P. Pierskalla in 1967 [1] and has recently been used to model some advanced assignment problems like the symbol-mapping problem posed in wireless communication systems and described in [3]. The Q3AP can be formulated as follows:

$$min \left\{ \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{l=0}^{n-1}\sum_{k=0}^{n-1}\sum_{s=0}^{n-1}\sum_{r=0}^{n-1} c_{ijlksr}x_{ijl}x_{ksr} \quad + \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{l=0}^{n-1} b_{ijl}x_{ijl} \right\} \tag{1}$$

where:

$$X = (x_{ijl}) \in I \cap J \cap L, \tag{2}$$

$$x_{ijl} \in \{0,1\}, \quad i,j,l = 0,1,...,n-1. \tag{3}$$

I, J and L sets are defined as follows:

$$I = \{X = (x_{ijl}) : \sum_{j=0}^{n-1}\sum_{l=0}^{n-1} x_{ijl} = 1, \quad i = 0,1,...,n-1\};$$

$$J = \{X = (x_{ijl}) : \sum_{i=0}^{n-1}\sum_{l=0}^{n-1} x_{ijl} = 1, \quad j = 0,1,...,n-1\};$$

$$L = \{X = (x_{ijl}) : \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} x_{ijl} = 1, \quad l = 0,1,...,n-1\}.$$

Whereas for the QAP the problem is to find a 2-dimensional permutation matrix that minimizes a quadratic function, the problem for the Q3AP is to minimize a quadratic function over the 3-dimensional assignment polytope $I \cap J \cap P$. That is the reason why this problem is referred as the quadratic 3-dimensional assignment problem.

An alternative formulation that is frequently used is the *permutation-based formulation*. The Q3AP given by Eqs. (1)-(3) can be expressed in permutation-based formulation as follows:

$$min \left\{ f(p,q) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ip_{(i)}q_{(i)}jp_{(j)}q_{(j)}} \quad + \sum_{i=1}^{n} b_{ip_{(i)}q_{(i)}} \right\} \tag{4}$$

where p and q are permutations over the set $\{0,1,\ldots,n-1\}$. According to this formulation, minimizing the Q3AP consists in finding a double permutation (p,q) which minimizes (4).

As mentioned earlier in this section, the Q3AP is an extension of the QAP and of the axial 3-dimensional assignment problem which are both NP-hard problems. Therefore, the Q3AP is proved to be also a NP-hard problem. Furthermore, this problem is particularly difficult since the number of feasible solutions of an instance of size n is $n! \times n!$.

III. Local Search Neighborhoods for the Q3AP

A. Formalization of Neighborhoods

The definition of the neighborhood is a required common step for the design of any LS algorithm. Indeed, If the neighborhood structure is not adequate to the problem, any LS metaheuristic will fail to solve the problem. Regarding the Q3AP, a basic neighborhood for LS algorithms has been proposed in [8] and investigated in different works of the literature [4], [9]. This neighborhood is based on the Q3AP permutation-based formulation (4) where each generated solution is obtained by an exchange of two positions in either the first permutation p or the second q. The number of generated neighbors is thus equal to $n \times (n-1)$. Formally, the basic neighborhood can be expressed as follows:

$$\begin{aligned} N_{Basic}(p,q) = \{ & \quad (p',q') : p'[k] = p[l], p'[l] = p[k] \\ & \quad 0 \leq k \neq l < n; \\ & \quad p'[i] = p[i], \quad 0 \leq i \neq k,l < n; \\ & \quad q'[j] = q[j], \quad 0 \leq j < n \\ & \} \\ & \bigcup \\ & \{ \quad (p',q') : q'[k] = q[l], q'[l] = q[k] \\ & \quad 0 \leq k \neq l < n; \\ & \quad q'[i] = q[i], \quad 0 \leq i < n; \\ & \quad p'[j] = p[j], \quad 0 \leq j \neq k,l < n \\ & \} \end{aligned} \tag{5}$$

For different optimization problems, theoretical and experimental studies have shown that the increase of the neighborhood size may improve the effectiveness of LS algorithms

[6]. As a consequence, regarding the Q3AP, we propose an advanced neighborhood where more candidate solutions are evaluated. It consists in exchanging two positions in both permutations p and q. Therefore, the size of this new neighborhood is equal to $(\frac{n\times(n-1)}{2})^2$. A formal definition of the advanced neighborhood can be expressed as follows:

$$\begin{aligned} N_{Advanced}(p,q) = \{ \quad & (p',q') : p'[k] = p[l], p'[l] = p[k], \\ & q'[r] = q[s], q'[s] = q[r] \\ & 0 \leq k \neq l < n, 0 \leq r \neq s < n; \\ & p'[i] = p[i], \quad 0 \leq i \neq k, l < n; \\ & q'[j] = q[j], \quad 0 \leq j \neq r, s < n \\ \} & \end{aligned} \tag{6}$$

