A survey of parallel local search methods application in scheduling and logistics

by

Wojciech Bożejko[1,3], Szymon Jagiełło[1], Mieczysław Wodecki[2,3]

[1] Wrocław University of Technology, Poland,

[2] University of Wrocław, Poland,

[3] College of Management "Edukacja" Wrocław, Poland

Corresponding author:

Wojciech Bożejko

Institute of Computer Engineering, Control and Robotics,

Wrocław University of Technology

Wyb. Wyspiańskiego 27, 50-370 Wrocław, Poland

e-mail: wojciech.bozejko@pwr.wroc.pl

ABSTRACT

The main issue considered in the paper is concerned with solving difficult optimization problems in parallel calculating environments, such as multiprocessor computers, clusters or distributed calculation nodes in networks, by applying algorithms which use various parallelization technologies. Strongly sequential character of the scheduling algorithms is considered to be the main obstacle in designing sufficiently effective parallel algorithms. On the one hand, up till now sequential algorithms exhausted the possibilities of significant growth in the power of solution methods. On the other hand, parallel computations offer essential advantages of solving difficult problems of discrete optimization, pushing towards theory, methodology and engineering of solution algorithms.

Keywords:

Parallel algorithms, discrete optimization, metaheuristics.

INTRODUCTION

The development of optimization methods, particularly applied in production tasks arrangement, has proceeded towards modern and more effective sequence approaches since the beginning of this field. At the end of the 1970s, the turning point in the combinatorial optimization methods was the branch and bound (B&B) method regarded those days as a remedy for nearly all problems of great size which could not be solved by means of methods applied at that time. However, it soon occurred that the B&B method only slightly extended the scope of solvable problems (e.g. for a sum-cost, single-machine scheduling problem this size extended from 20 to 40 - 50 tasks). What is more, the cost necessary to obtain an optimal solution is much too high compared to economic benefits and its use in practice. The conclusion of these investigations was the definition of a bounded area of the B&B scheme application.

The next breakthrough concerned the occurrence of advanced metaheuristic methods: first the simulated annealing method and next the method of genetic algorithms and the tabu search method. Enthusiasm lasted much longer: until around 2005 several dozen of different metaheuristics had been proposed though again those methods reached the limit of their abilities to the moment where the size of effectively solvable problems (i.e., these for which an average deviation from the optimal solutions was smaller than 1\%) might be shifted to a number reaching thousands, but not millions or hundred millions. Eventually the concept of 'no-free-lunch' by Wolpert and Macready [29] finished the discussion. With reference to rough methods this concept may be paraphrased in the following way: without using special attributes of examined problems considerable advantage of one metaheuristic over the other cannot be obtained. What is interesting Wolpert and Macready proved that 'free-lunch' was possible to be obtained in co-evolutional, multi-cultural metaheuristics, i.e., parallel in a natural way. Since the mid-1980s, indeed, parallel many-levelled metaheuristics had been developed, firstly as simple paralleling of the most time-consuming elements of sequence algorithms (usually as the goal function determination), then since the end of the 1990s as multi-track methods.

A marked enhancement of the quality of designed algorithms started when producers of computer equipment realized that further increase of the speed (i.e., the clock frequency) of processors was very costly, while this goal could be more easily obtained applying multi-core constructions, i.e., parallel calculating environments (and in this context among producers of hardware there also exist the term `no-free-lunch'). Today processors of popular producers

such as Intel or AMD have got 4 cores (some Intel processors have 9 cores, and prototypes even 80 cores) and GPU processors (Graphic Processing Unit) at first being used exclusively as graphic processors and nowadays also as strictly computing ones possess even 960 processors (e.g. products of nVidia Tesla series).

SCHEDULING PROBLEMS

Some elementary notions are used in mathematical model building of a job scheduling problem: a *job* and a *resource*. The job consists in executing a sequence of operations which need some resources. A number of data can be connected with the job: *due date* or *deadline*, possibility of breaking the job (*divisibility*), ways of operation execution (specific requirements of resources, alternative ways of execution), etc. Resources can be renewable (processor, machine, memory) or non-renewable (operational materials, natural resources) and dual-bounded (energy, capital). The features of the resources include: accessibility (in time windows), cost, amount, divisibility. All of these features have to be mathematically formalized by constructing a mathematical model of a problem.

