

Article

# Evaluating Typical Algorithms of Combinatorial Optimization to Solve Continuous-Time Based Scheduling Problem

Alexander A. Lazarev <sup>\*</sup>, Ivan Nekrasov <sup>\*,†</sup> and Nikolay Pravdivets <sup>†</sup>

Institute of Control Sciences, 65 Profsoyuznaya Street, 117997 Moscow, Russia; pravdivets@ya.ru

<sup>\*</sup> Correspondence: jobmath@mail.ru (A.A.L.); ivannekr@mail.ru (I.N.); Tel.: +7-495-334-87-51 (A.A.L.)

<sup>†</sup> These authors contributed equally to this work.

Received: 22 February 2018; Accepted: 12 April 2018; Published: 17 April 2018



**Abstract:** We consider one approach to formalize the Resource-Constrained Project Scheduling Problem (RCPSP) in terms of combinatorial optimization theory. The transformation of the original problem into combinatorial setting is based on interpreting each operation as an atomic entity that has a defined duration and has to be resided on the continuous time axis meeting additional restrictions. The simplest case of continuous-time scheduling assumes one-to-one correspondence of resources and operations and corresponds to the linear programming problem setting. However, real scheduling problems include many-to-one relations which leads to the additional combinatorial component in the formulation due to operations competition. We research how to apply several typical algorithms to solve the resulted combinatorial optimization problem: enumeration including branch-and-bound method, gradient algorithm, random search technique.

**Keywords:** RCPSP; combinatorial optimization; scheduling; linear programming; MES; Job Shop

## 1. Introduction

The Resource-Constrained Project Scheduling Problem (RCPSP) has many practical applications. One of the most obvious and direct applications of RCPSP is planning the fulfilment of planned orders at the manufacturing enterprise [1] that is also sometimes named Job Shop. The Job Shop scheduling process traditionally resides inside the Manufacturing Execution Systems scope [2] and belongs to principle basic management tasks of any industrial enterprise. Historically the Job Shop scheduling problem has two formal mathematical approaches [3]: continuous and discrete time problem settings. In this paper, we research the continuous-time problem setting, analyze its bottlenecks, and evaluate effectiveness of several typical algorithms to find an optimal solution.

The continuous-time Job Shop scheduling approach has been extensively researched and applied in different industrial spheres throughout the past 50 years. One of the most popular classical problem settings was formulated in [4] by Manne as a disjunctive model. This problem setting forms a basic system of restrictions evaluated by different computational algorithms depending on the particular practical features of the model used. A wide overview of different computational approaches to the scheduling problem is conducted in [5,6]. The article [5] considers 69 papers dating back to the XX century, revealing the following main trends in Job Shop scheduling:

- Enumerating techniques
- Different kinds of relaxation
- Artificial intelligence techniques (neural networks, genetic algorithms, agents, etc.)

Artificial intelligence (AI) techniques have become mainstream nowadays. The paper [6] gives a detailed list of AI techniques and methods used for scheduling.

From the conceptual point of view, this paper deals with a mixed integer non-linear (MINLP) scheduling problem [5] that is relaxed to a combinatorial set of linear programming problems due to the linear “makespan” objective function. As a basic approach we take the disjunctive model [4]. A similar approach was demonstrated in [7] where the authors deployed the concept of parallel dedicated machines scheduling subject to precedence constraints and implemented a heuristic algorithm to generate a solution.

## 2. Materials and Methods

In order to formalize the continuous-time scheduling problem we will need to introduce several basic definitions and variables. After that, we will form a system of constraints that reflect different physical, logical and economic restrictions that are in place for real industrial processes. Finally, introducing the objective function will finish the formulation of RCPSp as an optimization problem suitable for solving.

### 2.1. Notions and Base Data for Scheduling

Sticking to the industrial scheduling also known as the Job Shop problem, let us define the main notions that we will use in the formulation:

- The enterprise functioning process utilizes resources of different types (for instance machines, personnel, riggings, etc.). The set of the resources is indicated by a variable  $R = \{R_r\}$ ,  $r = 1, \dots, |R|$ .
- The manufacturing procedure of the enterprise is formalized as a set of operations  $J$  tied with each other via precedence relations. Precedence relations are brought to a matrix  $G = \langle g_{ij} \rangle$ ,  $i = 1, \dots, |J|$ ,  $j = 1, \dots, |J|$ . Each element of the matrix  $g_{ij} = 1$  iff the operation  $j$  follows the operation  $i$  and zero otherwise  $g_{ij} = 0$ .
- Each operation  $i$  is described by duration  $\tau_i$ . Elements  $\tau_i, i = 1, \dots, |J|$ , form a vector of operations’ durations  $\rightarrow \tau$ .
- Each operation  $i$  has a list of resources it uses while running. Necessity for resources for each operation is represented by the matrix  $Op = \langle op_{ir} \rangle$ ,  $i = 1, \dots, |J|$ ,  $r = 1, \dots, |R|$ . Each element of the matrix  $op_{ir} = 1$  iff the operation  $i$  of the manufacturing process allocates the resource  $r$ . All other cases bring the element to zero value  $op_{ir} = 0$ .
- The input orders of the enterprise are considered as manufacturing tasks for the certain amount of end product and are organized into a set  $F$ . Each order is characterized by the end product amount  $v_f$  and the deadline  $d_f$ ,  $f = 1, \dots, |F|$ . Elements inside  $F$  are sorted in the deadline ascending order.