B. Incremental Evaluation Functions

According to the Q3AP permutation-based formulation, a full evaluation of a solution of size n requires the computation of n^2 cost terms c_{ijklmn}. A more efficient way to evaluate the set of neighboring candidates is to consider the incremental evaluation (or partial evaluation). It consists in evaluating only the move transformation applied to a solution (Δ calculation) rather than the complete evaluation of the objective function.

For the first basic neighborhood, an incremental evaluation function has been proposed in [8] where the evaluation of a neighbor requires the computation of $8(n+1)$ cost terms. The Δ computation is given by:

- If the swapping of positions k and l ($k \neq l$) occurs in p:

$$\Delta = \sum_{j=1}^{n} \left\{ \begin{array}{l} - c_{kp(k)q(k)ip(i)q(i)} + c_{jp(i)q(j)kp(k)q(k)} \\ + c_{kp(l)q(l)ip(i)q(i)} - c_{lp(l)q(l)ip(i)q(i)} \\ - c_{ip(i)q(i)kp(k)q(k)} + c_{ip(i)q(i)lp(k)q(l)} \\ + c_{jp(j)q(j)kp(l)q(k)} - c_{jp(j)q(j)kp(k)q(k)} \end{array} \right\}$$

$$\Delta \mathrel{+}= \left\{ \begin{array}{l} - c_{kp(k)q(k)lp(l)q(l)} + c_{kp(l)q(k)lp(k)q(l)} \\ - c_{lp(l)q(l)kp(k)q(k)} + c_{lp(l)q(l)kp(l)q(k)} \\ - c_{kp(k)q(k)kp(k)q(k)} + c_{kp(k)q(k)kp(l)q(k)} \\ + c_{lp(k)q(l)lp(k)q(l)} - c_{lp(l)q(l)lp(l)q(l)} \end{array} \right\}$$

- If the swapping of positions k and l ($k \neq l$) occurs in q:

$$\Delta = \sum_{i=1}^{n} \left\{ \begin{array}{l} - c_{kq(i)p(k)iq(i)p(i)} + c_{lq(k)p(l)iq(i)p(i)} \\ + c_{kq(j)p(k)iq(i)p(i)} - c_{lq(l)p(l)iq(i)p(i)} \\ - c_{iq(k)p(i)kq(k)p(k)} + c_{iq(i)p(i)lq(k)p(l)} \\ + c_{iq(k)p(i)kq(l)p(k)} - c_{iq(i)p(i)lq(l)p(l)} \end{array} \right\}$$

$$\Delta \mathrel{+}= \left\{ \begin{array}{l} - c_{kq(k)p(k)lq(l)p(l)} + c_{kq(l)p(k)lq(k)p(l)} \\ - c_{lq(l)p(l)kq(k)p(k)} + c_{lq(k)p(l)kq(l)p(k)} \\ - c_{kq(k)p(k)kq(k)p(k)} + c_{kq(l)p(k)kq(l)p(k)} \\ + c_{lq(k)p(l)lq(k)p(l)} - c_{lq(l)p(l)lq(l)p(l)} \end{array} \right\}$$

Since exchanges are performed in both p and q permutations, the incremental evaluation function of our proposed advanced neighborhood is more complex. As a consequence, only a part on the Δ computation is described in this paper. Let k and l be the swapped positions in the first permutation p and let r and s be the swapped positions in the second permutation q. If we suppose that $k \neq r$ and $l \neq s$, the Δ computation is given by:

$$\Delta = \sum_{\substack{i=0 \\ i\neq k,l \\ i\neq r,s}}^{n-1} \left\{ \begin{array}{l} c_{ip(i)q(i)kp(l)q(k)} - c_{ip(i)q(i)kp(k)q(k)} \\ + c_{ip(i)q(i)lp(k)q(l)} - c_{ip(i)q(i)lp(l)q(l)} \\ + c_{ip(i)q(i)rp(r)q(s)} - c_{ip(i)q(i)rp(r)q(r)} \\ + c_{ip(i)q(i)sp(s)q(r)} - c_{ip(i)q(i)sp(s)q(s)} \end{array} \right\}$$

