World Academy of Science, Engineering and Technology
International Journal of Electronics and Communication Engineering
Vol:11, No:3, 2017

Parallel 2-Opt Local Search on GPU

Wen-Bao Qiao, Jean-Charles Créput

Abstract—To accelerate the solution for large scale traveling salesman problems (TSP), a parallel 2-opt local search algorithm with simple implementation based on Graphics Processing Unit (GPU) is presented and tested in this paper. The parallel scheme is based on technique of data decomposition by dynamically assigning multiple K processors on the integral tour to treat K edges' 2-opt local optimization simultaneously on independent sub-tours, where K can be user-defined or have a function relationship with input size N. We implement this algorithm with doubly linked list on GPU. The implementation only requires $O(N)$ memory. We compare this parallel 2-opt local optimization against sequential exhaustive 2-opt search along integral tour on TSP instances from TSPLIB with more than 10000 cities.

Keywords—Doubly linked list, parallel 2-opt, tour division, GPU.

I. Introduction

HEURISTICS optimization algorithms like 2-opt or 3-opt have been widely proved useful to optimize permutation problems, like Traveling Salesman Problem (TSP) or Vehicle Routing Problem (VRP), and have been widely used for various applications.

When applying classical sequential 2-opt local search optimization algorithm with the first evaluation strategy (first optimization, first accept) for large scale TSP instances possessing N edges ($N > 1000$), only one edge is being locally optimized at one time. While many other edges do not participate in current edge's local optimization if they are not in the range of current edge's local optimization. One straightforward solution for this problem is to make multiple 2-opt local search happened simultaneously and independently for different edges' local optimization. Comparing different parallel techniques base on various computing platforms, we present a parallel 2-opt local search by using technique of data decomposition working on Graphics Processing Unit (GPU). The reasons are following: Firstly, parallel granularity of data decomposition is determined by the volume of data and can be very large [1]; secondly, GPU provides efficient multi-threading read/write operation on shared memory.

Any parallel implementation of 2-opt local search needs to consider one nature attribute of 2-opt for permutation problems that have ordering, like TSP tour order. This attribute is shown in Fig. 1: After any one execution of 2-exchange, tour order of all cities between the related two 2-opt edges has been totally inverted. Instead of making parallel 2-opt algorithm work on 1-dimension buffer memory where the ordering of cities in memory represents current TSP tour at any time [2], as shown in Fig. 2 (a), we propose to make this parallel 2-opt local search algorithm work on doubly linked list for large-scale TSP. Comparison of these two representation methods are shown in Fig. 2, parallel 2-opt local search with doubly linked list only takes $O(N)$ memory.

W. B. Qiao and J. C. Créput are with Le2i FRE2005, CNRS, Arts et Métiers, Univ. Bourgogne Franche-Comté, France (e-mail: http://www.multiagent.fr/People:Qiao_wenbao, http://www.multiagent.fr/People:Creput_jean-charles).

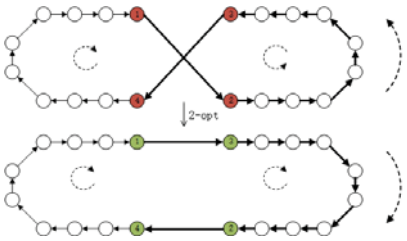


Fig. 1 After each execution of 2-exchange, tour order of all cities between the related two 2-opt edges has been totally inverted

The paper is organized as follows: Section II presents related work on parallel 2-opt; Section III reviews advantages of using doubly linked list as representation for TSP; Section IV presents our strategy of parallel 2-opt local search algorithm working on that doubly linked list; Section V includes experiments and comparison.

II. Related Work

Even though researchers have studied possible parallel strategies for heuristic 2-opt optimization algorithms since more than two decades, the principles to explain "parallel computing" should be classified. Johnson [3], [4] discussed parallel schemes like "geometric partitioning and tour-based partitioning" and Verhoeven et al. [5] distinguished parallel algorithms between data and function parallelism [5] in which he proposed a tour repartitioning scheme that guarantees their algorithm will not halt until it has found a minimum for the complete problem [5]. Luong [6] and Rocki [7] adopt parallel strategies similar to "function parallelism" which means one sequential step is executed in parallel, as Rocki [7] tries to distribute the calculation for one edge's 2-opt optimization between threads, but only the first edge' optimization has finished, the second edge begins its optimization.