Let $J = \{1, 2, ..., n\}$ be a set of jobs which have to be executed by using a set of types of machines $M = \{1,2,\ldots,m\}$. Each job i is a sequence of o_i operations $O_i = (l_{i-1}+1, l_{i-1}+2, \ldots, l_i)$, $l_i = \sum_{k=1}^{i} o_k$, $l_0 = 0$. Operations inside a job have to be executed in a defined technological order (in the defined sequence), i.e., an operation j has to be executed after having finished an operation $j-1$ execution and before starting the execution of an operation $j+1$. A set of operations of a job i will be denoted by O_i for simplicity of notation. For each operation $j \in O$, $O = \bigcup_{i=1}^{n} O_i$ the following terms are determined:

M_j – sequence of m_j subsets of machines which define alternative methods of operation execution; $M_j = (M_{1j}, M_{2j}, \ldots, M_{m_j,j}), M_{ij} \subseteq M$; an operation j needs a set of machines M_{ij} for its execution, where $1 \leq i \leq m_j$,

p_{ij} – time of execution of an operation j by the i-th method (i.e., on the i-th machine),

v_j – method of executing the operation (decision variable),

S_j – term of an operation execution beginning (decision variable),

C_j – term of an operation execution finishing, $C_j = S_j + p_{v_j j}$ if the operation cannot be broken.

In turn, for a job i the following terms are needed to be determined:

o_i – number of operations in the job,

r_i – the earliest possible term of the job execution beginning,

d_i – due date of the job execution finishing,

S_i – term of the job execution beginning, $S_i = S_{l_{i-1}+1}$,

C_i – term of the job execution finishing, $C_i = C_{l_i}$,

L_i – non-timeliness of the job execution finishing, i.e., being tardy or early, $L_i = C_i - d_i$,

T_i – tardiness of the job execution finishing, $T_i = \max\{0, C_i - d_i\}$,

E_i – earliness of the job execution finishing, $E_i = \max\{0, e_i - C_i\}$, $f_i(t)$ – non-decreasing cost function connected with the job i execution finishing in a time $t \geq 0$,

F_i – a time of flow of the job i through the system, $F_i = C_i - r_i$,

U_i – unitary tardiness of the job i.

The majority of scheduling problems do not need to define all the above data and decision variables. Usually a minimal set of notions which is sufficient to describe the model is used. For example, if for each $j \in O$ we have $m_j = 1$, $|M_{1j}| = 1$ it means that the problem has dedicated machines, therefore decision variables v_j do not undergo any choice. Then v does not occur in the model of the problem.

Taxonomy. To describe precisely the scheduling problem a three-field notation $\alpha|\beta|\gamma$ is applied. This notation was proposed in [18] and next developed in [22]. It has three fields $\alpha|\beta|\gamma$ specifying the execution environment α, additional constraints β, and the objective function γ.

Here we propose an extended Graham notation which includes representations of hybrid systems or flexible systems with parallel machines. This kind of scheduling problems cannot be described by the original Graham notation. We propose to set the symbol α as a composition of three symbols $\alpha_3\alpha_2\alpha_1$ which have the following meaning. The symbol α_1 describes a finite number of machines in the system: 1, 2, ...; if this number is not specified then an empty symbol is put here which means any number of machine m. The symbol α_2 describes the method of jobs flowing through the system, where the following traditional ways are enhanced:

F – flow shop in which all the jobs have the same technological path and they all have to be executed on all the machines; each machine needs to determine different sequence of input jobs,

F^* – permutation flow shop, a model which has the same assumptions as F with an additional requirement that a sequence of job execution on all the machines has to be the same (compatible with the order of the sequence of jobs input into the system),

J – job shop, in which jobs can have different (in terms of the number and the order of visiting machines) technological paths,

G – general shop, in which each job is a single operation and technological relationship is given by a graph,

O – open shop, in which all the operations of jobs have to be executed, but the technological order of operations inside the job is not specified.

The number of machines $\alpha_1 = 1$ implicates that both α_3 and α_1 symbols have to be empty. The symbol α_3 determines the mode of executing each operation. If α_3 is an empty symbol then we assume that for each operation a machine has been dedicated on which it will be executed, that is $m_j = 1,\ |M_{ij}| = 1,\ j \in O$. Otherwise, we assume that $m_j \geq 1,\ |M_{ij}| = 1,$ $i = 1,2,\ldots,m_j,\ j \in O$ and an operation can be executed on exactly one machine from a set of:

P – identical parallel machines,

Q – uniform machines, or

R – non-uniform machines.

As we have already mentioned both α_3 and α_1 symbols can be empty, which means that any realization mode can be accepted, or (α_1 empty symbol) any (but fixed) number of machines can be used.