Using the definitions introduced above, we can now formalize the scheduling process as residing all  $|J|$  operations of all  $|F|$  orders on the set of resources  $R$ . Mathematically this means defining the start time of each operation  $i = 1, \dots, |J|$  of each order  $f = 1, \dots, |F|$ .

### 2.2. Continuous-Time Problem Setting

The continuous-time case is formulated around the variables that stand for the start moments of each operation  $i$  of each order  $f$ :  $x_{if} \geq 0$ ,  $i = 1, \dots, |J|$ ,  $f = 1, \dots, |F|$  [3]. The variables  $x_{if} \geq 0$  can be combined into  $|F|$  vectors  $\vec{x}_f \geq 0$ . The main constraints of the optimization problem in that case are:

- Precedence graph of the manufacturing procedure:

$$G\vec{x}_f \geq (\vec{x}_f + \vec{\tau}), \quad f = 1, \dots, |F|.$$

- Meeting deadlines for all orders

$$x_{|J|f} + \tau_{|J|} \leq d_f, \quad f = 1, \dots, |F|.$$

The constraints above do not take into consideration the competition of operations that try to allocate the same resource. We can distinguish two types of competition here: competition of operations within one order and competition of operations of different orders. The competition of operations on parallel branches of one order is considered separately by modelling the scheduling problem for a single order (this is out of the scope of this paper). The main scope of this research is to address the competition of operations from different orders. Assuming we have the rule (constraint) that considers the competition of operations within one and the same order let us formalize the competition between the operations of different orders. Consider we have a resource, which is being allocated by  $K$  different operations of  $|F|$  different orders.

For each resource  $r = 1, \dots, |R|$  let us distinguish only those operations that allocate it during their run. The set of such operations can be found from the corresponding column of the resource allocation matrix  $Op = \langle op_{ir} \rangle, i = 1, \dots, |J|, r = 1, \dots, |R|$ .

$$x^r = x|col_r(Op) = 1.$$

The example for two competing operations of two different orders is shown in Figure 1.

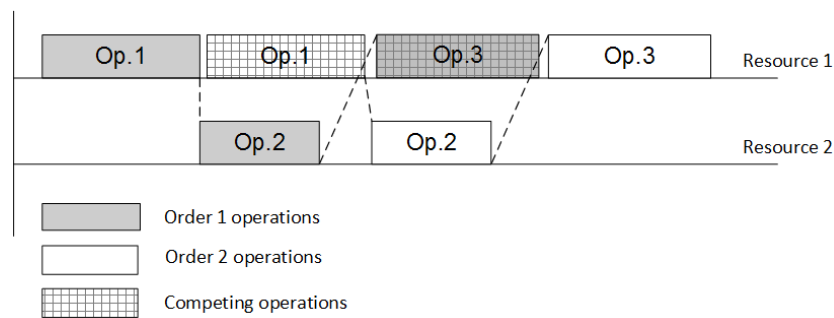


Figure 1. Competing operations on one resource.

For each resource  $r = 1, \dots, |R|$  each competing operation  $i = index(x^r) = 1, \dots, K$  of order  $f = 1, \dots, |F|$  will compete with all other competing operations of all other orders, i.e., going back to the example depicted in Figure 1 we will have the following constraint for each pair of operations:

$$x_{i\varphi} \geq x_{j\psi} + \tau_j \text{ XOR } x_{j\psi} \geq x_{i\varphi} + \tau_i,$$

where indexes are as follows  $i, j = 1, \dots, K; \varphi, \psi = 1, \dots, |F|; \varphi \neq \psi$ . Implementing an additional Boolean variable  $c_k \in \{0, 1\}$  will convert each pair of constraints into one single formula:

$$c_k \cdot x_{i,\varphi} + (1 - c_k) \cdot x_{j,\psi} \geq c_k \cdot (x_{j\psi} + \tau_j) + (1 - c_k) \cdot (x_{i\varphi} + \tau_i).$$

From the above precedence constraints, we can form the following set of constraints for the optimization problem:

$$g_{i,j} \cdot x_{i,f} \geq x_{i,f} + \tau_i, \quad f = 1, \dots, |F|, \quad i = 1, \dots, |J|, \quad j = 1, \dots, |J|, \quad (1)$$

$$x_{|J|f} + \tau_{|J|} \leq d_f, \quad f = 1, \dots, |F|, \quad (2)$$

$$c_k \cdot x_{i,\varphi} + (1 - c_k) \cdot x_{j,\psi} \geq c_k \cdot (x_{j\psi} + \tau_j) + (1 - c_k) \cdot (x_{i\varphi} + \tau_i), \quad \varphi, \psi = 1, \dots, |F|, \quad \varphi \neq \psi, \quad (3)$$

$$x_{if} > 0, \quad f = 1, \dots, |F|,$$

$$c_k \in \{0, 1\}, \quad k = 1, \dots, K.$$

where the variables  $x_{if} > 0, f = 1, \dots, |F|, i = 1, \dots, |J|$  are continuous and represent the start times of each operation of each order; and variables  $c_k \in \{0, 1\}, k = 1, \dots, K$  are Boolean and represent the position of each competing operation among other competitors for each resource.