$$\Delta \mathrel{+}= \left\{ \begin{array}{l} c_{kp(l)q(k)kp(l)q(k)} - c_{kp(k)q(k)kp(k)q(k)} \\ + c_{kp(l)q(k)lp(k)q(l)} - c_{kp(k)q(k)lp(l)q(l)} \\ + c_{kp(l)q(k)rp(r)q(s)} - c_{kp(k)q(k)rp(r)q(r)} \\ + c_{kp(l)q(k)sp(s)q(r)} - c_{kp(k)q(k)sp(s)q(s)} \end{array} \right\}$$

$$\Delta \mathrel{+}= \left\{ \begin{array}{l} c_{lp(k)q(l)kp(l)q(k)} - c_{lp(l)q(l)kp(k)q(k)} \\ + c_{lp(k)q(l)lp(k)q(l)} - c_{lp(l)q(l)lp(l)q(l)} \\ + c_{lp(k)q(l)rp(r)q(s)} - c_{lp(l)q(l)rp(r)q(r)} \\ + c_{lp(k)q(l)sp(s)q(r)} - c_{lp(l)q(l)sp(s)q(s)} \end{array} \right\}$$

$$\Delta \mathrel{+}= \left\{ \begin{array}{l} c_{rp(r)q(s)kp(l)q(k)} - c_{rp(r)q(r)kp(k)q(k)} \\ + c_{rp(r)q(s)lp(k)q(l)} - c_{rp(r)q(r)lp(l)q(l)} \\ + c_{rp(r)q(s)rp(r)q(s)} - c_{rp(r)q(r)rp(r)q(r)} \\ + c_{rp(r)q(s)sp(s)q(r)} - c_{rp(r)q(r)sp(s)q(s)} \end{array} \right\}$$

$$\Delta \mathrel{+}= \left\{ \begin{array}{l} c_{sp(s)q(r)kp(l)q(k)} - c_{sp(s)q(s)kp(k)q(k)} \\ + c_{sp(s)q(r)lp(k)q(l)} - c_{sp(s)q(s)lp(l)q(l)} \\ + c_{sp(s)q(r)rp(r)q(s)} - c_{sp(s)q(s)rp(r)q(r)} \\ + c_{sp(s)q(r)sp(s)q(r)} - c_{sp(s)q(s)sp(s)q(s)} \end{array} \right\}$$

$$\Delta \mathrel{+}= \sum_{\substack{j=0 \\ j\neq k,l \\ j\neq r,s}}^{n-1} \left\{ c_{kp(l)q(k)jp(j)q(j)} - c_{kp(k)q(k)jp(j)q(j)} \right\}$$

$$\Delta \mathrel{+}= \sum_{\substack{j=0 \\ j\neq k,l \\ j\neq r,s}}^{n-1} \left\{ c_{lp(k)q(l)jp(j)q(j)} - c_{lp(l)q(l)jp(j)q(j)} \right\}$$

$$\Delta \mathrel{+}= \sum_{\substack{j=0 \\ j\neq k,l \\ j\neq r,s}}^{n-1} \left\{ c_{rp(r)q(s)jp(j)q(j)} - c_{rp(r)q(r)jp(j)q(j)} \right\}$$

$$\Delta \mathrel{+}= \sum_{\substack{j=0 \\ j\neq k,l \\ j\neq r,s}}^{n-1} \left\{ c_{sp(s)q(m)jp(j)q(j)} - c_{sp(s)q(s)jp(j)q(j)} \right\}$$

Our proposed Δ computation method also considers the three other cases where ($k = r$ and $l \neq s$), ($k \neq r$ and l = s) and ($k = r$ and $l = s$). It can be proved that the incremental function of our proposed advanced neighborhood requires the computation of $16 \times (n-1)$ cost terms c_{ijklmn} in the worst case.

Most of LS algorithms use neighborhoods which are in general a linear or quadratic function of the input instance size. Some large neighborhoods may be high-order polynomial of the size of the input instance. For instance, the size of our proposed neighborhood is equal to $(\frac{n\times(n-1)}{2})^2$ i.e. this latter is a quartic function of the input size. Then, the complexity of the search will be much higher.

So, in practice, such large neighborhoods for LS algorithms are unusable because of their high computational cost. In the other sections, we will show how the use of GPU computing allows to fully exploit parallelism in such algorithms.