The symbol β determines the existence of additional assumptions and constraints, e.g. different release times (the earliest possible times of beginning job execution, r_i), existence of a partial technological order of job execution (*prec*), constraints *no wait*, *no store*, *no idle* (without time gaps), $p_{ij} = 1$ (all times are identical and equal 1), *pmtn* (jobs can be stopped and started again), etc.

The last parameter γ has the symbolic form of the criteria function. Two classes of this function occur in theory and practice of job scheduling, namely

$$f_{\max} = \max_{1 \le i \le n} f_i(C_i)$$

and

$$\sum f_i = \sum_{i=1}^{n} f_i(C_i),$$

where $f_i(t)$ are some non-decreasing functions. These classes include, among others, many frequent criteria from the practice, for example: the length of the schedule (makespan)

$$C_{\max} = \max_{1 \le i \le n} C_i,$$

an average time of the job flow

$$\sum F_i = \frac{1}{n} \sum_{i=1}^{n} F_i.$$

In the second case we may include a different weight of jobs $w_i \ge 0$ in the cost function $f_i(t) = w_i t$. For jobs with due dates d_i one can construct measures $f_i(t) = \max\{0, t - d_i\}$ or $f_i(t) = w_i \max\{0, t - d_i\}$. Therefore, we obtain

$$T_{\max} = \max_{1 \le i \le n} T_i = \max_{1 \le i \le n} \max\{0, C_i - d_i)$$

or the weighted sum of job tardiness

$$\sum w_i T_i = \sum_{i=1}^{n} w_i T_i = \sum_{i=1}^{n} w_i \max\{0, C_i - d_i\}.$$

The most frequent considered (in the literature) are the following criteria: makespan ($C_{\max}$), sum of job finishing times (C_{sum}, $\sum C_i$) and weighted sum of job tardiness ($\sum w_i T_i$). The criteria cited above are known as typical in practice and they generate troubles during optimization (they are difficult).

LOGISTICS PROBLEMS

The Vehicle Routing Problem (VRP) belongs to the NP-hard problems class. Solving it provides a key for increasing efficiency in transportation management. It was first described in 1959 by G. B. Danzig and J. H. Ramser [14]. Since that time VRP literature was enriched with thousands of new positions containing a variety of different VRP types. VRP solving tools provide great savings (about 5-30% [17]) for society every day. Exact methods are not capable of solving instances of VRP problems that contain more than 50-100 customers in reasonable time so there is still much need for research, particularly for large scale rich instances [17]. This is a very important limitation, when taking into account that the number of customers can reach several thousand for bigger companies.

The problem described by Danzig and Ramser was Capacity constrained VRP (CVRP). It is one of the simplest types of VRP problems. In CVRP there is:

- a constant number of homogeneous vehicles with a specified capacity,
- a constant number of customers,
- a demand with specified size (capacity needed) and a localization is assigned to each customer,
- the vehicles are able to pick up the demands from the customers but the sum of demand sizes that have been handled by a vehicle cannot exceed its capacity,
- the vehicles are located in a single depot which is the start and end point of their tour,
- travel costs between localizations (customers and depot) are given,
- each customer has to be served exactly once.

The sum of travel costs between localizations of a tour is the cost of the tour and the sum of tour costs is the cost of the solution. The goal of CVRP optimization is to find a set of K (the number of vehicles) tours with minimal total travel cost and preserving problem assumptions.

As stated before CVRP is one of the simplest VRP problems but there are many other simple types often quoted in literature. One of them is the *Multi Depot VRP* (MDVRP) which is a

generalization of the *Single Depot VRP* (SDVRP) [10]. The most significant characteristic of MDVRP is that the number of depots can take values greater than one. Therefore the vehicles can start from multiple depots but still have to return to the start depot at the end of the travel.

In the *Periodic VRP* (PVRP) a time horizon is added [16]. Thanks to that the PVRP can be used to provide solutions for situations where delivery routes are constructed for a period of time. A real life case is for example collecting trash from customers three times a week.

The *Distance constrained VRP* (DVRP) is very similar to the CVRP [HaKl, Ka]. The difference lies in the constrained dimension. In DVRP the travel length or duration for each vehicle cannot exceed a given maximum value. Another basic model the *Distance and Capacity constrained VRP* (DCVRP) which utilizes the limitations present in both models can be constructed by combining them.

The *Vehicle Routing Problem with Time Windows* (VRPTW) [2] should be applied for modeling services like bank or postal deliveries where the time of the delivery has to be taken into consideration. The routing plan has to be constructed in such a way that each customer demand is served within a given time interval. Depending on the VRPTW subtype the time windows can be either soft (the window can be violated at a cost) or hard (vehicles are not allowed to arrive at a customer after the given interval).

