Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in DISCRETE MATHEMATICS

Electronic Notes in Discrete Mathematics 58 (2017) 199–206 www.elsevier.com/locate/endm

A performance study on multi improvement neighborhood search strategy

Eyder Rios [a,1] Luiz Satoru Ochi [a,2] Cristina Boeres [a,3]
Igor M. Coelho [b,4] Vitor N. Coelho [c,5] Nenad Mladenović [d,6]

[a] *Universidade Federal Fluminense (UFF), Niterói, BR*

[b] *Universidade do Estado do Rio de Janeiro (UERJ), Rio de Janeiro, BR*

[c] *Instituto de Pesquisa e Desenvolvimento de Tecnologias, Ouro Preto, BR*

[d] *Mathematical Institut, SANU, Belgrade, Serbia*

Abstract

Among the methods to deal with optimization tasks, parallel metaheuristics have been used in many real-world and scientific applications to efficiently solve these kind of problems. This paper presents a novel Multi Improvement strategy for dealing with the Minimum Latency Problem (MLP), an extension the classic Traveling Salesman Problem. This strategy is embedded in a Graphics Processing Unit (GPU) local search procedure, in order to take advantage of the highly parallel processors from this architecture. In order to explore multiple neighborhoods simultaneously in a CPU/GPU heterogenous and distributed environment, a variant of the classic Variable Neighborhood Descent (VND) is also proposed, named Distributed VND (DVND). The DVND was inspired by a randomized version of the VND (called RVND) and a comparison was made, achieving competitive results in terms of solution quality. The variant of the DVND using two processes succeeded in achieving superlinear speedups up to 2.85, demonstrating that the DVND scalability and capability to explore both GPUs and CPUs. Finally, experiments demonstrate that the Multi Improvement is capable of finding better quality solutions in shorter computational times, when compared the classic Best Improvement strategy, motivating

http://dx.doi.org/10.1016/j.endm.2017.03.026

future applications in other hard optimization problems.

Keywords: DVND, GPU, neighborhood search, multi improvement, minimum latency problem

1 Introduction

Transportation problems are still some of the most challenging tasks in the Combinatorial Optimization field, although many recent advances in heuristic search techniques and, specially, in metaheuristics (general purpose heuristic frameworks), have been developed. Furthermore, the practical applications of these problems in industry and science are becoming increasingly large and complex, requiring more computational power and specialized methods for a scalable solution. One of such problems is the Minimum Latency Problem (MLP), that has many important applications in business and industry, which consists in finding a tour such that the total accumulated costs (or waiting time) is minimum [2].

In order to solve the MLP and to provide a scalable solution, all computational power available should be explored. In this direction, the Graphics Processing Units (GPUs) have emerged as an alternative high computational power, provided by its massively parallel architecture (manycore) and attractive cost/benefit trade-off for high performance systems [6]. However, integrating GPUs and multicore CPUs is not a trivial task. The CPUs are designed to provide fast response times for a wide range of sequential applications resulting in a more complex device, in which concentrate almost all of the current heuristic techniques. On the other hand, GPUs focus on applications with high data parallelism level, where each processor performs the same task on different pieces of data. As result, GPUs trade single-thread performance by parallel processing power [3], while CPUs should explore task parallelism [7]. Therefore, to fully benefit from CPU/GPU heterogeneous platforms, metaheuristics should properly explore the parallel resources available in those systems seeking to achieve a good trade-off on both data and task parallelism.

The current best results for the MLP have been achieved by means of

[1] Email: eyder@ic.uff.br
[2] Email: satoru@ic.uff.br
[3] Email: boeres@ic.uff.br
[4] Email: igor.machado@ime.uerj.br
[5] Email: vncoelho@gmail.com
[6] Email: nenad@mi.sanu.ac.rs

metaheuristics which are capable of exploring multiple neighborhood structures [5,8], including a special intensification strategy called the Variable Neighborhood Descent (VND) [4]. This procedure provides a systematic way to explore neighbor solutions by using different move operators of multiple neighborhoods. The idea behind VND is that when a solution is a local optimum regarding one neighborhood structure, it may not necessarily be the optimum to other neighborhoods. So, this multi-neighborhood local search starts evaluating the faster (or smaller) neighborhoods and progress to the slower, as long as no improvement is found in the process. If an improvement is found, the search is restarted from the first neighborhood. However, due to the the high computational complexity of the neighborhood exploration in the VND, this usually limits the number of neighborhoods used in the method. Moreover, due to its naturally sequential behaviour, that strongly depends on previous iterations, it is very difficult to develop an efficient parallelization of the method.