### 2.3. Combinatorial Optimization Techniques

From the mathematical point of view, the resulting basic problem setting belongs to linear programming class. The variables  $c_k \in \{0, 1\}, k = 1, \dots, K$  form a combinatorial set of subordinate optimization problems [8]. However, we must suppress that the combinatorial set  $c_k \in \{0, 1\}, k = 1, \dots, K$ , is not full as the sequence of operations within each order is dictated by manufacturing procedure. Consequently, the number of combinations is restricted only to those variants when the operations of different orders trade places without changing their queue within one order. Let us research possible ways to solve the given combinatorial set of problems. In this article, the optimization will be conducted with “makespan” criterion formalized with the formula:

$$\sum_{f=1}^{|F|} x_{|J|f} \rightarrow \min.$$

#### 2.3.1. Enumeration and Branch-and-Bound Approach

Enumerating the full set of combinations for all competing operations is an exercise of exponential complexity [9]. However, as we have mentioned above the number of combinations in constraint (3) is restricted due to other constraints (2) and (1). The precedence graph constraint (1) excludes the permutations of competing operations within one order making the linear problem setting non-feasible for corresponding combinations. The way we enumerate the combinations and choose the starting point of enumeration also influences the flow of computations significantly. The common approach to reducing the number of iterations in enumeration is to detect the forbidden areas where the optimal solution problem does not exist (e.g., the constraints form a non-feasible problem or multiple solutions exist). The most widely spread variant of the technique described is branch-and-bound method [10] and its different variations.

Let us choose the strategy of enumerating permutations of operations in such a way that it explicitly and easily affords to apply branch-and-bound method. The starting point of the enumeration will be the most obvious positioning of orders in a row as shown in Figure 2. We start from the point where the problem is guaranteed to be solvable—when all orders are planned in a row—which means that the first operation of the next order starts only after the last operation of the previous order is finished. If the preceding graph is correct and the resources are enough that means the single order can be placed on the resources that are totally free (the problem is solvable—we do not consider the unpractical case when the graph and/or resources are not enough to fulfill a single order). Infeasibility here can be reached only in the case when we start mixing the orders. This guarantees that the linear programming problem is solvable, however, this combination is far from being optimal/reasonable.

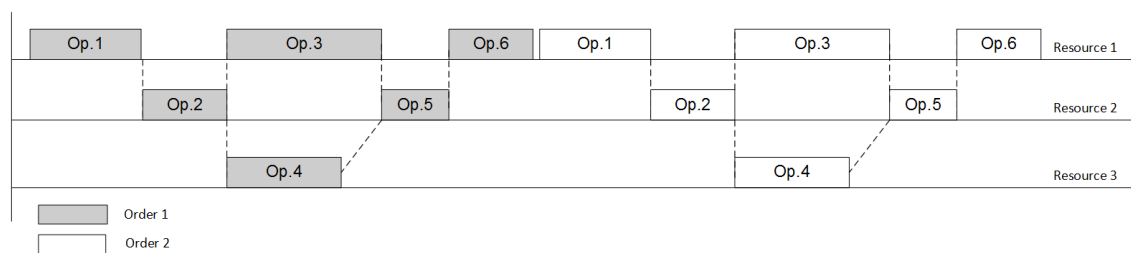


Figure 2. Initial state: all orders are placed in a row.

The enumeration is organized by shifting the operations of the last order to the left in such a way that they trade their places with previous competing operations of preceding orders (see Figure 3). As soon as the operation reaches its leftmost feasible position and the current combination corresponds to a linear programming problem with no solution we roll back to the previous combination, stop shifting operations of the last order and switch to the operations of the previous order. After the enumeration for orders on the current resource is over, we proceed the same iterations with the next resource. The formal presentation of the designed branch-and-bound algorithm is described by Procedure 1.

**Procedure 1: branch-and-bound**

1. **Find the initial solution** of the LP problem (1)–(3) for the combination  $c_k \in \{0, 1\}$ ,  $k = 1, \dots, K$  that corresponds to the case when all orders are in a row (first operation of the following order starts only after the last operation of preceding order is completed).
2. **Remember** the solution and keep the value of objective function  $\Phi$  as temporarily best result  $Opt = \Phi$ .
3. **Select the last order**  $f = |F|$ .
4. **Select the resource**  $r = |R|$  that is used by last operation  $i = |J|$  of the precedence graph  $G$  of the manufacturing process.
5. **The branch**  $\langle i, r, f \rangle$  is now formulated.
6. **Condition:** Does the selected resource  $r$  have more than one operation in the manufacturing procedure that allocates it?
  - 6.1. **If yes then**

**Begin** evaluating branch  $\langle i, r, f \rangle$

**Shift** the operation  $i$  one competing position to the left

**Find** the solution of the LP problem (1)–(3) for current combination and calculate the objective function  $\Phi$

**Condition:** is the solution feasible?

**If feasible then**

**Condition:** Is the objective function value  $\Phi$  better than temporarily best result  $Opt$ ?

**If yes then**

**save current solution** as the new best result  $Opt = \Phi$ .

**End of condition**

**Switch to the preceding operation**  $i = i - 1$  of currently selected order  $f$  for the currently selected resource  $r$ .