The type of VRP that needs to be used strongly depends on the real life case. When analyzing real life cases there is almost always a need to use more complex (rich) VRP types e.g. *Fleet Size and Mix VRP with Multiple Time Windows* (FSMVRPMTW) [17]. The example model includes optimal composition of a fleet of heterogeneous vehicles that can satisfy customer demands in alternative time windows.

VRP problems are generalizations of the Traveling Salesman Problem (TSP) which is one of the most famous optimization problems. In its simplest form it includes [5]:

- a constant number of cities,
- one traveling salesman,
- travel costs between cities,
- the Salesman has to begin and end his tour in a specified start city and visit the other cities exactly once.

The sum of travel costs between cities is the cost of a tour. The goal of TSP optimization is to find a tour with minimal travel cost and preserving problem assumptions. In order to create an instance of TSP by using CVRP the following assumptions have to be made:

- the start city is the depot,
- number of vehicles is set to 1 (number of traveling salesman),
- the number of customers is set to the number of cities minus one,
- the capacity of the salesman is unlimited.

PARALLEL ARCHITECTURES

In recent years, several theoretical models of parallel computing systems were proposed. Up till now some of them have been physically realized. These theoretical models take into account only the ways of manipulating instructions (instruction set) and the type of data streams. We extend this taxonomy by adding memory architectures.

The fundamental classification of parallel architectures was given by Flynn [15]. Here we present it based on a survey taken from [1].

- **SISD machines.** *Single Instruction stream, Single Data stream.* Classic serial machines belong to this class. They contain one CPU and hence can accommodate one instruction stream that is executed serially. Many large mainframes can have more than one CPU but each of them execute instruction streams that are unrelated. Therefore, such systems still should be regarded as multiple SISD machines acting on different data spaces. Examples of SISD machines are mainly workstations like those of DEC, Hewlett-Packard, IBM and Silicon Graphics.
- **SIMD machines.** *Single Instruction stream, Multiple Data stream.* These systems often possess a large number of processing units, ranging from 100 to 100,000 all of which can execute the same instruction on different data. Thus, a single instruction manipulates many data items in parallel. Examples of SIMD machines are the CPP DAP Gamma II and the Quadrics Apemille. Other subclasses of the SIMD systems embrace the vector processors which manipulate on arrays of similar data rather than on single data items using CPUs with special instructions (e.g. MMX, SSE2). If data can be manipulated by these vector units the results can be delivered at a rate of one, two and three per clock cycle. That is why vector processors work on their data in a parallel way but this only refers to the vector mode. In this case they are several times faster than when executing in conventional scalar mode. An extension of the vector processing idea is GPGPU.

- **MISD machines.** *Multiple Instruction stream, Single Data stream.* This category includes only a few machines, none of them being commercially successful or having any impact on computational science. One type of system that fits the description of an MISD computer is a systolic array which is a network of small computing elements connected in a regular grid. All the elements are controlled by a global clock. In each cycle, an element will read a piece of data from one of its neighbors, perform a simple operation and prepare a value to be written to a neighbor in the next step.
- **MIMD machines.** *Multiple Instruction stream, Multiple Data stream.* MIMD machines execute several instruction streams in parallel on different data. Compared to the multi-processor SISD machines mentioned above the difference lies in the fact that the instructions and data are related because they represent different parts of the same task to be executed. Therefore, MIMD systems can run many subtasks in parallel in order to shorten the time-to-solution for the main task to be executed. There is a large variety of MIMD systems and especially in this class the Flynn taxonomy proves to be not fully adequate for the classification of systems. If we focus on the number of system processors this class becomes very wide, from a NEC SX-9/B system with 4-512 CPUs or clusters of workstations to a thousand processors IBM Blue Gene/P supercomputer and Cray XT5-HE (224162 cores) which breaks the petaflops barrier.

Memory architectures. The Flynn taxonomy does not recognize memory architecture. In our opinion memory architecture types have an influence on parallel algorithm efficiency. Therefore, we propose to select two classes here.

- *Shared memory systems.* They have multiple CPUs all of which share the same address space (shared memory). It means that the knowledge of where data is stored is of no concern to the user as there is only one memory accessed by all CPUs on equal basis. Shared memory systems can be both SIMD and MIMD. Single-CPU vector processors can be regarded as an example of the former, while the multi-CPU models of these machines are examples of the latter. The abbreviations SM-SIMD and SM-MIMD are usually used for the two subclasses.
- *Distributed memory systems.* Each CPU possesses its own associated memory in this class. The CPUs are connected by a network and they may exchange data between their respective memories if necessary. Unlike with the shared memory machines the

user has to be aware of the data location in the local memories, besides they will have to move or distribute these data explicitly if necessary. The distributed memory systems may be either SIMD or MIMD.