Here, according to the problems presented in Section I, we propose to adopt a parallelism much like "data parallelism" as Verhoeven [5] distinguished, but not exactly the same to dynamically separate the entire tour into disconnected parts and check 2-opt in these parts simultaneously. Besides, our parallelism of 2-opt local search optimization algorithm works on doubly linked list, which economizes memory occupancy on parallel devices as well as allows two directional local search.

In our method, to make sure there is no interaction between massive parallel 2-exchanges, which may cut the integral tour into disconnected parts, our parallel implementation of 2-opt

local search exactly follows the concept of local search along one same tour direction. In this strategy, massive K edges check their independent 2-opt optimization simultaneously along the same TSP tour direction, but each edge only checks its optimization in its closest N/K neighboring edges along the tour. After these K edges have been optimized locally in one tour direction, the algorithm begins next K edges' optimization separately on next independent N/K neighboring edges until all edges have a chance to be locally optimized. To enlarge the possibility of accessing one edge's local optimization position, we also propose to check each edge's optimization among its another N/K neighboring edges in opposite tour direction. In this way, the algorithm reduces complexity of original problems and there exist no chance to divide the tour into disconnected parts or one edge's 2-opt optimization would influence another.

III. Doubly Linked List as Representation of TSP

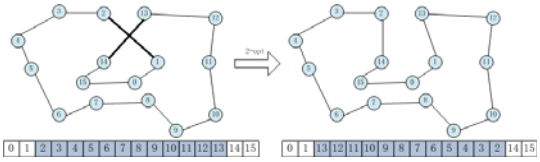
Most TSP tours are represented by using 1-dimension buffer memory or unidirectional list. A drawback of using 1-dimension memory is shown in Fig. 2 (a), all cities' memory position marked by blue color need to be inverted after each 2-opt to begin next 2-opt local search. While the drawback of using unidirectional list is that the neighborhood local search for one edge only goes in one direction from this edge.

Here, we propose to use doubly linked list to represent permutation problems, with which 2-exchange can be easily executed just by changing the related four cities' links and the 2-opt local search can go easily in two directions from this edge. As shown in Fig. 2 (b), every node (city) is connected with two and only two neighbor nodes, while these neighbor cities are not necessary to be adjacency on memory of a hardware. Doubly linked means that if node A connects node B, node B should necessarily connect node A. For TSP applications, every node has one father-link and son-link, they are used to construct TSP tour solution and execute 2-exchanges only by operations on links. Here, which neighbor node acts as father-link or son-link is not unchangeable because it is the staring node and tour direction that decide which neighbor node acts as father or son. As shown in Fig. 2 (b), starting from city V_0 as a father node and V_1 as its second visiting city, all nodes marked in blue circles act as son-links in current TSP tour.

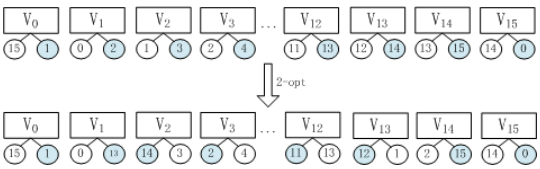
IV. Parallel 2-opt within Tour Division

When applying sequential 2-opt local search optimization algorithm with the "first evaluated, first accept" strategy for large scale TSP instances, many other edges do not have chances to participate in current edge's local optimization.

