Applying sequential and parallel programming to solve a non-linear transportation problem

Tatiana PAȘA, Valeriu UNGUREANU
State University of Moldova
pasa.tatiana@yahoo.com, v.a.ungureanu@gmail.com

***Abstract* — Some aspects related to the structure of a non-linear network transportation problem are investigated. An improvement of an algorithm proposed earlier is provided by the means of sequential and parallel programming and by using several initial admissible solutions. An analysis of different ways of the algorithm implementation in the Wolfram Language is done. Obtained testing results on problems with different dimensions and complexities are presented.**

***Index Terms* — non-linear programming, transportation network, optimal solutions, admissible solutions, local minimum, global minimum.**

I. INTRODUCTION

The transportation network problem is a widespread problem in the planning of product shipments, the location of the production sites or standard problem in the design of communication networks such as the electricity, gas and water networks.

A transportation network is given by a series of points and links between these points, but, as it often happens, we do not always have a direct link between two points, which means that we need to cross some intermediary points to reach the destination. Beside the fact that the functions that describe the transport costs are concave, there can also be restrictions on the amount of flow that can be transported from one point to another, so we can have restrictions of the type "no less than" or "no more than".

As a basis for solving various practical problems, especially the economical ones, are the informational technologies, which serve as a tool for obtaining the optimum solutions. When the computations needed for obtaining the optimum solution are complex and/or the problem is very big, i.e. a big number of data and/or many variables, we have to construct such a method that not only finds the optimum solution, but also does it as fast as possible.

This leads us to the idea of implementing parallel programming when sequential programming doesn't give the desired result. At the same time, we must also take into consideration that parallel computing has its limits in the implementation. Jack Worlton (1986) studying the limits of parallel computations assumed that a parallel program consists of:

- processing sections distributed on different processors;
- synchronizing sections;
- sections that are executed sequentially and constitute an overhead.

Therefore, depending on the case, we can parallelize a part of the algorithm but only rarely the whole algorithm.

II. PROBLEM FORMULATION AND TEOREICAL BASIC CONCEPTS

Let us consider the transportation network [4, 5] described by the graph $G = (V, E)$, $|V| = n$, $|E| = m$. A real bounded function of supply and demand $q = V \to R$ is defined on the finite set of its vertices V. Concave cost functions $\varphi_e(x_e)$ which depend on arcs flow are defined for each arc. We need to determine such a flow x^* that minimizes a non-linear objective function

$$F(x) = \sum_{e \in E} \varphi_e(x_e)$$

It is required to solve the non-linear optimization problem:

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

where X is a set of admissible flows on G described by the following system:

$$\sum_{e \in E^+(v)} x(e) - \sum_{e \in E^-(v)} x(e) = q(v)$$

with both non-negativity constraints and constraints on the transportation capacities of arcs $l(e) \le x(e) \le u(e)$, for all $e \in E$.

Definition: A transportation network [6] is an oriented graph $G = (V, E)$, without loops, which satisfies the following properties:

1. There is a vertex (source) $v_0 \in V$ which has only outgoing edges;
2. There is a vertex (destination, sink) $v_t \in V$ which has only incoming edges;
3. For each arc it is associated a value $c(e)$, for any $e \in E$, named capacity of the arc.

Definition: A flow in a network is a function $f: E \to \mathbb{R}$ which satisfies the following properties:

1. Capacity constraint: For all $e \in E$ the condition $f(e) \le c(e)$ is satisfied;
2. Skew symmetry: For all $(v_1, v_2) \in V$ the condition $f(v_1, v_2) = -f(v_2, v_1)$ is satisfied;

3. Flow conservation: $\sum_{x\in E^+(v)} x(e) - \sum_{x\in E^-(v)} x(e) = 0$, where $E^+(t)$ is the set of edges that enter $t \in V$ and $E^-(s)$ - that exit $s \in V$.

Theorem: The transportation network problem with concave function of cost φ_e for any e ∈ E is NP-complete.

So, we can say that the transportation network problem with concave cost functions can be solved using finite algorithms that study all non-cyclic subgraphs to which a flow is associated and the size of the function is calculated. In [3], the algorithm for solving the transportation network problem is described for the case when the flow on the network is bounded by an upper and lower value and the cost functions, $\varphi_e(x_e)$ for all $e \in E$, are piecewise concave functions because all concave functions can be approximated with some error to a series of linear functions. Based on the several tests performed on networks of different sizes, it was concluded that the algorithm doesn't always give the optimum solution. This is due to the fact that the obtained optimum solution depends on the initial solution of the system with which the algorithm begins its execution and which can be obtained differently depending on the method used to solve the system, that will depend on the size and complexity.