Although the difference between shared- and distributed-memory machines seems to be clear, this is not always entirely the case from the user's point of view. Virtual shared memory can be simulated at the programming level. For example, a specification of High Performance Fortran (HPF) was published in 1993 [19] which, by means of compiler directives, distributes the data over the available processors. That is why the system on which HPF is implemented in this case will look like a shared memory machine to the user. Other vendors of Massively Parallel Processing systems (sometimes called MPP systems), like HP and SGI, are also able to support proprietary virtual shared-memory programming models due to the fact that these physically distributed memory systems are able to address the whole collective address space. Therefore, for a user such systems have one global address space spanning all of the memory in the system.

The other important issue from the user's point of view is the access time to each memory address of the shared memory. If this access time is constant, we say that the system is of UMA (uniform memory access) type, if it is not we call it NUMA (non-uniform memory access). Additionally, there is a distinction if the caches are kept coherent (coherent cache or CC-NUMA) or not (non-coherent cache or NC-NUMA).

For SM-MIMD systems we can mention OpenMP [11] that can be applied to parallelize Fortran and C++ programs by inserting comment directives (Fortran 77/90/95) or pragmas (C/C++) into the code. Also many packages to realize distributed computing are available. Their examples are PVM (Parallel Virtual Machine, [16]), and MPI (Message Passing Interface, [25]). This programming style, called the 'message passing' model has become so accepted that PVM and MPI have been adopted by nearly all major vendors of distributed-memory MIMD systems and even on shared-memory MIMD systems for compatibility reasons. In addition, there is a tendency to cluster shared-memory systems, for instance by HiPPI channels, to obtain systems with a very high computational power. E.g., the NEC SX-8, and the Cray X1 have this structure. Thus within the clustered nodes a shared-memory programming style can be applied, whereas between clusters a message-passing should be used. Nowadays, PVM is not applied a lot any longer and MPI has become the standard.

Distributed systems are usually composed of a set of workstations (so-called cluster) connected by a communication network such as Infiniband, Myrinet or Fast Ethernet. Such a *cluster of workstations* (COW) has better price-to-performance ratio, and it is more scalable and flexible compared to multiprocessor systems. On the other hand, MPP (*massively parallel processor*) systems are composed of thousands of processors, which can belong to multiple organizations and administrative domains, creating so-called *grids*, built on the basis of the Internet infrastructure.

Recent trends. For the last few years GPGPU parallel programming model has been used for massive shared-memory applications. GPUs are regarded as SIMD processors (or MIMD when the processors can handle multiple copies of the same code executing with different program fragments, e.g. counters, see Robilliard et al. [24]). In the CUDA programming environment, developed by nVidia, the GPU is viewed as a computing device capable of running a very high number of threads in parallel, operating as a coprocessor of the main CPU. Both the host (CPU) and the device (GPU) maintain their own DRAM, referred to as the host memory and device memory, respectively. One can copy data from one DRAM to the other through optimized API calls that utilize the device's Direct Memory Access (DMA) engines.

The GPU is especially well-suited to address problems that can be expressed as data-parallel computations – SIMD – with high arithmetic intensity (the number of arithmetic operations is significantly greater than the number of memory operations). Because the same program is executed on many data elements and has high arithmetic intensity, the memory access latency can be hidden with calculations instead of big data caches. In practice GPU programming is very close to the PRAM machine model from the programmers' point of view, offering a simple tool for checking the theoretical PRAMs algorithm efficiency (see Bożejko et al. [7]).

PARALLEL METAHEURISTICS

Metaheuristics based on the local search method can be presented as processes of a graph searching in which vertices are the points of the solution space (e.g. permutations) and arcs correspond to the neighborhood relation – they connect vertices which are neighbors in the solution space. We will call it *neighborhood graph*. For all NP-hard problems the related neighborhood has an exponential size. Moving on such a graph defines some *path* (in other words, *trajectory*) in the solution space. Parallel metaheuristic algorithms make use of many processes for parallel generation or search of the neighborhood graph.

One can define two approaches to parallelization of the local search process in relation to the number of trajectories which are concurrently generated in the neighborhood graph:

1. *single-walk parallelization* (single trajectory): fine-grained algorithms for fast communication purposes (the most computationally expensive parts of the algorithm are parallelized),
2. *multiple-walk parallelization* (many trajectories): coarse-grained algorithms, communication is less frequent, compared to the single-walk parallelized algorithms.