Go to the p. 5 and **start evaluating the new branch**

**If not feasible then**

Stop evaluating branch  $\langle i, r, f \rangle$

Switch to the preceding resource  $r = r - 1$

Go to the p. 5 and **start evaluating the new branch**

**End of condition:** is the solution feasible?

**End of condition:** p. 6
7. **Switch to the preceding order**  $f = f - 1$
8. **Repeat** pp. 4–7 **until** no more branches  $\langle i, r, f \rangle$  are available
9. **Repeat** pp. 3–8 **until** no more feasible shifts are possible for all operations in all branches.

Iterating combinations for one order on one resource can be considered as evaluating one branch of the algorithm [10]. Finding the leftmost position for an operation is similar to detecting a bound. Switching between branches and stopping on bounds is formalized with a general branch-and-bound procedure.

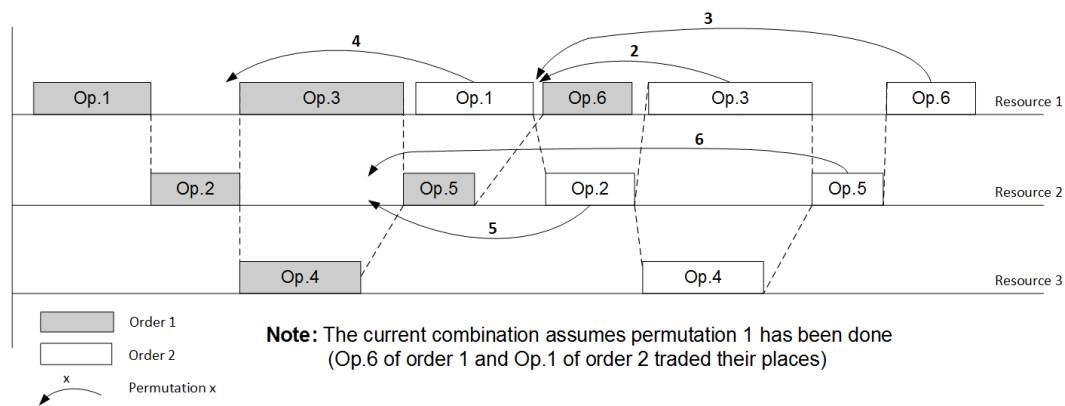


Figure 3. Permutations of competing operations in branch-and-bound algorithm.

### 2.3.2. Gradient-Alike Algorithm

The opposite of the mentioned above approach would be searching for a local minimum among all combinations with a gradient algorithm [11]. Here, we will compute an analogue of the derivative that is used to define the search direction at each point [12]. ‘Derivative’ for each competing operation is calculated by trying to shift it by one position to the left. In case the objective function reduces (or increases for maximization problems) we shift the operation to the left with the defined step (the step ratio is measured in a number of position operation skips moving to the left). The formal presentation of the designed gradient-alike algorithm is described by Procedure 2.

#### Procedure 2: gradient algorithm

1. **Find the initial solution** of the LP problem (1)–(3) for the combination  $c_k \in \{0, 1\}, k = 1, \dots, K$  that corresponds to the case when all orders are in a row (first operation of the following order starts only after the last operation of preceding order is completed).

2. **Remember** the solution and keep the value of objective function  $\Phi$  as temporarily best result  $Opt = \Phi$ .

3. **Select the last order**  $f = |F|$ .

4. **Select the resource**  $r = |R|$  that is used by first operation  $i = 1$  of the sequence graph of the manufacturing process.

5. **The current optimization variable for gradient optimization is selected**—position of operation  $i$  of the order  $f$  on the resource  $r < i, r, f >$ .

6. **Condition:** Does the selected resource  $r$  have more than one operation in the manufacturing procedure that allocates it?

6.1. **If yes then**

**Begin** optimizing position  $< i, r, f >$

6.1.1. **Set the step of shifting** the operation to maximum (shifting to the leftmost position).

$D(i, r, f) = \max$

6.1.2. **Find the ‘derivative’ of shifting** the operation  $i$  to the left

**Shift** the operation  $i$  one position to the left

**Find** the solution of the LP problem (1)–(3) for current combination and calculate the objective function  $\Phi$

**Condition A:** is the solution feasible?

**If feasible then**

**Condition B:** Is the objective function value  $\Phi$  better than temporarily best result  $Opt$ ?

**If yes then**

We found the optimization direction for position  $< i, r, f >$ , proceed to p. 6.1.3

**If not then**

No optimization direction for the current position  $\langle i, r, f \rangle$   
**stop** optimizing position  $\langle i, r, f \rangle$   
**switch** to the next operation  $i = i + 1$   
**go to p 6 and repeat search for position**  $\langle i + 1, r, f \rangle$

**End of condition B**

**If not feasible then**

No optimization direction for the current position  $\langle i, r, f \rangle$   
**stop** optimizing position  $\langle i, r, f \rangle$   
**switch** to the next operation  $i = i + 1$   
**go to p 6 and repeat search for position**  $\langle i + 1, r, f \rangle$

**End of condition A**

6.1.3. **Define the maximum possible optimization step** for the current position  $\langle i, r, f \rangle$ , initial step value  $D(i, r, f) = \max$

**Shift** the operation  $i$  left using the step  $D(i, r, f)$ .

**Find** the solution of the LP problem (1)–(3) for current combination and calculate the objective function  $\Phi$

**Condition C:** Is the solution feasible and objective function value  $\Phi$  better than temporarily best result  $Opt$ ?