Here, with the parallel technique of data decomposition, we test a simple parallel scheme to check K active edges' possible 2-exchanges on separate N/K sub-tours along the same tour direction, which also follows the idea of local search. And we execute the firstly accepted 2-exchanges for these K active edges simultaneously on GPU side. The overall parallel scheme is shown in Fig. 3 and the detailed Kernel function for optimization is presented in Algorithm 1. Before



(a) Permutation order represented by using 1-dimension buffer memory



(b) Permutation order represented by using doubly linked list

Fig. 2 Comparison of memory operation after one same 2-exchange using different representation methods for TSP tour order (a): Tour order represented by using 1D buffer memory. The algorithm needs extra temporary memory to invert all cities' position between these two related 2-opt edges because their tour order has been totally inverted. (b): Tour order represented by doubly linked list. The algorithm just needs to change links of the related four cities and can go easily in two opposite directions from current edge

starting the algorithm, initial TSP tour solution has already been presented by using doubly linked list.

Figs. 3 and 4 explain well about this simple parallel 2-opt local search strategy. As shown in Fig. 3, for a TSP instance with N size, the algorithm starts with a random active node p_i acting as the first father-node in the tour. To make sure that multiple 2-opt optimization happens at correct direction and do not create independent sub-tours, the algorithm needs to mark every node's tour $ID : i_{p_i}$, which is named as step of "refresh TSP tour order" in Fig.3 and done easily by using a simple operation that each city finds its next connected but unvisited link taking advantage of the doubly linked list. For example, start from node $p_{i,(i=0,1,...,N-1)}$ with tour $ID : i_{p_i} = 0$, then, choose one of the two links of p_i as the second visiting city with tour $ID : i_{p_1} = 1$ and assign every remaining node a specific tour $ID : i_p = 2, 3, 4, ...(N-1)$ increasing one after another. For opposite direction, starting from the same node p_i but marking its another link as its son-link, then the integral tour is inverted by just finding every unvisited son-link. After this step, the new tour information is copied to GPU global memory.

Before beginning multiple K edges' 2-opt optimization on GPU side, we should activate these K edges. This is easy by using CUDA programming [8], we begin with a random non-activated node p_{a0} as the first active node and mark other nodes whose $i_p = i_{p_{a0}} + k * (N/K), k = 1, 2, ..., K$ as the rest active nodes. With every node's tour ID being known already, K edges are activated to search and execute 2-opt simultaneously on device side exactly as what the kernel function shows in Alg. 1.

This parallel 2-opt local search optimization method works in a way shown in Fig. 4, each active node only checks its first 2-opt optimization along the same tour direction until it encounters next active city. In the opposite tour direction, every edge gets a chance to be optimized among its another

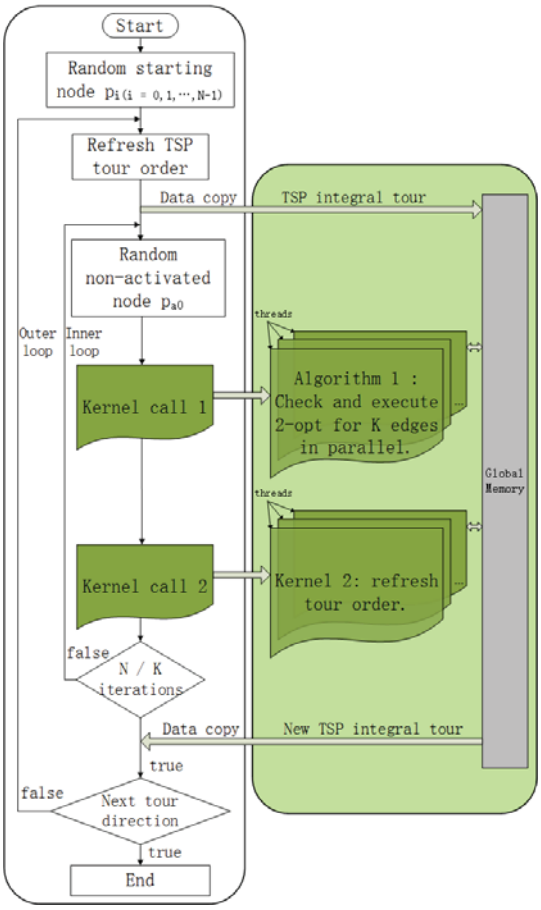


Fig. 3 Overall algorithm flow of our parallel 2-opt local search algorithm with dynamic tour division: One inner loop optimize K edges simultaneously; every edge has been optimized once after finishing the inner loops, namely one outer loop; edges have chances to be optimized twice along its two neighboring directions after two outer loops, which makes up *one run* in our test. We search and execute 2-exchanges without cutting the integral TSP tour in the kernel function