Theorem: There is a bijective correspondence between the admissible basic solutions of a system of linear equations and the vertices of a polyhedral set of admissible solutions of this system. Each vertex of the polyhedral set is a basic solution.

Based on the above results, the algorithm that solves the transportation network problem with concave cost functions should be modified to start from m admissible solutions and choose the best among the obtained optimum solutions. In this case we will surely get the best optimum solution, regardless of the structure complexity, the size of the graph that describes the transportation network or the concave cost functions.

Another aspect of the problem formulated above is the notion of an optimum solution for non-linear problems with concave cost functions that has to be minimized in the constraints that have to be satisfied by such solution.

Definition: A function f is concave on an interval, if for all x and y from the interval and all $\alpha \in [0,1]$ the following is true: $f\big((1-\alpha)x + \alpha y\big) \geq (1-\alpha)f(x) + \alpha f(y)$.

In this paper, concave functions are non-decreasing piecewise functions, defined on the interval $[\mathbf{0}, +\infty]$. They describe the cost of shipping the product along an arc.

Definition: For a real function $f: D \in \mathbb{R}^m \longrightarrow \mathbb{R}$ with m real variables, a point $y = (y_1, y_2, \dots, y_m) \in D \subset \mathbb{R}$ is called a local minimum of the function if there exists a neighborhood V of the point y such that $f(x_1, x_2, .., x_m) \geq f(y_1, y_2, .., y_m)$, for all $(x_1, x_2, .., x_m) \in V \cap D$.

Definition: Let $f: D \in \mathbb{R}^m \longrightarrow \mathbb{R}$ be a real function with m real variables. A point $y = (y_1, y_2, \dots, y_m) \in D \subset \mathbb{R}$ is called global minimum of the function if $f(x_1, x_2, .., x_m) \geq f(y_1, y_2, .., y_m)$ holds for all $(x_1, x_2, .., x_m) \in D$.

As it is known, for a problem of non-linear programming often an optimum solution is obtained as a result of determining a series of solutions and the user is the one that puts restrictions when to stop this series and to consider an approximate optimal solution as the final. Usually this decision is made depending on the available time, computer hardware or the complexity of computations.

Another aspect of the non-linear optimization problem is that it can have many local minimums which is a barrier in the successful solving of the problem, because most often the algorithms give a local minimum that is not necessarily close enough to the global minimum. A similar situation was also observed in solving the problem formulated above using the algorithm proposed in [1] and improved as described in [2]. In this paper, we aim to increase the possibility of obtaining the global optimal solution.

Traditionally, we used software written for serial computation because problem is broken into a discrete series of instructions which are executed sequentially one after another on a single processor.

Parallel programming means utilizing several computing hardware for the simultaneous execution of the sequences broken in discrete parts that can be solved concurrently. Each sequence consists from a series of instructions and all instructions from each part execute simultaneously on different cores or processors all of which work through the computation at the same time.

Parallel programming implies using computing resources this way:

- a single computer with multiple processors or cores;
- an arbitrary number of such computers connected by a network;
- a combination between a computer with multiple processors or cores and an arbitrary number of such computers connected by a network.

We can highlight two important advantages of parallel programming over sequential programming:

- enables solving complex and/or big problems that cannot be solved using a single computer;
- by executing the instructions in parallel we save time and money.

We may highlight that the primary objective of parallel computing is to increase the available computation power for faster application processing. The application server sends a computation or processing request that is distributed in small components, which are executed on each processor or core.

According to Flynn's taxonomy of computer architectures [8], computation systems can be classified based on the flow of executed instructions and the flow of data to be processed. Each of these characteristics can have a unique or multiple state. This way we can define the following classes:

- SISD (Single Instruction, Single Data) – a single processing unit, at each step executes a single instruction (sequence of instructions) that operates with

a single input data. This model involves the application of sequential algorithms.

- SIMD (Single Instruction, Multiple Data) – a single control unit that manages N identical processors with their own memory that, at each step, executes the same instruction using multiple input data. For the communication between the processors Shared Memory or Interconnection Network will be used.
- MISD (Multiple Instruction, Single Data) – is a model in which there are N processors with their own control unit and shared memory. At each step the processors execute different instruction using the same input data.
- MIMD (Multiple Instruction, Multiple Data) – this model uses N processors that operate with N input data and N instructions. So, at each step each processor solves different problems with different input data. The communication between processors is made using Shared Memory.

The best system to use in the context of the formulated problem is SISD or SIMD depending on the dimensions of the network and the necessary time to obtain the optimum solution.