These approaches demand that the algorithm meet some requirements as regards communication and synchronization frequency, which implies the kind of granularity. Single-walk parallel metaheuristics are usually fine-grained algorithms (e.g. Bożejko, Pempera and Smutnicki, [8]), multiple-walk metaheuristics – coarse-grained (e.g. Bożejko, Pempera and Smutnicki, [6]).

Single-walk parallel algorithms. Single walk algorithms go along the single trajectory, but they can use multithread calculations for the neighborhood decomposition (see *representatives* method, [23]) or parallel cost function computation. For example, calculations of the cost function value for more complicated cases are frequently equivalent to determining the longest (critical) path in a graph, as well as maximal or minimal flow.

Multiple-walk parallel algorithms. Algorithms which make use of a multithread multiple-walk model search concurrently a solution space by searching threads working in parallel. Additionally, these algorithms can be divided into subclasses due to communication among threads (information about current search status): (1) *independent* search processes and (2) *cooperative* search processes. If the multithread application (i.e., concurrently running search processes) does not exchange any information we can talk about *independent* processes of search. However, if information accumulated during an exploration of the trajectory is sent to another searching process and used by it, then we can talk about *cooperative* processes (see Bożejko et al. [6]). We can also come across a mixed model, so-called *semi-independent* (see Czech [13]) executing independent search processes keeping a part of common data.

Implementation. Due to the specificity of the metaheuristic type, as well as parallel environment architecture (SIMD, MIMD, shared memory, etc.) different programming languages are used for coding. As we can see in Table 2 SIMD algorithms for GPU are implemented in C++ with CUDA programming library – nowadays it is the most commonly

used programming environment for nVidia GPUs. SIMD algorithms for multiprocessor computers without shared memory are implemented in Ada95 high-level programming language, due to the simplicity of designing them. Algorithms for distributed MIMD clusters are implemented in C++ programming language with the use of MPI (*Message Passing Interface*) communication library, also the most commonly used tool for programming clusters.

PARALLEL LOCAL SEARCH METHODS

Let us consider a *discrete optimization problem* formulated as follows. Let X be a discrete solution space and let $F : \mathsf{X} \leftarrow \mathsf{R}^+$ be a non-negative function defined on the solution space X. We are looking for the optimal element $x^* \in \mathsf{X}$ such that

$$F(x^*) = \min_{x \in \mathsf{X}} F(x).$$

A major class of discrete optimization problems solving algorithms (apart from population-based methods) is a *local search* approach, in which an algorithm creates a searching trajectory which passes through the solution space X. Before its parallelization, let us formally describe this class of methods.

The well-known local optimization procedure begins with an initial solution x^0. In each iteration for the current solution x^i the neighborhood $\mathsf{N}(x^i)$ is determined. Next, from the neighborhood the best element $x^{i+1} \in \mathsf{N}(x^i)$ is chosen (i.e., with the best cost function value $F(x^{i+1})$) constituting the current solution in the next iteration. The method is exhaustive. An outline of the local search method is presented in Figure 3. The method generates a solutions sequence $x^0, x^1, x^2, \ldots, x^s$ such that $x^{i+1} \in \mathsf{N}(x^i)$. We called this sequence a *trajectory*. The problem (2.1) can be replaced by

$$F(x^A) = \min_{x \in \mathsf{Y}} F(x),$$

where

$$\mathsf{Y} = \{x^0, x^1, x^2, \ldots, x^s\} \subseteq \mathsf{X}.$$

We call the mechanism of a neighbor generation a *move*. More precisely, the move μ is a function $\mu : \mathsf{X} \to \mathsf{X}$ which generates solutions $\mu(x^i) = x^{i+1} \in \mathsf{N}(x^i) \subseteq \mathsf{X}$ from a solution $x^i \in \mathsf{X}$.

A crucial ingredient of the local search algorithm is the definition of the neighborhood function in combination with the solution representation. It is obvious that the choice of a good neighborhood is one of the key factors ensuring efficiency of the neighborhood search method. A neighborhood $N(x)$ is defined as a subset $N(x) \subset \mathsf{X}$ of solutions 'close to' a solution $x \in \mathsf{X}$. A metric of the 'nearness' can be a distance metric in this solution space (e.g. Hamming's or Caley's), or the number of moves.