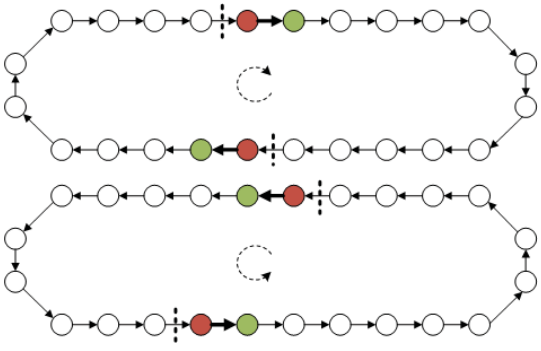


Fig. 4 Parallel 2-opt local search with dynamic tour division: Simultaneously check 2-opt local optimization for different edges (bold arrows) along the tour. This check stops once the first 2-opt optimization is found or it reaches to the edge marked by dash line before the next active red node. Then, invert tour order and check these edges' local optimization in another tour direction

local N/K neighboring edges.

To reduce the time taken for copying data from GPU to CPU side, we propose to refresh tour order directly on GPU side, shown as "Kernel call 2" in Fig. 3. As we optimize these K edges in the same tour direction without cutting the tour, the original tour order is influenced only inside each independent N/K sub-tours. So we refresh the tour order of each sub-tour beginning with the tour ID of each p_{ak} on GPU side. After this step, the algorithm begins with next K edges' optimization in next inner loop until all nodes have one chance to be optimized in one outer loop.

After the inner loop, the algorithm inverts the tour direction from the same starting node p_i, and provides a chance that edges can be locally optimized among their another neighboring N/K edges in the second outer loop.

Algorithm 1 Kernel function: Parallel 2-opt local optimization on doubly linked list with dynamic tour division along the same tour direction. Input TSP instance (N size) is oriented with every node having its unique tour order $i_p = 0, 1, 2...(N-1)$ increasing one by one according to the tour direction. For each active node $p_{a_k,k=(0,1,...k)}$, each thread runs the following same code

Require: Current active node p_{a_k}, marked as the first node p_1;
Choose one of the two links from p_1 to be p_2 according to current tour direction;
Mark exchange link positions for p_1 and p_2 separately;
Choose one of the two links from node p_2 to be p_t, make sure $p_t \neq p_1$;
if $p_t = p_{a_{k+1}}$ **then**
  return
else
  $p_3 \leftarrow p_t$;
  while $p_3 \neq p_{a_{k+1}}$ **do**
    Choose one of the two links from node p_3 to be p_4 according to current tour direction;
    Mark exchange link positions for p_3 and p_4 separately;
    if $dis(p_1,p_3) + dis(p_2,p_4)$ less than $dis(p_1,p_2) + dis(p_3,p_4)$ **then**
      Execute 2-exchange;
      break;
    else
      $p_3 \leftarrow p_4$
    end if
  end while
end if

The complexity of this parallel 2-opt local optimization method depends on the length of local search for each edge. For a TSP instance with size N and the number of active threads is K, for example $K = 10$, there are totally K edges are optimized at the same time. The length of local search for every node equals to N/K. So the total complexity in one run of this algorithm, which makes sure all edges can be optimized once, reduces with a division factor of k^2 while the step of refreshing TSP order on GPU side should also be considered, because it needs maximum $O(N/K)$ complexity.

V. Experiment

The parallel 2-opt local search optimization methods presented in this paper work well with user-defined local

search range (namely the length of N/K). We test this parallel strategy based on GPU with CUDA programming. However, different local search ranges for each edge produce various optimization result for the same initial TSP solution.