IV. GPU Computing for Metaheuristics

Driven by the demand for high-definition 3D graphics on personal computers, GPUs have evolved into a highly parallel, multithreaded and many-core environment. Indeed, this architecture provides tremendous computational horsepower and very high memory bandwidth compared to traditional CPUs. Since more transistors are devoted to data processing rather than data caching and flow control, GPU is specialized for compute-intensive and highly parallel computation. A complete review of GPU architecture can be found in [7].

Recently, their use has been extended to other application domains [10] (e.g. computational science) thanks to the publication of the CUDA (Compute Unified Device Architecture) development toolkit that allows GPU programming in C-like language. In some areas such as numerical computing [11], we are now witnessing the proliferation of software libraries such as CUBLAS for GPU. However, in other areas such as combinatorial optimization, in particular metaheuristics, the arrival of GPU does not know the same growth. Indeed, there only exists few research works related to evolutionary algorithms on GPU: genetic algorithm [12], [13], genetic programming [14], [15] and evolutionary programming [16], [17]. To the best of our knowledge GPU computing has never deeply investigated for LS algorithms [18], [19]. With the arrival of OpenCL as the open standard programming language on GPU and the arrival of future compilers for this language, like other application areas, combinatorial optimization on GPU will generate a growing interest.

Nevertheless, the use of GPU-based parallel computing for metaheuristics is not straightforward. Indeed, several challenges mainly related to the hierarchical memory management have to be considered. The major issues are the efficient distribution of data processing between CPU and GPU, the thread synchronization, the optimization of data transfer between the different memories, the capacity constraints of these memories, etc. Such issues have been dealt with in one of our previous work [20] for the re-design of parallel LS models to allow solving of large scale optimization problems on GPU architectures.

Basically, in general-purpose computing on graphics process units, the CPU is considered as a host and the GPU is exposed as a device coprocessor. This way, each GPU has its own memory and processing elements that are separate from the host computer, where data must be transferred between the memory space of the host and device. Each device processor supports the single program multiple data (SPMD) model, i.e. multiple autonomous processors simultaneously execute the same program on different data.

For achieving this, the notion of kernel is defined. It is a function callable from the host and executed on the specified device simultaneously by several processors in parallel. Figure 1 illustrates an example of this concept. Memory transfer from the CPU to the device memory is a synchronous operation which is time consuming. Bus bandwidth and latency between CPU and GPU can significantly decrease performance of a program. As a consequence, data transfers between the GPU and the host memory must be minimized to avoid significant bottleneck.

The adaptation of metaheuristics on GPU requires to take into account at the same time the characteristics and underlined issues of the GPU architecture and the parallel models of metaheuritics. The main challenge which persists is the efficient distribution of the search process among the CPU and the GPU minimizing the data transfer between them. Therefore, in designing metaheuristics on GPU, one has to identify what must be performed on CPU and GPU.

V. Design and Implementation of a Tabu Search on GPU

To allow solving Q3AP instances, a tabu search algorithm using our proposed neighborhood has been considered. Basically, the tabu search enhances the performance of a local search method by using memory structures. Indeed, the main memory structure called the tabu list represents the history of the search trajectory. In this way, using this list allows to avoid cycles during the search process. More details of this algorithm are given in [21].

A. The Proposed GPU-based Algorithm

As quoted above, a GPU is organized following the SPMD model, meaning that multiple autonomous processors simultaneously execute the same program at independent points. Adapting traditional LS methods to GPU is not a straightforward task because hierarchical memory management on GPU has to be handled. As previously said, memory transfers from CPU to GPU are slow and these copying operations have to be minimized.

We propose a tabu search on GPU (see algorithm 1) for the Q3AP in agreement with the previous general GPU model presented in Section IV (Fig. 1). This algorithm can be seen as a cooperative model between the CPU and the GPU. Indeed, the GPU is used as a coprocessor in a synchronous manner. The resource-consuming part i.e. the generation and evaluation kernel is calculated by the GPU and the rest is handled by the CPU.

First of all, at initialization stage, memory allocations on GPU are made: the input matrices and the candidate solution of the Q3AP must be allocated (lines 4 and 5). Since GPUs require massive computations with predictable memory accesses, a structure has to be allocated for storing all the