**If yes then**

**save current solution** as the new best result  $Opt = \Phi$   
**stop** optimizing position  $\langle i, r, f \rangle$   
**switch** to the next operation  $i = i + 1$   
**go to p 6 and repeat search for position**  $\langle i + 1, r, f \rangle$

**If not then**

reduce the step twice  $D(i, r, f) = \frac{D(i, r, f)}{2}$  and repeat operations starting from p. 6.1.3

**End of condition C**

**Switch to the next operation**  $i = i + 1$ , go to p. 6 and optimize position  $\langle i, r, f \rangle$

7. **Switch to the preceding resource**  $r = r - 1$

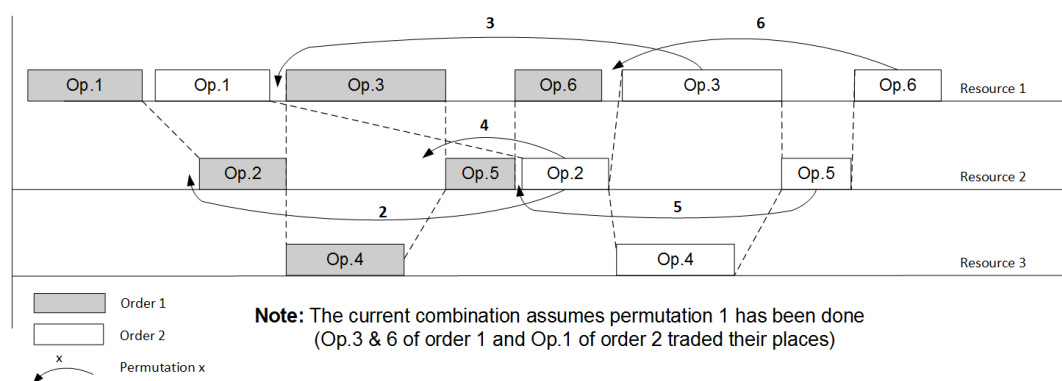
8. **Repeat** pp. 5–7 for currently selected resource  $\langle i, r - 1, f \rangle$

9. **Switch to the preceding order**  $f = f - 1$

10. **Repeat** pp. 4–9 for currently selected order  $\langle i, r, f - 1 \rangle$ .

11. **Repeat** pp. 3–10 **until** no improvements and/or no more feasible solutions exist.

The optimization variables in gradient algorithm represent positions of all competing operations relative to each other. The maximum optimization step for each iteration is detected on-the-fly by trying to shift the current operation to the leftmost position (as it is shown in Figure 4) that shows the LP problem (1)–(3) is solvable and the objective function is improved.



**Figure 4.** Permutations of competing operations in gradient algorithm.

### 3. Results of the Computational Experiment

As a reference trial example let us take the following scheduling problem. The manufacturing procedure contains 13 operations that require 6 different resources. The schedule of the manufacturing procedure for a single order is shown in Figure 5. In this example, we want to place two orders at the depicted resources and start evaluating from the point ‘all orders in a row’ (see Figure 6).

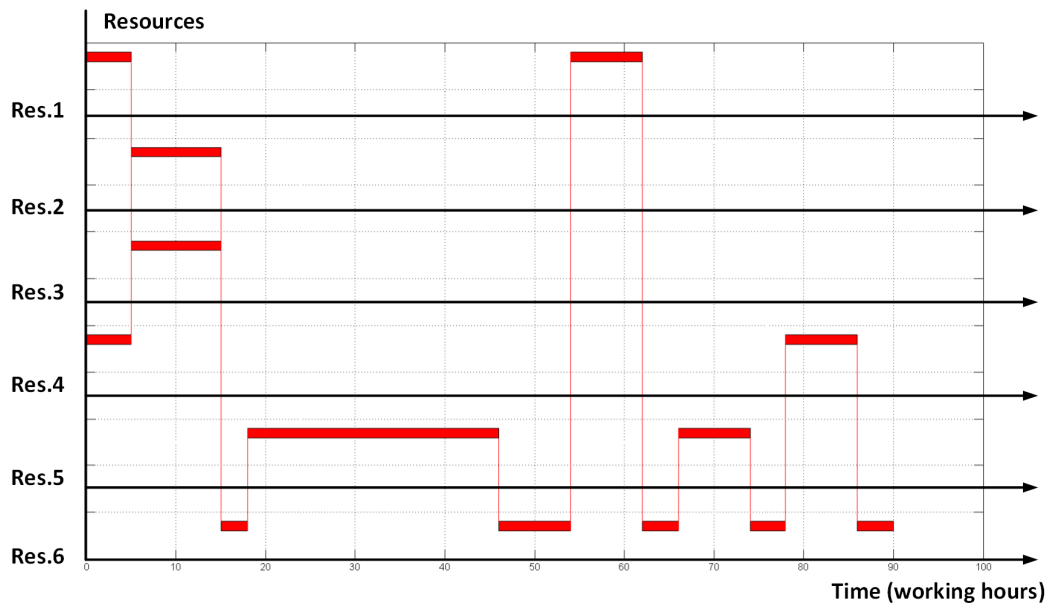


Figure 5. Gantt diagram for single order manufacturing procedure.