In our tests for each TSP instance, we try to find the best performance of this parallel 2-opt local search algorithm. So the length of N/K is randomly tested for different TSP instances at first. we begin with the initial TSP solution provided by original TSP files from TSPLIB. Visual result of one test using this parallel 2-opt local search algorithm with dynamic tour division strategy is shown in Fig. 5, from initial TSP solution with total distance 261879 in Fig. 5 (a), each inner loop of the algorithm in Fig. 3 optimizes two or three edges simultaneously on GPU side with local search range $N/K = 322$; after eight runs, the TSP solution for lu980.tsp reduces to 12460 and can not be optimized further using this method, as shown in Fig. 5 (i). This process is called *one test* in our experiments, which is automatically finished. Average values of ten tests using this algorithm to optimize the same initial TSP solutions are shown in Table 1, in which each test makes sure the tour can not be further optimized. In Table 1, "distance" indicates the length of final TSP solution; "time(s)" is the average total time taken in one test, including necessary time for generating random non-activated starting node, time for refreshing TSP tour order and time for copying data from GPU to CPU; "%PDM" is the percentage deviation between the mean solution and the optimum solution; "%PDB" is the percentage deviation between the best solution and the optimum solution; "Runs" indicates the average quantity of runs in one test to get the final TSP solution; "N/K" is the experimental local search range for each edge. Besides that, we also build sequential 2-opt exhaustive search algorithm working on doubly linked list as comparison. For sequential exhaustive 2-opt optimization, we also begin with random starting node p_{a0}, make sure every edge is optimized once in one run and the TSP solution can not be further optimized using this 2-opt strategy in one test. The results are shown in TABLE 2, where "2-opt FIRST" indicates that we adopt "first optimized first accept" strategy for sequential exhaustive 2-opt optimization algorithm along the tour, and "2-opt BEST" indicates that the evaluation strategy is "the best optimized, the first accept". The time taken in each test also includes time for generating random non-optimized edges and time for refreshing TSP tour after each 2-opt optimization. These tests are executed on laptop with CPU: Inter(R) Core(TM) i7-4710HQ 2.5GHz, GPU: GeForce GTX 850M.

Compared with Tables 1 and 2, we can conclude that with appropriate local search range (N/K) for each edge, our parallel 2-opt local search algorithm with tour division has the ability to produce similar (see %PDM) or even better (see %PDB) results compared with sequential exhaustive 2-opt optimization along the integral tour for each edge. Further more, the total running time is decreased by using this parallelism of 2-opt local search algorithm even though we also count the time for copying data from GPU to CPU. Three factors affect the final TSP solution when using this parallel scheme. The first one is the choice of K that can be user-defined or varied with the input size. The choice of K should ensure the length of local search (N/K) that should not be too short for each edge's local optimization. The second one is the random choice of staring nodes in Fig. 3. And the last factor is the initial status of TSP instance, here we use the given sequence of cities provided in original tsp files.

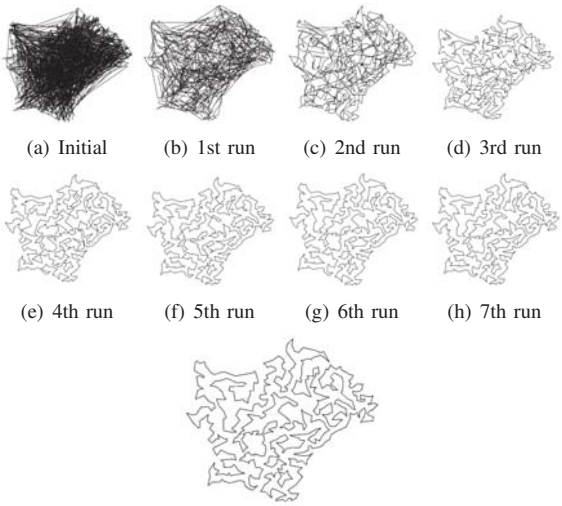
TABLE I
STATISTICS OF PARALLEL 2-OPT LOCAL SEARCH WITH DYNAMIC TOUR DIVISION WORKING ON GPU

instance	distance	time(s)	% PDM	% PDB	Runs	N/K
lu980	12724	3.12	12.20	10.03	7.4	437
rw1621	29407.87	14.01	12.89	10.77	8.3	837
mu1979	97603	14.05	12.33	10.68	8.6	837
nu3496	109624.2	28.08	14.03	12.24	9.3	973
tz6117	455373.8	75.86	15.36	13.21	8.7	2000
eg7146	196279.8	121.46	13.85	12.49	9.6	2347
fi10639	596419.8	207.76	14.57	13.38	8	2837