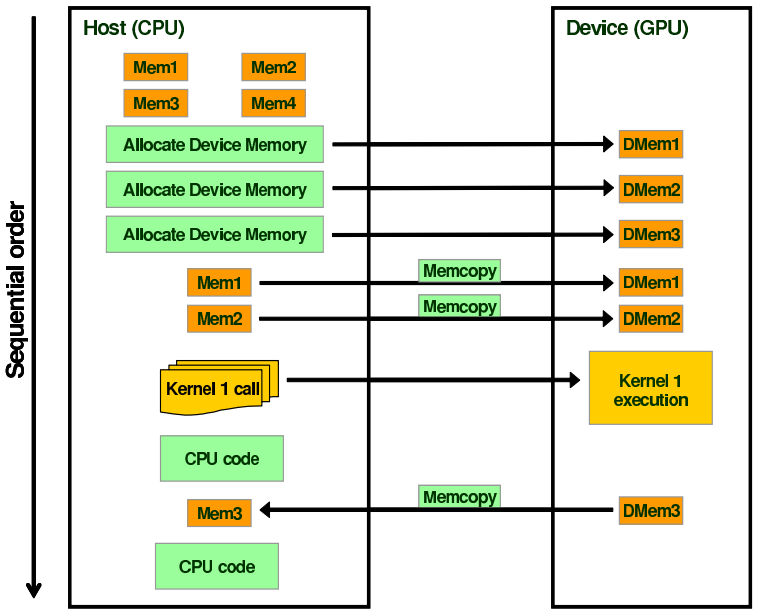

Fig. 1. Illustration of the general GPU model.

Algorithm 1 Tabu Search Template on GPU

1: Choose an initial solution
2: Evaluate the solution
3: Initialize the tabu list
4: Allocate the two Q3AP matrices on GPU device memory
5: Allocate a solution on GPU device memory
6: Allocate a neighborhood fitnesses structure on GPU device memory
7: Copy the two Q3AP matrices on GPU device memory
8: Copy the solution on GPU device memory
9: **repeat**
10: **for** each generated neighbor on GPU **do**
11: Incremental evaluation of the candidate solution
12: Insert the resulting fitness into the neighborhood fitnesses structure
13: **end for**
14: Copy the neighborhood fitnesses structure on CPU host memory
15: Select the best admissible neighboring solution
16: Update the tabu list
17: Copy the chosen solution on GPU device memory
18: **until** a maximum number of iterations reached

neighborhood fitnesses at different addresses (line 6). Second, the matrices and the initial candidate solution have to be copied on the GPU (lines 7 and 8). It is important to notice that the input matrices are a read-only structure and never change during all the execution of LS algorithms. Therefore, their associated memory is copied only once during all the execution. Third, comes the parallel iteration-level, in which each neighboring solution is generated, evaluated and copied into the neighborhood fitnesses structure (from lines 10 to 13). Fourth, since the order in which candidate neighbors are evaluated is undefined, the neighborhood fitnesses structure has to be copied to the host CPU (line 14). Then the selection strategy is applied to this structure (line 15): the exploration of the neighborhood fitnesses structure is done by the CPU. Finally, after a new candidate has been selected, this latter is copied to the GPU (line 17). The process is repeated until a given number of iterations has been reached.

VI. Experimental Results

A. Effectiveness of the Proposed Neighborhood

Before implementing any metaheuristic on GPU, we need first to evaluate the impact of our proposed neighborhood in terms of effectiveness. The following experiment intends to compare a simple tabu search algorithm with the two neighborhoods mentioned above for the Q3AP instances. On the one hand, for the first basic neighborhood addressed in the literature, a neighbor is obtained by swapping two elements in either the first or the second permutation. On the other hand, for our advanced neighborhood, generating a neighbor consists in exchanging two elements in both two permutations. The instance Nug15 has been considered and 50 executions have been performed for each algorithm. The number of iterations is set to 11025 for the advanced neighborhood and 2315250 for the basic one. This way, a fair comparison is made in accordance with the number of evaluated solutions (evaluations). The size of the tabu list is set to $\frac{m}{4}$ where m is the size of each neighborhood. Fig. 2 reports the average evolution of the fitnesses for the instance Nug15 on a Core 2 Duo 2Ghz with an approximate running time of 3 minutes.

From the beginning of the search process, the tabu search using the advanced neighborhood starts to find more improving solutions. As the number of iterations grows, our proposed