In this paper, we propose a distributed version of the VND, named DVND, that is focused on GPU environments, being capable of achieving high speedups even when a reduced number of neighborhoods is considered. This feat was possible by the development of a novel neighborhood exploration strategy called Multi Improvement (MI), which takes advantage of the massive amount of information processed by the GPU applying many neighbor moves at once. The results of the DVND and the MI are compared with the best search algorithms in literature for the MLP, demonstrating that these techniques are capable of finding competitive solutions in shorter computational times.

The remainder of this paper is organized as follows. Section 2 describes the MLP and the state-of-the-art VNS methods for this problem. The DVND and the Multi Improvement methods are presented in Section 3. In Section 4 these techniques are evaluated in terms of both solution quality and computational time, comparing these to the best methods in literature. Finally, Section 5 concludes the paper and suggests future research directions.

2 Minimum Latency Problem

The MLP is an extension of the classical Traveling Salesman Problem. The MLP can be defined as a directed graph $G = (V, E)$, where $V = \{0, 1, \dots, n\}$ is the set of vertices and $E = \{(i, j) : i, j \in V, i \neq j\}$ a set of arcs that connect the vertices. Each arc (i, j) has an associated traveling time equal to $t(i, j)$. The vertex 0 represents the starting point (depot) and the remaining vertices, the clients to be visited. The latency of a client $i \in V$, denoted by $l(i)$, is defined as the travel time between the depot and the vertex i. The goal of MLP is, starting from depot, determine the Hamiltonian circuit s that minimizes the total latency, expressed by $L(s) = \sum_{i=0}^{n} l(i)$.

Despite its simple formulation, it was proved that MLP is NP-Hard [8]. Although the MLP formulation is quite similar to the TSP, it is known to be more computationally difficult to solved than the TSP. That occurs because small changes in the inputs can lead to global changes in the final solution. Besides, the non-local nature of the objective function makes a simple insertion or change of position of a customer affects the latency of all subsequent clients. The MLP is also referenced by literature as Cumulative Travelling Salesman Problem [1], Deliveryman Problem [5] and Traveling Repairman Problem [10].

3 Methodology

State-of-the-art solutions for the MLP are based on metaheuristic techniques inspired by the Variable Neighborhood Search (VNS) [4]. However, to explore multiple neighborhood structures, two state-of-the-art versions of the VND technique are used. First, the GVNS-1 and GVNS-2 for MLP [5] use the classic VND implementation, considering a deterministic order for the input neighborhoods. Another technique used [8] is called Randomized VND (RVND), where the exploration order of the neighborhoods is changed for each search call. The RVND is the core of many recent efficient search algorithms [8,9] and the randomized nature of this algorithm motivated the development of a RVND version dedicated to distributed computing. The Distributed VND is a parallel local search procedure designed for multicore (CPU) and manycore (GPU) environments. The idea behind DVND is that when a solution is a local optimum regarding one neighborhood structure, it may not necessarily be the optimum to other neighborhoods. If an improvement is found, the search is restarted for all neighborhoods. The method stops when the solution is in a local optimum regarding all neighborhoods.

The parallelization of the RVND is a challenging task because of the natural dependency on the previously processed data after each iteration, since the current best solution is always considered as the initial search point. The DVND solves this issue by including a *history* data structure that stores the current best solution of each neighborhood search separately, together with a global best solution shared between all processes. There is no guarantee that each different search process will be using the best known solution at the same time and, therefore, after each local search, we restart the process with the best known solution in the *history*. The Algorithm 1 shows the DVND pseudocode. The DVND search method was designed to be composed of a set of tasks executed simultaneously, each one associated to a GPU device. The

Algorithm 1 The DVND pseudocode

```
procedure DVND($H, N$)
    $s \leftarrow \arg\min_{s' \in H} \{ f(s') \}$
    foreach $N_k \in N$ do
        $s_k \leftarrow s$
        Launch asynchronously $\texttt{kernel}_k(s_k)$
    end
    $Q \leftarrow \{ \}$
    while $|Q| < |N|$ do
        $k \leftarrow \texttt{wait\_kernel}()$
        $s \leftarrow$Result solution from $\texttt{kernel}_k(s_k)$ launching
        if $f(s) < f(s_k)$ then $H \leftarrow H \cup \{ s \}$ ;
        $s \leftarrow \arg\min_{s' \in H} \{ f(s') \}$
        if $f(s) < f(s_k)$ then
            $s_k \leftarrow s$;
            Launch asynchronously $\texttt{kernel}_k(s_k)$
        else
            $Q \leftarrow Q \cup \{(k, s_k)\}$
        end
        forall $(i, s_i) \in Q$ do
            if $f(s) < f(s_i)$ then
                $s_i \leftarrow s$
                Launch asynchronously $\texttt{kernel}_i(s_i)$
                $Q \leftarrow Q - \{ (i, s_i) \}$
            end
        end
    end
end
```