III. THE IMPROVED ALGORITHM, ANALISYS AND RESULTS

We can improve the algorithm by repeating the steps to determine the optimal solution for a few initial solutions and then to select the best one, which will be the optimal solution. We will also examine the possibility of the minimum attaining for several initial/starting points.

We will now explain the possibility of improving the algorithm in [3] using sequential programming and parallel programming. In both cases the initialization of data, i.e. obtaining the set of initial solutions, will be done the same way, therefore there will be no restrictions on the method used or the type of programming used in the implementation.

3.1 Sequential programming

As mentioned above, the computation system SISD is applied in the implementation of sequential algorithms because it satisfies the condition that a singe processing unit executes a singe sequence of instructions which in our case will be the sequence of determining the optimum solution. On the other hand, this model operates with a single data, which in our case will be the initial solution from which will start the algorithm to compute the optimum solution.

Below we describe the algorithm, that will be implemented using sequential programming.

I. Initialization of the data

Step 1.

Construct a table containing k admissible solutions of the system:

$$\begin{cases} \sum_{e\in E^+(v)} x(e) - \sum_{e\in E^-(v)} x(e) = q(v), & for\ all\ v \in V \\ x(e) \geq 0 & for\ all\ v \in V \end{cases}$$

which will be the initial solutions for obtaining the table of optimal solutions.

II. For every element of the table we obtain an optimal solution:

Step 2.

Determine the value of the function in the point:

$$F(x^0) = \sum_{e\in E} \varphi_e(x^0(e))$$

and compute the value of the coefficients:

$$C_e = \begin{cases} \dfrac{\varphi_e(x^0(e))}{x^0(e)}, & x^0(e) > 0 \\ F_e'(0), & x^0(e) = 0 \end{cases}$$

for every $e \in E$.

Step 3.

Solve the linear transport problem:

$$min \rightarrow z(x) = \sum_{e\in E} C_e x(e)$$

$$\begin{cases} \sum_{e\in E^+(v)} x(e) - \sum_{e\in E^-(v)} x(e) = q(v), & for\ all\ v \in V \\ x(e) \geq 0 & for\ all\ v \in V \end{cases}$$

and obtain the optimum solution $x^1 = \left(x^1(e_1), x^1(e_2), \dots, x^1(e_m)\right)$.

Step 4.

Compare the values $z(x^1)$ and $F(x^0)$. If $z(x^1) < F(x^0)$ or $z(x^1) = F(x^0)$ and $x^1 \neq x^0$ substitute x^0 with x^1 and go to *Step 2*. If $z(x^1) > F(x^0)$ or $z(x^1) = F(x^0)$ and $x^1 = x^0$ then the optimal solution of the non-linear transport problem is considered the value $x^* = x^0$, the value $F(x^0)$ and $x^* = x^0$ that correspond to it is preserved; go to the next initial solution.

III. Obtain the optimum solution of the problem

Step 5.

Compare the objective function's values obtained for different initial points and determine the minimal value; save it as the optimal solution and eliminate duplicates. **STOP**.

Wolfram Language [7] makes it easy to implement the algorithm. The code is compact, easy-to-read, it is easy to define new variables and functions.

To obtain an initial solution with which the program will start according to the algorithm, we use the *FindInstace[]* standard function which solves the system of equations. It provides the non-negative solutions and as a result we obtain one of the set of possible solutions on the basis of which a linear function is obtained. *LinearPrograming[]* is a standard function that solves the problem of linear programming.

All considered problems were solved by generating 100 initial admissible solutions and the optimal solution was selected from the values obtained based on these initial solutions. A comparison between the obtained solutions and the execution time of the algorithm in each case was made as we can see in Table 1.

TABLE I. EXECUTION TIME

Nr. of arcs (m)	For m initial solutions	For 100 initial solutions	*Minimize*
6	0.0781	0.6718	0.1280

7	0.9375	0.8282	0.2332
8	0.2187	1.9055	0.5189
9	0.4531	3.7220	1.1229
10	1.1562	9.9054	2,7257
11	1.9375	15.1700	17.0920
12	4.4531	4.4531	32.3139
13	10.0625	68.0372	72.2206
14	16.4063	99.1259	68.9724
15	40.6250	236.5117	81.7750
16	56.1250	313.3157	345.6110
17	140.9530	727.777	308.3290
18	333.2340	1599.7834	1827.4400

3.2 Parallel programming

As we can see, the same sequence of instructions is executed for every initial solution obtained in step 1. In this context we can assume that we have the right use the computing system SIMD that utilizes a single control unit that manages N processors such that each executes the sequence of instructions described in steps 2-4 of the algorithm. At the same time, as input data will serve a different initial solution. The exchange of data between the processors will be accomplished using a common memory where the results will be stored, which will be used later to complete step 5.