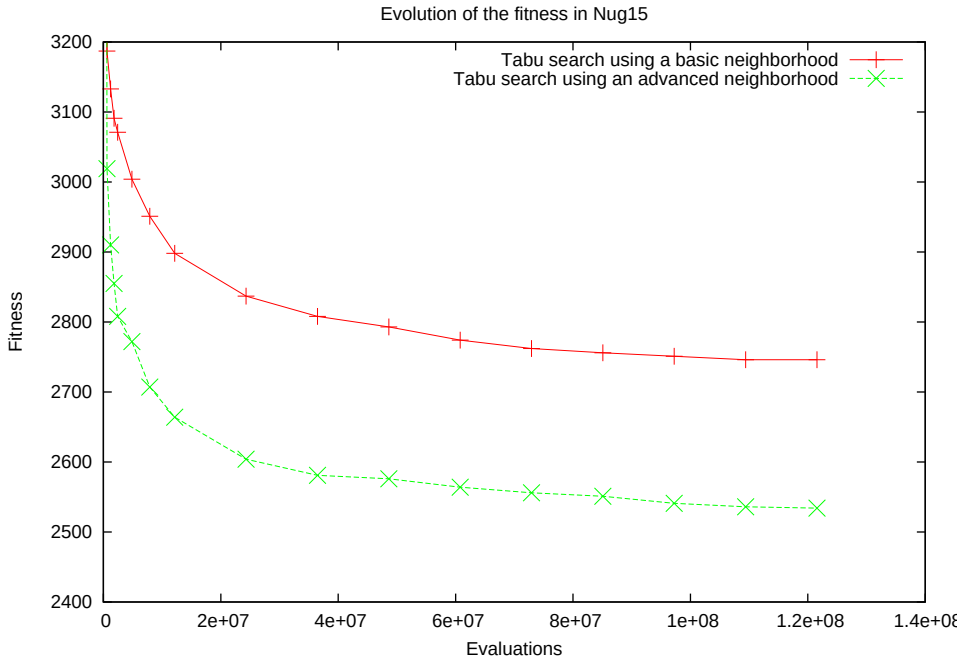

Fig. 2. Comparison of the average evolution of the fitnesses for two different neighborhoods.

neighborhood clearly outperforms the basic one. Indeed, since more solutions are evaluated at each iteration, designing operators to explore large promising regions of the search space of the Q3AP allows to improve the quality of the obtained solutions. Similar results not reported here can be obtained for the other Nugent instances (Nug8, Nug12, Nug13, Nug18 and Nug22).

B. Effectiveness in Comparison with the Literature

The following experiments intend to measure the effectiveness of our proposed neighborhood through LS algorithms. However, although the increase of the neighborhood size allows to improve the effectiveness for the Q3AP, using such neighborhood is generally CPU time-consuming. Therefore, the use of GPU computing provides an efficient complementary way to speed up the search.

An iterated local search using an embedded tabu search (ILS-TS) has been implemented on GPU. The iterated local search (ILS) may be used to improve the quality of successive local optima provided by TS methods. The principle of the ILS method consists in perturbing the local optima and reconsidering them as initial solutions [22]. Regarding our ILS-TS algorithm, the applied perturbation is a random number μ of swaps in either the first or the second permutation where $\mu \in [2 : n]$ (n is the instance size). From an implementation point of view, since the ILS process consists in a loop over the TS (thus performed on CPU), adapting this algorithm according to the proposed TS template on GPU (see Algorithm 1) is straightforward.

The used configuration for experiments is a Core 2 Duo 2GHz laptop with a NVIDIA GeForce 8600M GT where the number of multiprocessors is equal to 4. This graphic card has been chosen among others since it represents nowadays a standard in most of computers.

The number of ILS iterations and the number of TS iterations are respectively equal to 100 to 5000. These values have been set in accordance with those chosen in [4], [9] to perform less evaluations in terms of computational time.

The tabu list size is set to $\frac{m}{4}$ as before. The average time measurement for 50 executions is reported in seconds and acceleration factors compared to a standalone CPU are also considered. The algorithm is stopped when a maximum number of iterations has been reached or when the optimal/best known value has been discovered. Average and max values of the evaluation function have been measured. The number of successful tries (hits) and the average number of ILS iterations to converge to the optimal/best known value are also represented. The associated standard deviation for each average measurement are shown in sub-index. Since the computational time is too exorbitant for Nug18 and Nug22, the average expected time for the CPU implementation is deduced from the base of one ILS iteration per execution. Table I reports the obtained results for the ILS-TS using our neighborhood structure.

In comparison with the literature [4], [9], the obtained results by the ILS-TS with our proposed neighborhood are really competitive. Indeed, considering a smaller number of maximal evaluations (thus less computational time), this algorithm is able to find the optimal/best known value with a better significant rate success (varying from 62% to 100%) for most Nugent instances.