method starts detecting the number of GPU devices present in the system, and initializing the set N of neighborhood structures to be explored. Then, the neighborhoods set is partitioned aiming to balance the workload among the GPU devices. Next, the history data structure H is initialized with the DVND base solution s, and this global history permits the procedures to exchange information about the best solutions found during the searches, implementing a cooperation mechanism. The algorithm waits for each neighborhood search to finish, then compares the returned result with the best solution from history, relaunching the search processes when an improvement is made. The DVND finishes when the best solution in H is in a local optimum regarding all neighborhood structures.

Each GPU kernel calculates the cost of all moves in a given neighborhood, storing these results in a vector. By using the classic Best Improvement strategy, only the best move is selected from the list, this is why we use a new strategy called Multi Improvement that is capable of applying multiple *in-*

dependent moves simultaneously. We define a move m as independent from other move m', iff for all possible input solutions, the cost of m remains the same if m' is also applied to the same solution. So, after the detection of the resulting independent moves from the GPU, it is possible to apply all of them in a single solution, thus taking advantage of all information processed by the GPU. In this work, we applied these concepts to the neighboorhood structures defined previously in literature for the MLP [8].

4 Computational Results

The experiments were executed in a computer with one Intel® processor i7-4820K 3.70GHz, with 16GB of RAM memory. This machine is equipped with two GPUs NVIDIA® GTX-780, with 12 multiprocessor, each one embedding 192 cores, totaling 2,304 GPU cores. The algorithms were implemented in C++ using the CUDA™ API 7.5 and executed on Linux Ubuntu 14.04.

The first experiments were conducted in order to determine the potencial of the Multi Improvement (MI) compared to the classic Best Improvement (BI). Figure 1 considers instance TRP-S500-R1 (with 500 clients) and demonstrates the quality indicator gtime for each technique (BI and MI) regarding thousands of solutions with different qualities (gaps from 350% down to the best known upper bound). The gtime indicates the average the times in millisec for each technique to reduce a 1% gap. Note that the times increase drastically for both techniques when near optimal solutions are considered and that the MI technique times are always smaller than BI times. This can be explained by the capacity of the MI to generate bigger improvements in the current solution with relatively not much extra computational work, so the average ptime becomes much smaller than in BI. A similar behaviour was observed in all MLP instances available in literature [8].

The DVND was also compared to the RVND for the MLP, considering 30 different initial random solutions for eight problem instances with different sizes. Table 1 compares the RVND [8] with the DVND variants DVND-P1 and DVND-P2, using one and two GPUs, respectively. From this table, the DVND-P1 shows an improvement rate (column I(%)) of 87.13, which is slightly superior than the 87.11 from the RVND. The DVND-P2 achieved very similar results, what indicates that the three methods are equivalent in terms of solution quality, since small variantions are already expected due to the natural randomness of the processes. The number of neighborhood search calls in DVND-P1 and DVND-P2 are vastly superior than in RVND, what indicates that RVND is more efficient in terms of number of iterations. However, due to the use of the MI, the DVND-P1 is able to match RVND in terms of time. Finally, when two processes are considered, the DVND-P2 achieves a remarkable average superlinear speedup of 2.22 (up to 2.85) compared to the DVND-P1, what indicates that the DVND processes exploit both the GPU processes and also the two CPU threads involved in the search.