Even if we will apply parallel programming to improve the execution time of the algorithm, there exists a limit to the growth of the execution speed of the implemented system. In 1967, Gene Amdahl introduced the law regarding the execution rate and proposed a formula for the speed limit for parallel structures in relation to sequential structures.

The Wolfram Language [7] provides a uniquely integrated and automated environment for parallel computing. With zero configuration, full interactivity, and seamless local and network operation, the symbolic character of the Wolfram Language allows immediate support of a variety of existing and new parallel programming paradigms and data-sharing models. As with sequential programming, Wolfram Language can be applied to implement parallel programming. Among the standard functions can be named *$KernelCount* that returns the number of active kernels for parallel programming. For the synchronization of the kernels *WaitAll[{pid$_1$, pid$_2$, ..., pid$_n$}]* is used. *Parallelize[expr]* evaluates *expr* using automatic parallelization. *ParallelEvaluate[expr]* evaluates the expression *expr* on all available parallel kernels and returns the list of results obtained. The same function receive as input data on which kernels to execute the expression, in this case the function has the following template *ParallelEvaluate[expr, {ker$_1$, ker$_2$,... }]* or we can specify on which kernel the expression will be evaluated using the following template *ParallelEvaluate[expr, kernel]*. *ParallelSubmit[expr]* submits *expr* for evaluation on the next available parallel kernel and returns an expression representing the submitted evaluation.

Applying standard functions allows us to improve the execution time of the algorithm, but this will be better visible for solving problems of big dimensions. As mentioned in [3], the standard function used to obtain the optimum solution (sequentially) give a good result, but for problems with more than 18 arcs the execution time exceeds an hour.

IV. CONCLUSION

From the analysis of the results of the algorithm testing based on a series of problems of different dimensions, we can formulate the following conclusions:

1. The algorithm gives solutions as good as *Minimize[]* and better than *NMinimize[]*, as it gives an optimum solution only in 25% of cases;
2. The execution time increases with the number of initial solutions from which the algorithm starts, but also with the increase of the graph that describes the network;
3. The algorithm offers more solutions for obtained optimum, which in real life gives the possibility to choose the correct strategy in decision making;
4. To manage the execution time of the algorithm, we must carefully choose the number of initial solutions to be built. From the observed from the tests, the number of initial solutions must be at least equal to m, i.e. the number of variables the solution vector has. This will surely give a better final solution than using only one initial solution;
5. Using parallel programming improves the execution time;
6. The execution time using parallel programming doesn't improve proportionally with the number of processors used, because there are computations that cannot be parallelized and must be executed sequentially.

In the most cases, the number of optimal solutions remains approximately the same even the number of initial solutions with which the algorithm operates is increased.

REFERENCES

[1] Pasha T., Lozovanu D., *An algorithm for solving the transport problem on network with concave cost functions on flow of edges*. Computer Science Journal of Moldova, vol. 10, no 3, Kishinev (2002), pp. 341-347.

[2] Paşa T., Ungureanu V., *Solving the transportation problem with piecewise-linear concave cost function on edge flows*, A 20-a Conferinţă a SPSR, AFA "Henri Coandă", Departamentul de Ştiinţe Fundamentale şi Management, Braşov, România, 28-29 aprilie 2017, Editura ASF, p.37

[3] Paşa T., Ungureanu V., *Wolfram Mathematica as an enviroment for solving concave network transportation problems*. Proceeding CMSM4, The Fourth Conference of Mathematical Society of the Republic of Moldova dedicated to the centenary of Vladimir Andrunachevici (1917 - 1997), 28 iunie – 2 iulie 2017, Institute of Mathematics and Computer Science, Academy of Scieces of Moldova, Chişinău (2017), p. 429 – 432.

[4] Гольштейн Е. Г., Юдин Д. Б. *Задачи линейного программирования транспортного типа*. М.: Наука, 1969.

[5] Ермольев Ю. М., Мельник И. И. *Экстремальные задачи на графах*. Киев: Наукова думка, 1968.

[6] R. Trandafir, *Modele și algoritmi de optimizare*, AGIR, București, 2004.
[7] S. Wolfram, *An elementary introduction to the Wolfram Language.* Friesens, Manitoba, Canada, 1-st edition, 2016.
[8] Flynn, M.J. *Some computer organizations ant their effectiveness*, IEEE Transactions on Computers, C-21 (9), 1972, pp. 948-960.