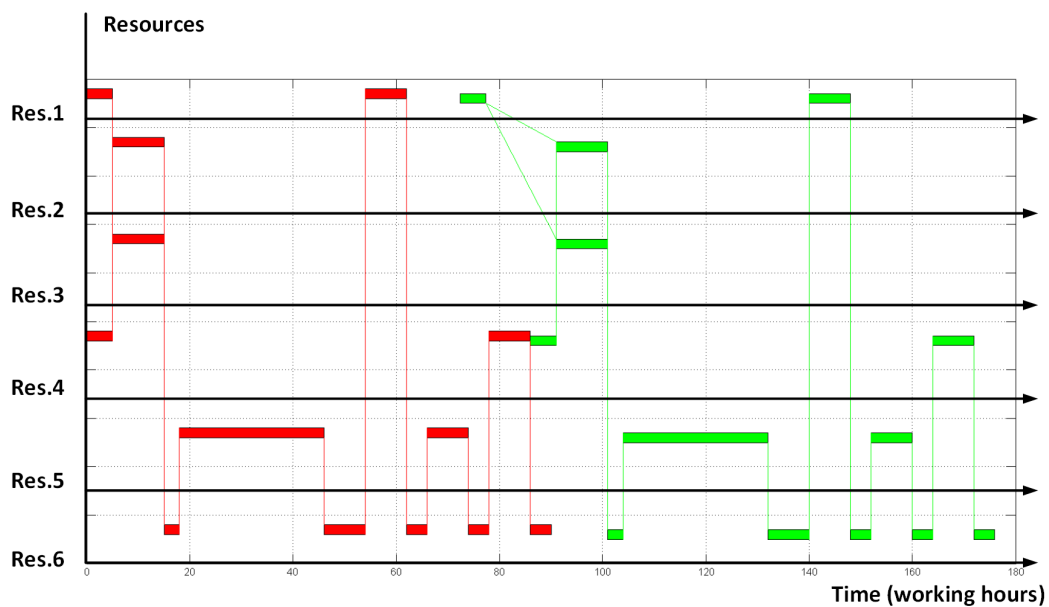


Figure 6. Gantt diagram for starting point of two orders schedule optimization.

The results of evaluating branch-and-bound and gradient algorithm are placed in the following Table 1. The resulting schedules are depicted in Figure 7 (for branch-and-bound algorithm) and Figure 8 (for gradient algorithm).



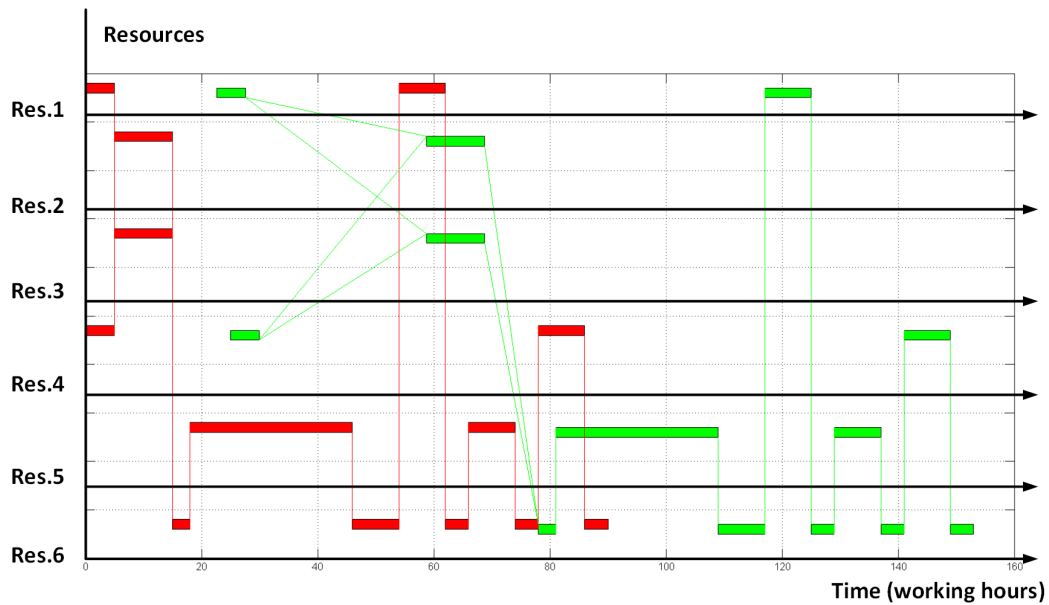


Figure 7. Branch-and-bound optimization result.