Regarding the execution time, the fact to generate and eval-

TABLE I
ILS TABU SEARCH FOR DIFFERENT Q3AP INSTANCES

Instance	Optimal/Best known value	Average value	Max value	Hits	CPU time	GPU time	Acceleration	ILS iteration
Nug12	**580**	$580.5_{3.4}$	604	98%	256_{246}	113_{120}	$\times 2.3$	18_{18}
Nug13	**1912**	$1917.6_{15.1}$	1974	74%	1879_{1568}	476_{309}	$\times 3.9$	57_{36}
Nug15	2230	2230	2230	100%	1360_{1216}	283_{301}	$\times 4.8$	15_{15}
Nug18	17836	$17874.4_{52.8}$	18026	62%	17447_{11523}	3130_{2117}	$\times 5.6$	59_{38}
Nug22	42476	42476	42476	100%	16147_{14239}	2647_{2341}	$\times 6.1$	15_{12}

uate the neighborhood in parallel on GPU provides an efficient way to speed-up the search process in comparison with a single CPU. Indeed, for the smallest instance Nug12, the GPU version starts to be faster than the CPU one (acceleration factor of $\times 2.2$). As long as the problem size increases, the speed-up grows significantly (up to $\times 6.1$ for the Nug22 instance).

The conclusion from this experiment indicates that the use of GPU provides an efficient way to deal with large neighborhoods. Indeed, our proposed neighborhood is unpractical in terms of single CPU computational resources for large Q3AP instances such as Nug18 or Nug22 (estimated to around 5 hours per run). So, implementing this algorithm on GPU has allowed to exploit parallelism in such neighborhood and improve the robustness/quality of provided solutions.

VII. DISCUSSION AND CONCLUSION

Local search algorithms based on large neighborhoods may allow to enhance the effectiveness and robustness in combinatorial optimization [6]. However, their exploitation for solving real-world problems is possible only by using a great computational power. High-performance computing based on GPU accelerators is recently revealed as an efficient way to use the huge amount of resources at disposal and fully exploit the parallelism of neighborhoods.

In this paper, we have particularly focused on the design and the achievement of a new neighborhood for the Q3AP. However, LS algorithms using such neighborhood is unpractical on traditional machines because of their high computational cost. Therefore, the use of GPU-based parallel computing is required as a complementary way to speed up the search.

GPU computing has thus permitted to design and implement an iterated tabu search and the obtained results on the Q3AP are particularly promising in terms of effectiveness. The experiments indicate that GPU computing allows not only to speed up the search process, but also to exploit large neighborhoods structures to improve the robustness and the quality of the obtained solutions.

Regarding the execution time, the reported speedups on a traditional GeForce 8600M GT (4 multiprocessors) provide significant results (up to $\times 6$) compared to traditional CPUs. It is obvious that the efficiency of the iterated tabu search in terms of acceleration would be drastically enhanced by choosing sophisticated cards such a GeForce 8800 GTX (16 multiprocessors) or GTX 280 (30 multiprocessors).

Beyond the improvement of the efficiency and the effectiveness, the parallelism of GPUs allows to push far the limits in terms of computational resources. As a consequence, a next perspective is to use a multi-GPU approach to tackle larger instances. It will consist of partitioning the neighborhood set, where each partition is executed on a single GPU. That way, a multi-GPU approach will allow to increase the speed-up of the exploration space of a given solution. But since each GPU has its own private memory, managing the context execution of different GPUs in an efficient way is not a straightforward task.

Furthermore, the GPU-based re-design of LS metaheuristics will be integrated in the ParadisEO platform [23]. This framework was developed for the reusable and flexible design of parallel hybrid metaheuristics dedicated to the mono and multiobjective optimization. ParadisEO is based on a clear conceptual separation of metaheuristics concepts, and can be seen as a white-box object-oriented with reusable components. The Parallel Evolving Objects (PEO) module of ParadisEO includes the well-known parallel and distributed models for metaheuristics such as LS methods. This module will be extended with multi-core and GPU-based implementations.

REFERENCES

[1] W. P. Pierskalla, "The multi-dimensional assignment problem," September 1967, technical Memorandum No. 93, Operations Research Department, CASE Institute of Technology.

[2] L. K. Rasmussen and B. W. Wicker, "Trellis-Coded, Type-I Hybrid-ARQ Protocols Based on CRC Error-Detecting Codes," *IEEE Trans. Commun.*, vol. COM-43, pp. 2569–2575, Oct. 1995.

[3] P. M. Hahn, B.-J. Kim, T. Stützle, S. Kanthak, W. L. Hightower, Z. D. H. Samra, and M. Guignard, "The quadratic three-dimensional assignment problem: Exact and approximate solution methods," *European Journal of Operational Research*, vol. 184, pp. 416–428, 2008.