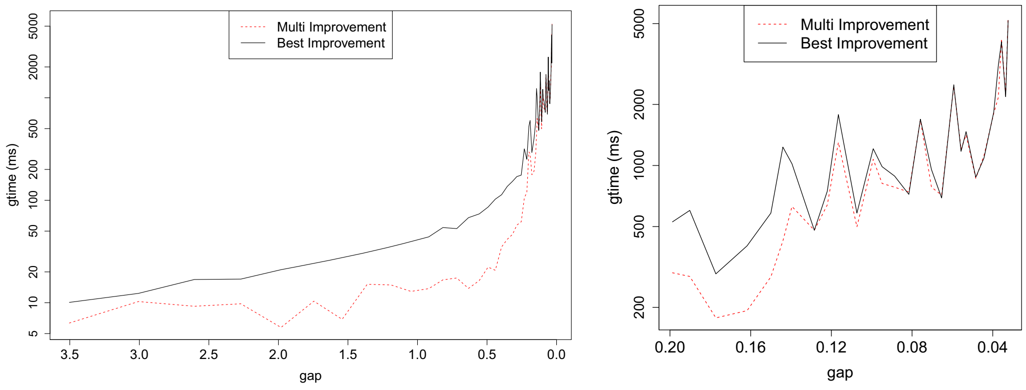

Fig. 1. Comparison between BI and MI strategies instance TRP-S500-R1

Table 1
Comparison between RVND, DVND-P1 and DVND-P2

Instance	RVND			DVND-P1			DVND-P2	
	T(ms)	I(%)	Calls	T(ms)	I(%)	Calls	Speedup	Calls
berlin52	2	68.30	36	2	68.30	90	0.67	90
kroD100	7	79.00	64	9	79.00	173	1.50	168
pr226	55	93.70	110	71	93.70	349	2.54	346
lin318	131	89.60	157	153	89.60	505	2.55	487
TRP-S500-R1	510	90.30	262	553	90.40	758	2.85	732
d657	1145	91.30	301	1211	91.30	940	2.85	901
rat784	1907	91.50	384	2066	91.50	1160	2.64	1129
TRP-S1000-R1	4372	93.20	469	4343	93.20	1417	2.15	1408
Average		**87.11**	**222.9**		**87.13**	**674.0**	**2.22**	**657.6**

5 Conclusions and Future Works

This paper presented the novel Multi Improvement strategy for dealing with a hard combinatorial optimization problem, the Minimum Latency Problem. Also, a distributed variant of the classic VND method was proposed, being able to explore neighbor solutions by means of GPU devices, far superior than CPUs in highly parallel tasks. Results indicate a superlinear speedup of 2.85 for DVND when two processes are considered, finding competitive solutions with a randomized VND technique in shorter computational times. The Multi Improvement appear as a fundamental part of DVND in order to achieve higher solution improvements, taking advantage of all information processed by the GPU. In future works, the DVND can be used inside a metaheuristic framework in order to improve current state-of-the-art MLP solvers, and the Multi Improvement could be applied to other hard optimization problems.

Acknowledgements

The authors thank to the Brazilian funding agencies CAPES, CNPq, FAPEMIG and FAPERJ for supporting this work.

References

[1] Bianco, L., A. Mingozzi and S. Ricciardelli, *The traveling salesman problem with cumulative costs*, Networks **23** (1993), pp. 81–91.

[2] Blum, A., P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan and M. Sudan, *The minimum latency problem*, in: *Proc. of the 26th ACM Symposium on Theory of Computing*, 1994, pp. 163–171.

[3] Lee, V., P. Hammarlund, R. Singhal, P. Dubey, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. Nguyen, N. Satish, M. Smelyanskiy and S. Chennupaty, *Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU*, ACM SIGARCH Computer Architecture News **38** (2010), p. 451–460.

[4] Mladenović, N. and P. Hansen, *Variable neighborhood search*, Computers & Operations Research **24** (1997), pp. 1097–1100.

[5] Mladenović, N., D. Urošević and S. Hanafi, *Variable neighborhood search for the travelling deliveryman problem*, 4OR **11** (2013), pp. 57–73.

[6] Owens, J. D., M. Houston, D. Luebke, S. Green, J. E. Stone and J. C. Phillips, *GPU computing*, Proc. of the IEEE **96** (2008), pp. 879–899.

[7] Schulz, C., *Efficient local search on the GPU - investigations on the vehicle routing problem*, Journal of Parallel and Distributed Computing **73** (2013), pp. 14–31.

[8] Silva, M., A. Subramanian, T. Vidal and L. Ochi, *A simple and effective metaheuristic for the minimum latency problem*, European Journal of Operational Research **221** (2012), pp. 513–520.

[9] Subramanian, A., E. Uchoa and L. Ochi, *A hybrid algorithm for a class of vehicle routing problems*, Computers & Operations Research **40** (2013), pp. 2519–2531.

[10] Tsitsiklis, J. N., *Special cases of traveling salesman and repairman problems with time windows*, Networks **22** (1992), pp. 263–282.