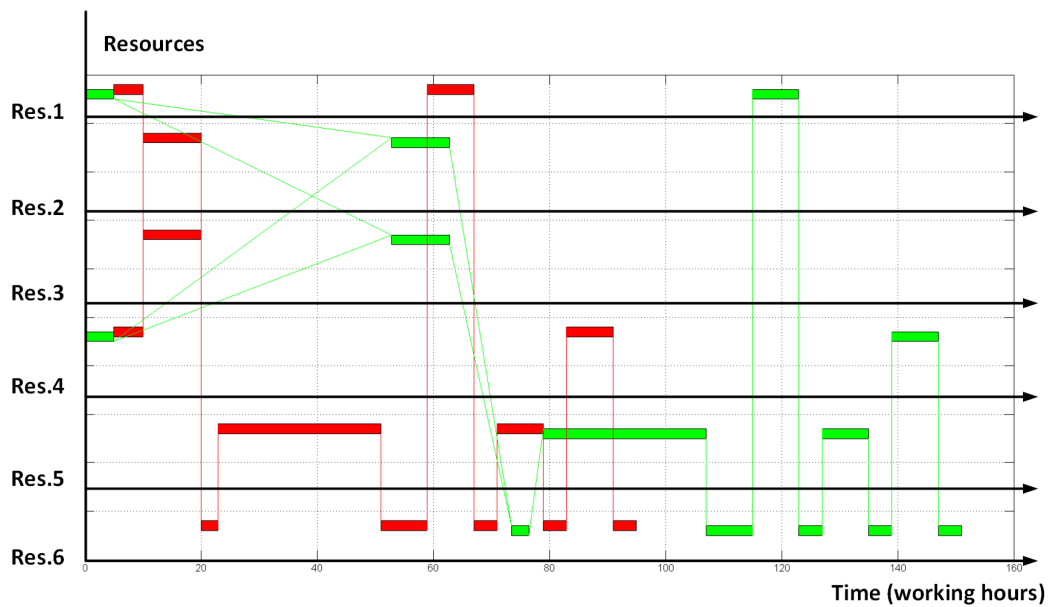


Figure 8. Gradient algorithm optimization result.

Results of analysis gives us understanding that for a modest dimensionality scheduling problem both considered approaches are applicable. However, as the number of operations in manufacturing procedure grows and the number of orders increases, we will experience an enormous growth of algorithm iterations for enumeration (branch-and-bound) technique and the gradient-alike algorithm will obviously detect local optimum as the best achievable solution (which means the optimization results will differ more and more compared to the digits in Table 1). Rising difference between the results is shown in Table 2.

Expanding the previous problem for a procedure of 500 operations in 2 orders we will get the results presented in Table 3.

**Table 1.** Algorithms test results: small number of operations (13 operations).

Algorithm	Resulting “Makespan” Objective Function	Times of Orders Finished		Full Number of Operations Permutations (Including Non-Feasible)	Number of Iterated Permutations	Number of Iterated Non-Feasible Permutations	Calculation Time, Seconds
		Order 1	Order 2				
B&B	235	86	149	318	25	11	3.6
gradient	238	91	147	318	18	4	2.8

**Table 2.** Results for 2 orders with number of operations increasing.

Number of Operations	Algorithm	Resulting “Makespan” Objective Function	Times of Orders Finished		Number of Iterated Permutations	Number of Iterated Non-Feasible Permutations	Calculation Time, Seconds
			Order 1	Order 2			
25	B&B	348	134	214	71	29	5.4
	gradient	358	139	219	36	9	3.1
50	B&B	913	386	527	201	59	25
	gradient	944	386	558	84	22	3.7
65	B&B	1234	488	746	490	70	112
	gradient	1296	469	826	126	66	11.2
100	B&B	1735	656	1079	809	228	288
	gradient	1761	669	1092	677	83	237

**Table 3.** Algorithms test results: increased number of operations (500 operations).

Algorithm	Resulting “Makespan” Objective Function	Times of Orders Finished		Full Number of Operations Permutations (Including Non-Feasible)	Number of Iterated Permutations	Number of Iterated Non-Feasible Permutations	Calculation Time, Seconds
		Order 1	Order 2				
B&B	4909	1886	3023	Unknown	≈1200 (Manually stopped)	Unknown (≈10% of total)	Manually stopped after ≈7 h
Gradient	4944	1888	3056	Unknown	≈550	Unknown (≈30% of total)	≈3 h
Random (gradient extension)	Trying implementing randomized algorithm led to high computation load of PC with no reasonable estimation of calculation time. Time to start first gradient iteration from feasible point took enumeration of more than 1000 variants.						

#### 4. Discussion

Analyzing the computational experiment result, we come to a classical research outcome that to solve the global optimization problem effectively we need to find a more computationally cheap algorithm that gives a solution closer to global optimum. Let us try several standard approaches to improve the solution with non-exponential algorithm extensions.

##### 4.1. Random Search as an Effort to Find Global Optimum

As mentioned above, the gradient algorithm, being more effective from the computational complexity point of view, affords to find only local suboptimal solutions [12]. A typical extension to overcome this restriction would be random search procedure. The main idea of this modification is to iterate gradient search multiple times going out from different starting points [13]. In case the starting points are generated randomly we can assume that the more repeating gradient searches we do the higher the probability of finding a global optimum we achieve. There was some research conducted in this area whose outcome recommends how many starting points to generate in order to cover the problem's acceptance region with high value of probability [14]. According to [14] the acceptable number of starting points is calculated as

$$N \cdot \dim(\Phi),$$

where  $N = 5 \dots 20$ , and  $\dim(\Phi)$  is the dimensionality of optimization problem being solved, i.e., for our case this is the number of all competing operations of all orders on all resources. The result of applying random search approach is represented in Table 2.

The main barrier for implementing the random search procedure for the combinatorial scheduling problem is generating enough feasible optimization starting points. As we can see from the results in Table 2, number of failures to generate feasible starting point is much higher than the quantity of successful trials. Leaning upon the results of enumeration algorithm in Table 1 we can assume that the tolerance regions for optimization problem (1)–(3) are very narrow. Even from the trial example in Table 1 we see that the number of feasible iterations (25) collect less than 10 percent of all possible permutations (318) which leaves us very low probability of getting a feasible initial point for further gradient optimization. Thus, we can make a conclusion that 'pure' implementation of random search procedure will not give a huge effect but should be accompanied with some analytic process of choosing feasible initial points of optimization. Such a procedure may be based on non-mathematical knowledge such as industry or management expertise. From our understanding, this question should be investigated separately.