TABLE II
STATISTIC OF SEQUENTIAL EXHAUSTIVE 2-OPT

TSP Instances	2-opt FIRST (Sequential)			2-opt BEST (Sequential)		
	t(s)	%PDM	%PDB	t(s)	%PDM	%PDB
lu980	4.35	12.93	11.22	3.81	13.11	11.16
rw1621	14.33	15.06	12.87	13.54	14.52	13.55
mu1979	17.720	11.57	9.61	15.73	13.25	11.56
nu3496	64.19	14.76	13.35	59.02	14.32	12.20
tz6117	176.20	14.99	14.47	170.32	15.53	14.53
eg7146	257.37	13.46	11.11	234.51	13.43	11.53
fi10639	541.73	14.62	13.68	540.88	14.68	14.18

∗ 2-opt FIRST : Sequential 2-opt along the integral tour for each edge with strategy of first optimized first accepted;
∗ 2-opt BEST : Sequential 2-opt along the integral tour for each edge with strategy of best optimized first accepted.



(a) Initial (b) 1st run (c) 2nd run (d) 3rd run

(e) 4th run (f) 5th run (g) 6th run (h) 7th run

(i) 8th run

Fig. 5 TSP solution for lu980.tsp after different runs of Fig.3 in one test: (a) Initial TSP solution according to original TSP files from TSPLIB; (i) After eight runs of Fig. 3, the TSP solution can not be further optimized by using this algorithm and reaches to final distance of 12460 in this test.

VI. CONCLUSION

This paper presents a parallel 2-opt local search algorithm for TSP using doubly linked list to achieve two directional

local optimization while remaining less memory occupancy. The TSP tour is partitioned into multiple sub-tours, each managed by a single processor. The algorithm assigns massive processors along the tour to treat various edges' 2-opt local optimization simultaneously. Experiments show that with appropriate 2-opt local search range for each edge, our algorithm performs better than 2-opt exhaustive search along the integral tour with substantial acceleration factor as the instance's size grows. We think that this straightforward GPU implementation of the parallel 2-opt local search allows for further experiments on very large problems to get increasing acceleration factor as the number of physical cores will grow in GPU systems.

REFERENCES

[1] R. D. Lawrence, G. S. Almasi, and H. E. Rushmeier, "A scalable parallel algorithm for self-organizing maps with applications to sparse data mining problems," *Data Mining and Knowledge Discovery*, vol. 3, no. 2, pp. 171–195, 1999.

[2] S. A. Mulder and D. C. Wunsch, "Million city traveling salesman problem solution by divide and conquer clustering with adaptive resonance neural networks," *Neural Networks*, vol. 16, no. 5, pp. 827–832, 2003.

[3] D. S. Johnson and L. A. McGeoch, "The traveling salesman problem: A case study in local optimization," *Local search in combinatorial optimization*, vol. 1, pp. 215–310, 1997.

[4] ——, "Experimental analysis of heuristics for the stsp," in *The traveling salesman problem and its variations*. Springer, 2007, pp. 369–443.

[5] M. Verhoeven, E. H. Aarts, and P. Swinkels, "A parallel 2-opt algorithm for the traveling salesman problem," *Future Generation Computer Systems*, vol. 11, no. 2, pp. 175–182, 1995.

[6] T. Van Luong, N. Melab, and E.-G. Talbi, "Gpu computing for parallel local search metaheuristic algorithms," *IEEE Transactions on Computers*, vol. 62, no. 1, pp. 173–185, 2013.

[7] K. Rocki and R. Suda, "Accelerating 2-opt and 3-opt local search using gpu in the travelling salesman problem," in *High Performance Computing and Simulation (HPCS), 2012 International Conference on*. IEEE, 2012, pp. 489–495.

[8] C. CUDA, "Programming guide: Cuda toolkit documentation."