[4] P. M. Hahn, B.-J. Kim, T. Stützle, S. Kanthak, W. L. Hightower, H. Samra, Z. Ding, and M. Guignard, "The quadratic three-dimensional assignment problem: Exact and approximate solution methods," *European Journal of Operational Research*, vol. 184, no. 2, pp. 416–428, 2008.

[5] E.-G. Talbi, *Metaheuristics: From design to implementation*. Wiley, 2009.

[6] R. K. Ahuja, J. Goodstein, A. Mukherjee, J. B. Orlin, and D. Sharma, "A very large-scale neighborhood search algorithm for the combined through-fleet-assignment model," *INFORMS Journal on Computing*, vol. 19, no. 3, pp. 416–428, 2007.

[7] S. Ryoo, C. I. Rodrigues, S. S. Stone, J. A. Stratton, S.-Z. Ueng, S. S. Baghsorkhi, and W. mei W. Hwu, "Program optimization carving for gpu computing," *J. Parallel Distribributed Computing*, vol. 68, no. 10, pp. 1389–1401, 2008.

[8] B.-J. Kim, "Investigation of methods for solving new classes of quadratic assignment problems (*QAPs*)," Ph.D. dissertation, University of Pennsylvania, 2006.

[9] L. Loukil, M. Mehdi, N. Melab, E.-G. Talbi, and P. Bouvry, "A parallel hybrid genetic algorithm-simulated annealing for solving q3ap on computational grid," in *IPDPS*. IEEE, 2009, pp. 1–8.

[10] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron, "A performance study of general-purpose applications on graphics processors using cuda," *J. Parallel Distributed Computing*, vol. 68, no. 10, pp. 1370–1380, 2008.

[11] C. Tenllado, J. Setoain, M. Prieto, L. Piuel, and F. Tirado, "Parallel implementation of the 2d discrete wavelet transform on graphics processing units: Filter bank versus lifting," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 3, pp. 299–310, 2008.

[12] J.-M. Li, X.-J. Wang, R.-S. He, and Z.-X. Chi, "An efficient fine-grained parallel genetic algorithm based on gpu-accelerated," in *Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference*, 2007, pp. 855–862. [Online]. Available: http://dx.doi.org/10.1109/NPC.2007.108

[13] O. Maitre, L. A. Baumes, N. Lachiche, A. Corma, and P. Collet, "Coarse grain parallelization of evolutionary algorithms on gpgpu cards with easea," in *GECCO*, F. Rothlauf, Ed. ACM, 2009, pp. 1403–1410.

[14] D. M. Chitty, "A data parallel approach to genetic programming using programmable graphics hardware," in *GECCO*, 2007, pp. 1566–1573.

[15] W. Banzhaf and S. Harding, "Accelerating evolutionary computation with graphics processing units," in *GECCO (Companion)*, F. Rothlauf, Ed. ACM, 2009, pp. 3237–3286.

[16] T.-T. Wong and M. L. Wong, "Parallel evolutionary algorithms on consumer-level graphics processing unit," in *Parallel Evolutionary Computations*, 2006, pp. 133–155.

[17] K.-L. Fok, T.-T. Wong, and M. L. Wong, "Evolutionary computing on consumer graphics hardware," *IEEE Intelligent Systems*, vol. 22, no. 2, pp. 69–78, 2007.

[18] A. Janiak, W. A. Janiak, and M. Lichtenstein, "Tabu search on gpu," *J. UCS*, vol. 14, no. 14, pp. 2416–2426, 2008.

[19] W. Zhu, J. Curry, and A. Marquez, "Simd tabu search for the quadratic assignment problem with graphics hardware acceleration," *International Journal of Production Research*, 2008.

[20] T. V. Luong, N. Melab, and E.-G. Talbi, "Parallel Local Search on GPU," INRIA, Research Report RR-6915, 2009. [Online]. Available: http://hal.inria.fr/inria-00380624/en/

[21] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers and Operations Research*, vol. 13, no. 5, pp. 533–549, 1986.

[22] T. Stützle, "Iterated local search for the quadratic assignment problem," *European Journal of Operational Research*, vol. 174, no. 3, pp. 1519–1539, 2006.

[23] S. Cahon, N. Melab, and E.-G. Talbi, "ParadisEO: a Framework for the Reusable Design of Parallel and Distributed Metaheuristics," *Journal of Heuristics*, vol. 10, pp. 353–376, 2004, kluwer Academic Publishers.