Parallel local search strategies. Generally, several approaches to convert LSM to parallel LSM (p-LMS) can be formulated:

1. calculating $F(x)$ faster for a given $x \in \mathsf{X}$,
2. making a choice of $x^{i+1} \in N(x^i)$ faster,
3. making a space decomposition among p searching threads, i.e.,

$$F(x^A) = \min_{1 \le k \le p} F(x^{Ak})$$

where

$$F(x^{Ak}) = \min_{x \in \mathsf{Y}^k} F(x), \mathsf{Y}^k = \{x^{0k}, x^{1k}, \ldots, x^{sk}\}.$$

4. using cooperative trajectories.

Alba [1] proposed the following classification:

- *Parallel multi-start model.* In this model several local search processes are executed concurrently, each one starting from the different solution. Either homogeneous or heterogeneous version of this model can be applied. They can be based on the same searching strategy, or have different strategies. Multiple working searching processes can also start from the same starting point, but with different searching strategies (e.g. with different parameters). Simple classification of such algorithms on the tabu search metaheuristic example was proposed by Voss in [28]. This model belongs to the multiple-walk parallelization class.
- *Parallel moves model.* This is a low-level parallelization model which consists in neighborhoods concurrent searching. The main metaheuristic which uses this kind of parallelism, computes the same results as the sequential version but faster. Each

processor evaluates a part of neighborhood preparing the best element (so-called representative) as the proposition for the controlling processor which chooses the best solution from all representatives. This model is usually implemented as a master-slave model of parallelization, yet it can be developed both as the single-walk method and the multiple-walk parallelization (i.e., inside a hybrid method as a low level parallelism).

- *Move acceleration model*. The goal function value is calculated in a parallel way in this model. Such a parallelization is problem-oriented and strongly dependent on the goal function form. For example, it is difficult or even impossible to parallelize the function which has a recurrent form. Usually loops, minimum or sum calculations, are parallelized in this model. Because of the input-output intensity that kind of parallelism needs a shared-memory fine-grained parallel environments such as multi-processor mainframe computers or GPUs. Similarly to the previous (parallel moves) model it can be developed both as the single-walk method and as the multiple-walk parallelization.

Most survey works consider only parallel multi-start model of parallel local search metaheuristics, see [Alba, Ba, Bo, CrGe, Kn, Ta]. This is due to the difficulty of designing parallel moves and move acceleration models which are strongly dependent on the optimization problem formulation (see Bożejko [9] and Steinhöfel et al. [26]). This parallelization also needs to take advantage of the special properties of the optimization approach, i.e., neighborhood determination method, cost function calculation and methods of calculations distribution among processors.

Here we propose an *extension* of Alba taxonomy of parallel local search methods by including (at least) the following additional model:

- *Parallel tree-based model*. In this model, local search processes are concurrently executed; each one starting from the solution found by another process, i.e., as soon as its best solution is found. The most frequent approaches are: the blackboard broadcasting method using shared memory, and the master-slave model in which the master process is controlling the whole searching process and local search threads are executed on slave processors.

CONCLUSIONS

We present a survey of modern multithreaded (parallel and distributed) approaches of solving hard problems of discrete optimization, especially in the field of scheduling and logistics. Parallel machines architecture and programming environment was presented. We propose a new extended taxonomy of parallel local search strategies used by parallel and distribute metaheuristics approximate algorithms.

REFERENCES

[1] Alba E., Parallel Metaheuristics. A New Class of Algorithms, Wiley & Sons Inc. (2005).

[2] Shahrzad Amini, Hassan Javanshir, Reza Tavakkoli-Moghaddam, A PSO approach for solving VRPTW with real case study, International Journal of Research and Reviews in Applied Sciences, Vol: 4 Issue: 3, (2010).

[3] Badeau P., Guertin F., Gendreau M., Potvin J.Y., Taillard E., A parallel tabu search heuristic for the vehicle routing problem with time windows, Transportation Research-C 5 (1997), 109–122.

[4] Bożejko W., A new class of parallel scheduling algorithms, Wrocław University of Technology Publishing House, (2010), 1–280.

[5] Bożejko W., Wodecki M., Parallel Evolutionary Algorithm for the Traveling Salesman Problem, Journal of Numerical Analysis, Industrial and Applied Mathematics Vol. 2, No. 3-4, (2007), 129–137.

[6] Bożejko W., Pempera J., Smutnicki A., Multi-thread parallel metaheuristics for the flow shop problem, in: L. Zadeh, L. Rutkowski, R. Tadeusiewicz, J. Żurada (Eds.), International Conference on Artificial Intelligence and Soft Computing (ICAISC 2008), IEEE Computational Intelligence Society – Poland Chapter and the Polish Neural Network Society (2008), 454–462.