#### 5. Conclusions

Research of a continuous-time scheduling problem is conducted. We formalized the scheduling problem as a combinatorial set [8] of linear programming sub problems and evaluated typical computational procedures with it. In addition to the classical and estimated resulting conflict between the "complexity" and "locality" of optimization algorithms we came to the conclusion that the classical approach of randomization is unhelpful in terms of improving the locally found suboptimal solution. The reason here is that the scheduling problem in setting (1)–(3) has a very narrow feasibility area which makes it difficult to randomly detect a sufficient number of starting points for further local optimization. The efficiency of random search might be increased by introducing a martial rule or procedure of finding typical feasible starting points. The other effective global optimization procedures are mentioned in a very short form and are left for further authors' research. They are:

- Genetic algorithms. From the first glance evolutionary algorithms [15] should have a good application case for the scheduling problem (1)–(3). The combinatorial vector of permutations  $c_k \in \{0, 1\}$ ,  $k = 1, \dots, K$ , seems to be naturally and easily represented as a binary crossover [15] while the narrow tolerance region of the optimization problem will contribute to the fast convergence of the breeding procedure. Authors of this paper leave this question for further research and discussion.
- Dynamic programming. A huge implementation area in global optimization (and particularly in RCPSP) is left for dynamic programming algorithms [16]. Having severe limitations in amount and time we do not cover this approach but will come back to it in future papers.

The computation speed of the high dimension problem using an average PC is not satisfactory. This fact forces authors to investigate parallel computing technologies. Future research assumes adoption of created algorithms to a parallel paradigm, for instance, implementing map-reduce technology [17].

**Acknowledgments:** This work was supported by the Russian Science Foundation (grant 17-19-01665).

**Author Contributions:** A.A.L. conceived conceptual and scientific problem setting; I.N. adopted the problem setting for manufacturing case and designed the optimization algorithms; N.P. implemented the algorithms, performed the experiments and analyzed the data.

**Conflicts of Interest:** The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, and in the decision to publish the results.

## References

1. Artigues, C.; Demasse, S.; Néron, E.; Sourd, F. *Resource-Constrained Project Scheduling Models, Algorithms, Extensions and Applications*; Wiley-Interscience: Hoboken, NJ, USA, 2008.
2. Meyer, H.; Fuchs, F.; Thiel, K. *Manufacturing Execution Systems. Optimal Design, Planning, and Deployment*; McGraw-Hill: New York, NY, USA, 2009.
3. Jozefowska, J.; Weglarz, J. *Perspectives in Modern Project Scheduling*; Springer: New York, NY, USA, 2006.
4. Manne, A.S. On the Job-Shop Scheduling Problem. *Oper. Res.* **1960**, *8*, 219–223, doi:10.1287/opre.8.2.219.
5. Jones, A.; Rabelo, L.C. Survey of Job Shop Scheduling Techniques. In *Wiley Encyclopedia of Electrical and Electronics Engineering*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 1999. Available online: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.37.1262&rep=rep1&type=pdf> (accessed on 10 April 2017).
6. Taravatsadat, N.; Napsiah, I. Application of Artificial Intelligent in Production Scheduling: A critical evaluation and comparison of key approaches. In Proceedings of the 2011 International Conference on Industrial Engineering and Operations Management, Kuala Lumpur, Malaysia, 22–24 January 2011; pp. 28–33.
7. Hao, P.C.; Lin, K.T.; Hsieh, T.J.; Hong, H.C.; Lin, B.M.T. Approaches to simplification of job shop models. In Proceedings of the 20th Working Seminar of Production Economics, Innsbruck, Austria, 19–23 February 2018.
8. Trevisan, L. *Combinatorial Optimization: Exact and Approximate Algorithms*; Stanford University: Stanford, CA, USA, 2011.
9. Wilf, H.S. *Algorithms and Complexity*; University of Pennsylvania: Philadelphia, PA, USA, 1994.
10. Jacobson, J. *Branch and Bound Algorithms—Principles and Examples*; University of Copenhagen: Copenhagen, Denmark, 1999.
11. Erickson, J. *Models of Computation*; University of Illinois: Champaign, IL, USA, 2014.
12. Ruder, S. *An Overview of Gradient Descent Optimization Algorithms*; NUI Galway: Dublin, Ireland, 2016.
13. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*, 3rd ed.; Massachusetts Institute of Technology: London, UK, 2009.
14. Kalitkyn, N.N. *Numerical Methods*; Chislennyye Metody; Nauka: Moscow, Russia, 1978. (In Russian)
15. Haupt, R.L.; Haupt, S.E. *Practical Genetic Algorithms*, 2nd ed.; Wiley-Interscience: Hoboken, NJ, USA, 2004.

16. Mitchell, I. *Dynamic Programming Algorithms for Planning and Robotics in Continuous Domains and the Hamilton-Jacobi Equation*; University of British Columbia: Vancouver, BC, Canada, 2008.
17. Miner, D.; Shook, A. *MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems*; O'Reilly Media: Sebastopol, CA, USA, 2013.



© 2018 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).