[7] Bożejko W., Smutnicki C., Uchroński M., Parallel calculating of the goal function in metaheuristics using GPU, in: G. Allen et al. (Eds.), ICCS 2009, Part I, Lecture Notes in Computer Science No. 5544 (2009), 1022–1031.

[8] Bożejko W., Pempera J., Smutnicki C., Parallel single-thread strategies in scheduling, in: L. Rutkowski, R. Tadeusiewicz, L.A. Zadeh, J.M. Żurada (Eds.), Artificial Intelligence and Soft Computing – ICAISC 2008, Lecture Notes in Artificial Intelligence No. 5097, Springer (2008), 995–1006.

[9] Bożejko W., Parallel path relinking method for the single machine total weighted tardiness problem with sequence-dependent setups, Journal of Intelligent Manufacturing, Vol. 21 Issue 6, Springer (2010), 777–785.

[10] John Carlsson, Dongdong Ge, Arjun Subramaniam, Amy Wu and Yinyu Ye, Solving Min-Max Multi-Depot Vehicle Routing Problem, Solving min-max multi-depot vehicle routing problem, Fields Institute Communications, Lectures on Global Optimization, American Mathematical Soc., (2009), 31–46.

[11] Chandra R., Dagum L., Kohr D., Maydan D., McDonald J., Menon R., Parallel Programming in OpenMP, Morgan Kaufmann Publishers Inc. (2001).

[12] Crainic T.G., Gendreau M., Cooperative Parallel Tabu Search for Capacited Network Design, Journal of Heuristics 8 (2002), 601–627.

[13] Czech Z., Three parallel algorithms for simulated annealing, Lecture Notes in Computer Science No. 2328, Springer Verlag (2002), 210–217.

[14] G. B. Dantzig and J. H. Ramser, The Truck Dispatching Problem, INFORMS, Management Science, Vol. 6, No. 1 (Oct., 1959), 80–91.

[15] Flynn M.J., Very high-speed computing systems, Proceedings of the IEEE 54 (1966), 1901–1909.

[16] Peter Francis, Karen Smilowitz and Michal Tzur, The period vehicle routing problem with service choice, To appear in Transportation Science, October 11, (2005).

[16] Geist A., Beguelin A., Dongarra J., Manchek R., Jaing W., Sunderam V., PVM: A Users' Guide and Tutorial for Networked Parallel Computing, MIT Press, Boston (1994).

[17] Geir Hasle and Oddvar Kloster, Industrial Vehicle Routing, SINTEF ICT, Department of Applied Mathematics, P.O. Box 124 Blindern, NO-0314 Oslo, Norway.

[18] Graham M.R., Lawler E.L., Lenstra J.K., Rinnoy Kan A.H.G., Optimization an approximation in detrministic sequencing and scheduling: a survey, Annals of Discrete Mathematics 3 (1979), 287–326.

[19] High Performance Fortran Forum, High Performance Fortran language specification, Scientific Programming 2, 13 (1993), 1–170.

[20] Imdat Kara, Two Indexed Polynomial Size Formulations for Vehicle Routing, Baskent University, Department of Industrial Eng., Baglica, Ankara, Turkiye, Working paper, June 12, (2008).

[21] Knox J., Tabu search performance on the symmetric traveling salesman problem, Computers & Operations Research 21 (1994), 867–876.

[22] Lenstra J.K., Sequencing by Enumeration Methods, Mathematical Centre Tract 69, Mathematisch Centrum, Amsterdam (1977).

[23] Nowicki E., Smutnicki C., A fast tabu search algorithm for the job shop problem, Management Science 42 (1996), 797–813.

[24] Robilliard D., Marion-Poty V., Fonlupt C., Population parallel GP on the G80 GPU, in: O'Neil M. et al. (Eds.), Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008, Vol. 4971, Springer (2008).

[25] Snir M., Otto S., Huss-Lederman S., Walker D., Dongarra J., MPI: The Complete Reference Vol. 1, The MPI Core, MIT Press, Boston (1998).

[26] Steinhöfel K., Albrecht A., Wong C.K., Fast parallel heuristics for the job shop scheduling problem, Computers and Operations Research 29 (2002), 151–169.

[27] Taillard E., Parallel taboo search techniques for the job shop scheduling problem, ORSA Journal on Computing 6 (1994), 108–117.

[28] Voss S., Tabu search: Applications and prospects, in: D.Z. Du, P.M. Pardalos (Eds.), Network Optimization Problems, World Scientific (1993).

[29] Wolpert D.H., Macready W.G., No Free Lunch Theorems for Optimization, IEEE Trans. Evolutionary Computation 1(1), (1997), 67–82.